The Development of Random Generators of Weather and Industrial Pipelines Data using Parametric and Non-Parametric Approaches

by

Mubarak Khamis AL-Alawi

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Construction Engineering and Management

Department of Civil and Environmental Engineering

University of Alberta

ABSTRACT

Construction projects are unique and complex in nature. They are associated with many challenges regarding the randomness, complexity, and interdependency related to the operation/process, the environment hosting the operation, and the product being constructed. These challenges are also common in the area of simulation and modeling of a construction operation. Research in this field demands real life data but unfortunately the availability of such data is one of the major challenges. Also, the random generation of complex construction data structures that contain correlated attributes make it difficult to replicate real systems behaviors. The objective of this research is to investigate alternative techniques that can be used to randomly generate complex construction data structures while preserving the correlation between their formations' attributes.

This research focuses on two different types of construction-related data: weather data, and industrial pipelines data. A non-parametric approach in the form of bootstrapping technique was applied in the generation of weather data, and its performance was measured against a parametric weather generator constructed in the field of modelling construction operations. The validation results showed that the proposed technique performed in a manner similar to that of the parametric weather generator and outperformed it in some cases. A parametric approach in the form of Markov chain technique was applied to randomly generate industrial pipeline data structures, and its performance was tested against real pipeline data. The validation results showed that the proposed Markov chain model was able to generate an industrial pipeline data structure similar to those in reality. The majority (89%) of generated pipelines shared characteristics with 85.5 % of the original pipelines.

This research demonstrates the application of the developed generators in two areas. The first application modelled an earthmoving operation in oil sand mining and used the weather generator to analyse the effect of temperature on breakdown and repair durations. The second application involved building a pipe-spooling optimization model and used the industrial pipelines data generator to randomly generate instance problems to test the computational efficiency of the optimization's solution algorithm.

DEDICATION

This thesis is dedicated with love, gratitude, and respect to

my parents;
my wife and my children;
my brothers and sisters;
my friends

ACKNOWLEDGMENT

First and foremost all praise and thanks to Allah. I would like to express my sincere gratitude and appreciation to my supervisor Dr. Yasser Mohamed and to my co-supervisor Dr. Ahmed Bouferguene for their guidance, encouragement and unlimited support in my study.

I would like to express my great gratitude and thanks to my doctoral committee: Dr. Aminah Robinson, Dr. Osama Moselhi, and Dr. Vivek Bindiganavile for their valuable comments and suggestions.

I would like to express my great gratitude to Sultan Qaboos University, Oman, for their financial support during my study at the University of Alberta.

Finally, I would like to express my sincere thanks and gratitude to my wife and my children, my father and mother, and my brothers and sisters for their continuous supports, and encouragements.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and problem statement

The construction industry is complex and unique. Research in this industry grows rapidly to overcome its associated problems. Analysing and solving construction-related problems require collecting and using data. However, this data is not readily available. Collecting data plays a critical role in performing reliable research because it represent the nature of real systems. However, real systems' data can be complex, as it may be composed of a set of inter-dependent/correlated attributes. Preserving dependencies between data's attributes is challenging. Furthermore, preserving dependencies limits the available size of real case data which can be used for research. For instance, experimental analysis is normally performed to address and test variations of a real system with respect to changes in the system's formation components or variables. It is used to test new algorithms and procedures to solve real world problems. To provide valuable insight into and information about the system of study, experimental analysis depends greatly on the availability or size of data sets and their quality. Papageriou et al. [1] reported that there are no publicly available benchmark instances on which researchers, in the context of operational research, can test their algorithms. Otto, Otto, and Scholl [2] stated that due to the limited number of real-world problems reported in the literature, generating problem instances using random generators is a valuable source of test data. They also stated that choosing adequate test data sets is a focus of discussions in computational experiment guidelines. In information discovery and analysis systems, having data sets is important to develop test cases to cover hypothetical future scenarios. Privacy or the challenge and cost associated with collecting real data impose the need of generators capable of producing data sets [3]. Jeske et al. [3] emphasize the importance of generating test data sets to support performance studies of statistical and artificial intelligence techniques used in information discovery and system analysis.

In modeling and simulation, Trypula [4] noted that 10% to 40% of the total time required to build a simulation model is attributed to data gathering, cleaning, and validation. Perera and Liyanage [5] reported that the development of simulation models is delayed when the right data are not available in the right format at the right time. Perera and Liyanage [5] also concluded that poor data availability is ranked first among major pitfalls in input data collection. Input modeling is the practice of selecting probability distributions to represent the random nature of a system and that choosing the most appropriate distribution is easier if data are available [6]. Assuming that data are available and the system's objective and formation variables are well-defined, input modeling is performed by applying three steps: (1) selecting different probability distributions for an input model, (2) estimating the parameters of the model, and (3) assessing the goodness of fit. These steps are applied when the system variables are assumed independent from each other (called univariate input modeling). This practice in input modeling is widely applied in the simulation of construction operations. AbouRizk et al. [7] illustrated a numerical technique that can be used to fit beta distributions to sample data for construction engineering and examined its applicability to heavy construction operations. However, system variables may exert interdependency with each other. For instance, construction operations are subjected to uncertainty factors that causes variabilities in their work performance and weather is one of them [8]. In generating, weather variables for modeling construction operations, weather data in each day are represented in the form of a vector, as s in Figure 1- 1, containing hourly/daily weather variables (e.g. precipitation, temperature, relative humidity, etc.), and each variable may have a dependency in daily or hourly manners (dashed arrow) or may exert interdependency (solid arrow) with other variables. Such dependencies add complexity in modeling weather variables but it is important to preserve them in order to achieve a realistic modeling environment [9].

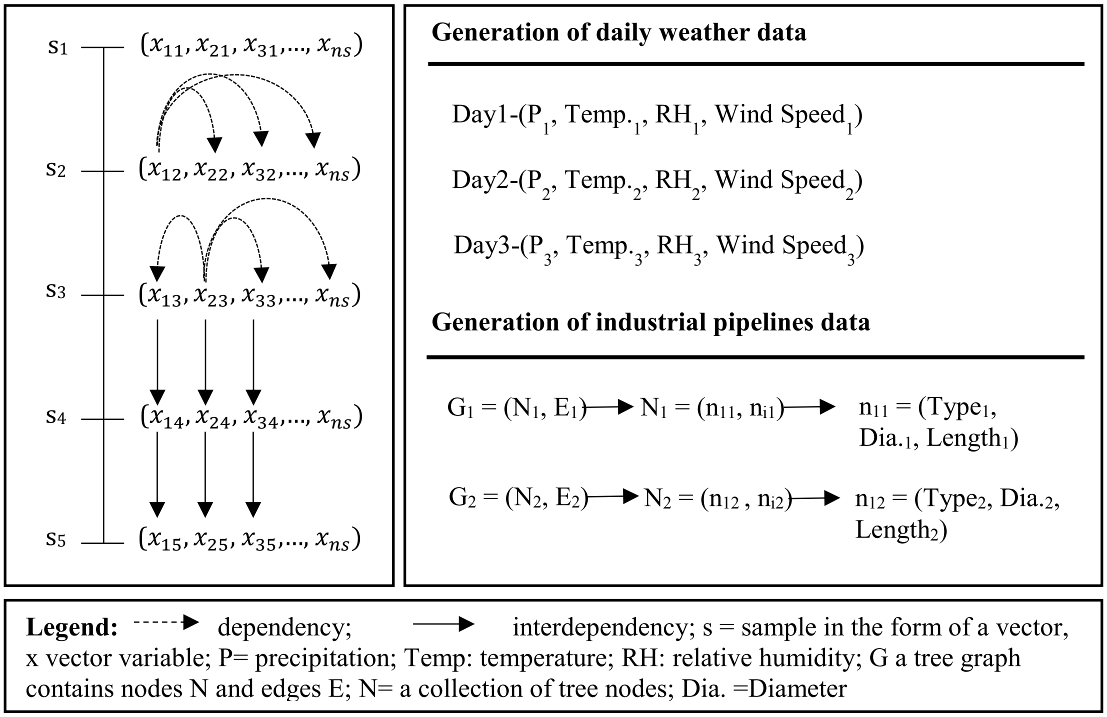

s1 $(x_{11}, x_{21}, x_{31}, \dots, x_{ns})$
s2 $(x_{12}, x_{22}, x_{32}, \dots, x_{ns})$
s3 $(x_{13}, x_{23}, x_{33}, \dots, x_{ns})$
s4 $(x_{14}, x_{24}, x_{34}, \dots, x_{ns})$
s5 $(x_{15}, x_{25}, x_{35}, \dots, x_{ns})$
Generation of daily weather data
Day1-(P_1, $Temp._1$, RH_1, Wind $Speed_1$)
Day2-(P_2, $Temp._2$, RH_2, Wind $Speed_2$)
Day3-(P_3, $Temp._3$, RH_3, Wind $Speed_3$)
Generation of industrial pipelines data
$G_1 = (N_1, E_1) \longrightarrow N_1 = (n_{11}, n_{i1}) \longrightarrow n_{11} = (Type_1, Dia._1, Length_1)$
$G_2 = (N_2, E_2) \longrightarrow N_2 = (n_{12}, n_{i2}) \longrightarrow n_{12} = (Type_2, Dia._2, Length_2)$
Legend: dependency; interdependency; s = sample in the form of a vector, x vector variable; P= precipitation; Temp: temperature; RH: relative humidity; G a tree graph contains nodes N and edges E; N= a collection of tree nodes; Dia. =Diameter

Figure 1- 1 A graphical definition of complex data structure and areas found in construction engineering research

Referring to Perera and Liyanage [5], the right data may not be available in the right format; in this study, we define data format as a data structure. Different applications may require different data structures. For example, in industrial construction projects such as refineries and chemical plants, piping represents a major element. Piping work goes through three major stages: (1) pipe spool fabrication, (2) module assembly, and (3) site installation [10]. In modeling piping stages, each stage may require sets of inputs with each set comprised of vectors containing attributes such as type (nominal), diameter (numerical), length (numerical), and weight (numerical). Considering the diversity and the uniqueness associated with input data is crucial to maintain a proper modeling results [11] [12]. This diversity in the type of data in a single vector creates a challenge in input modeling for optimization studies because (refer to Figure 1- 1) when pipeline data are generated, each pipeline is represented in the form of a tree structure G containing a collection of nodes N and edges E. Each node n in N is presented in the form of a vector containing properties such as the type of pipeline component, its diameter, and length. The connectivity between nodes or the reproduction of the type of node is depedendent on its neighbor, which adds more complexity in modeling such data. Furhermore, unlike the generation of weather variables, pipeline data structure represents a construction product. To maintain realistic data, it is necessary to take into consideration the topological structure in the generation process.

From the above perspective, it is clear that having data available on hand to support research studies is a critical issue. Furthermore, based on the modeled system, different types of data structures may exist. These range from a simple numerical type of data to a complex combinatorial type of data. The main thrust of this study is to investigate the use of mathematical techniques to construct and formulate reliable data generators capable of generating complex construction-based data sets with highly correlated attributes.

1.2 Research objectives

The main objective of this research is to investigate the use of parametric and non-parametric approaches for random generation of construction-related data structures. It focuses on two types of data structures: the weather, and the industrial pipelines data structures. The first one represent a sample of the external factors that affect construction operations. The second is a sample of the complex data structures that characterize construction products. The selection of these two is also affected by the availability of data sets that can be used during modeling and validation phases of the research. The more specific research objectives can be defined as follows:

1. To evaluate the use of a non-parametric approach in the form of bootstrapping technique to randomly generate weather data.
2. To develop and evaluate the use a parametric approach in the form of a Markov chain technique to randomly generate industrial pipelines data.
3. To investigate different validation approaches for testing the accuracy of the proposed techniques.
4. To demonstrate the application of the proposed techniques in construction-related case studies.

1.3 Research methodology

Figure 1- 2 shows the implemented research methodology. This research is split into two parts with a number of steps in each part. The following sections provide a summary of each step.

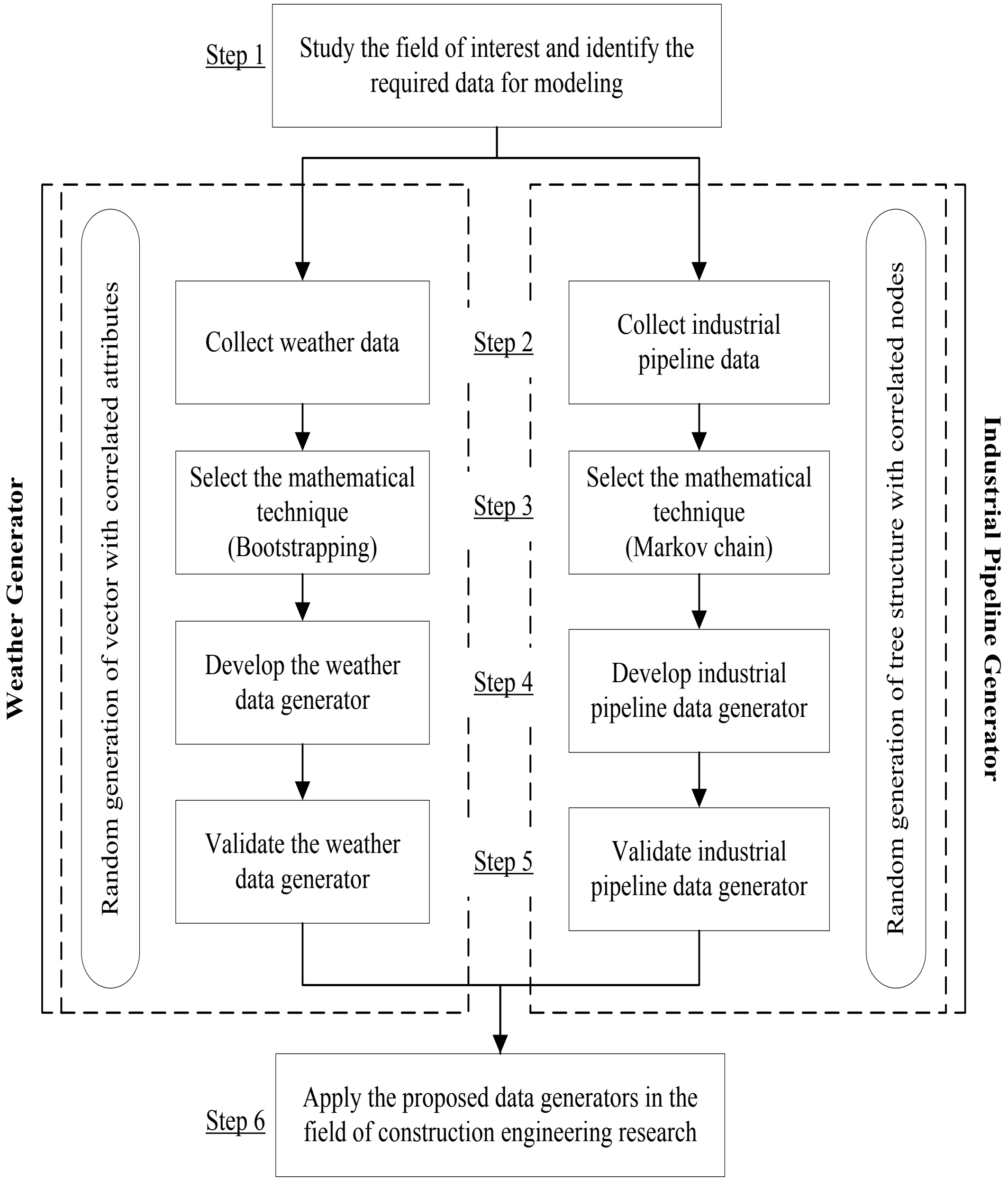

Step 1
Study the field of interest and identify the required data for modeling
Weather Generator
Random generation of vector with correlated attributes
Collect weather data
Select the mathematical technique (Bootstrapping)
Develop the weather data generator
Validate the weather data generator
Step 2
Step 3
Step 4
Step 5
Collect industrial pipeline data
Select the mathematical technique (Markov chain)
Develop industrial pipeline data generator
Validate industrial pipeline data generator
Random generation of tree structure with correlated nodes
Industrial Pipeline Generator
Step 6
Apply the proposed data generators in the field of construction engineering research

Figure 1- 2 Research methodology

Step 1: In this step, the field of interest is studied and the required data to be modeled is identified. As mentioned in the objectives, complex construction data structures are targeted; therefore, two types of data that differ in nature and complexity were selected to be investigated and modeled. The first data is related to weather variables that represent the environment that hosts the construction operation and the second type of data is related to industrial pipeline data that represent a construction product.

Step 2: In this step, data collection is performed. Records of 40 years of historical weather data were collected for modeling weather variables and records from about 1052 pipelines from an industrial construction project were collected to model pipeline data structure. Data collected from different sources usually require cleaning and preparation. More specifically, they require restructuring to serve the modeling objective of the research. In the context of weather data, historical weather records are normally clean and tabulated in the form of hourly or daily records. However, pipeline data are represented in the form of three-dimensional models, which require data extraction, extensive data cleaning, and data restructuring so that the properties of the pipeline tree structures can be analysed.

Step 3: In this step, mathematical techniques that be can be used in modeling and generating weather variables and pipeline data structure are investigated. The selection of what mathematical technique can be used for each type of data depends on the previous step. A non-parametric approach in the form of bootstrapping technique is selected to randomly generate weather variables with replacement, and a Markov chain model is selected to generate the industrial pipeline tree structure randomly.

Step 4: In this step, weather and industrial pipeline data generators are developed and implemented using Python [13]. The weather generator provides data containing weather variables which may affect construction operations. The industrial pipeline data generator provides data containing pipeline components' properties and their connectivity relationships with other components.

Step 5: In this step, a comprehensive three-stage validation process is performed to test the reliability of both the weather and the industrial pipeline data generators in generating realistic data. The performance of both generators is tested using different validation methods. The difficulty associated with this step is related to the validation of pipeline data structures. More specifically, the challenge lies in how to statistically measure the similarity between synthetic and original pipeline tree structures. This challenge is overcome by converting the tree structure of the pipelines to feature vectors capable of preserving the component properties and their unique location in the pipeline structure.

Step 6: Data generators are built for certain objectives/applications. In this step, the use of each data generator (the weather, and the industrial pipeline data generators) is demonstrated in two different applications. The weather generator is implemented in the context of simulation modeling, and the industrial pipeline data generator is implemented in the context of computational efficiency of optimization algorithms.

1.4 Thesis organization

Chapter 2 reviews the effect of weather on the construction operation and justifies the importance of integrating weather effects in modeling construction operations. It also reviews the parametric weather generation approach which has been recently used in the field of construction engineering research to generate weather variables. Its drawbacks are highlighted in this chapter and a non-parametric weather generation approach, in the form of the bootstrapping technique, is proposed. For the purpose of validation and evaluation of the non-parametric weather generation approach, two weather generation models were developed for this chapter: the parametric and the non-parametric weather generation approaches. This chapter also presents a comprehensive evaluation process that includes evaluating the assumptions used in building the models, their generated outputs, and their performances when applied on two weather-sensitive construction models.

Chapter 3 illustrates the application of the weather generator in modeling a construction operation. It highlights the weather effect in earthmoving operations, more specifically in earthmoving mining. In this chapter, mining the earthmoving operation located in Fort McMurray is modeled using distributed simulation with high level architecture (HLA) standards. A weather generator was built and integrated into the simulation model. The simulation model studies the effect of extreme winter temperature on the breakdown and repair duration of trucks and excavators. The weather generator rule was to provide different testing scenarios. At the end of this chapter, the weather scenario's effect on breakdown repair duration is analysed and reported.

Chapter 4 reviews the general input modeling technique. It highlights difficulties associated with modeling the tree structure type of industrial pipelines and proposes a Markov chain model in the generation process. In this chapter, the Markov chain model is used in the branching process of the industrial pipeline data structure. Furthermore, this chapter presents a detailed overview of the industrial pipeline data. It includes data collection, preparation, structuring, and statistical analysis processes that are performed before building the industrial pipelines data generator. As in the weather generation chapter, this chapter applies a comprehensive validation process. This chapter also shows a methodology of converting the industrial pipeline tree structure into a feature vector and demonstrates a three stage-validation process.

Chapter 5 illustrates how to apply the industrial pipeline data generator to test the efficiency of optimization algorithms. This chapter defines an optimization problem in the area of industrial construction, proposes an optimization algorithm, and tests the efficiency of the optimization algorithm using a data test set generated from the industrial pipeline data generator.

Chapter 6 presents the conclusions of this research, contributions, limitations, and future directions.

Chapter 2

Non-Parametric Weather Generator for Modelling Construction Operations: Comparison with the Parametric Approach and Evaluation of Construction-Based Impacts

2.1 Introduction

The construction industry is subject to a wide range of uncontrollable external factors that cause uncertainty in the planning, scheduling, and controlling phases of a project. Among these factors are changing weather conditions, which are environmental factors that significantly influence the efficiency of construction operations. The effect of weather conditions on construction projects is variable and is based on numerous factors, including types of construction, location, and season. Ahuja and Nandakumar [14] have stated that the reliability of project duration forecasting depends on the accuracy of network logic, individual activity duration estimates, and various uncertainty variables in the project environment including weather. Losses in man-hours can also result from changes in weather conditions, with the impacts of weather on labour cost being classified into five groups: (1) bad weather time (describes the scenario where workers are paid, but no work progress is made), (2) reduced productivity (describes the scenario where worker output is reduced and additional paid man-hours are required), (3) repetition of work resulting from damage caused by weather variables such as wind, rain, or ice, (4) stood-off time (describes the scenario where workers are dismissed, absent, or reported late due to bad weather), and (5) a reduced working schedule due to bad weather [15].

Randolph et al. [16]found that 30% of loss in steel operation productivity is due to cold winter temperatures. Kohen and Brown [17]indicated that three-quarters of worker compensation claims during the cold season are due to frostbite-related injuries. To maintain a healthy working environment, the American Conference of Governmental Industrial Hygienists (ACGIH) [18] developed a warm-up schedule for construction trades in cold regions. Productivity is most affected by changes in weather conditions when construction activity is entirely dependent on labour. For example, high wind speeds dramatically exacerbate drops in temperature, making it impossible to sustain a constant labour production rate under these conditions.

In earthmoving operations required for highway construction, weather conditions are a critical factor that must be considered in productivity estimates. Material excavation and hauling activities are sensitive to rainfall and in some instances work may either be stopped or suspended as a result of unworkable soil conditions [19]. Experts in highway construction have indicated that the impact of rainfall is dependent on rainfall amount and timing, as well as on drying conditions. They also reported that an average of 1.5 days of earthmoving productivity is lost when rainfall intensity is between 13-25 mm [19].

Previous studies investigated the effects of weather variables on construction activities. Ahuja and Nandakumar [14] and Kavanaga [20]considered the effect of weather as a percentage in their construction modelling and measured how frequently weather resulted in reduced activity. Moselhi et al. [21] quantified the impact of weather conditions on daily construction activity. El-Rayes and Moselhi [19] presented a decision support system for quantifying the impact of rainfall on productivity and duration of highway construction operations. Wales and AbouRizk [22] and Shahin et al. [23] developed a stochastic weather generator that produces weather variables for use in construction simulation models. Apipattanavis et al. [24] proposed a framework for quantifying and predicting weather-related highway construction delays, which included a weather generator to provide a probabilistic forecast of weather threshold values. Although methodologically different, these investigations followed a similar pattern to build the required models and quantify their impacts on real projects by: (1) studying construction processes, (2) understanding weather impact on processes, (3) determining the weather variables that affect the studied process, (4) searching for source(s) of weather data, (5) selecting a modelling technique, (6) generating weather variables (the generation of weather variables is normally performed by developing a weather generator tool), and, finally, (7) applying the model to a case study project.

Fatichi et al. [25] defined weather generators as numerical tools capable of generating a time-series of climatic variables with statistical properties similar to the observed climate. These generators are used to generate synthetic weather series to help study weather-dependent processes. Depending on the process being modelled, weather generators differ in terms of time steps, single or multiple locations, and number of variables (e.g., temperature, precipitation, and wind speed).

A universal weather generator framework was proposed by Shahin [26] to be used in construction engineering and management research. The framework illustrated the use of the parametric stochastic weather generation approach to generate synthetic weather series with multiple variables. It used a first-order Markov chain model to generate precipitation, a multivariate generation model to generate temperature and relative humidity, and a probability distribution model to generate wind speed. This approach is associated with drawbacks such as the selection of the order of the Markov chain model. Although the first-order Markov chain model is commonly applied to generate precipitation, this selection has been unjustified [27]. Chin [27] analysed 25-years records of precipitation from 100 weather stations in the United States and concluded that the first-order Markov chain model is adequate in resampling the wet and dry spells in the summer season. However, during the winter season, a higher-order Markov chain model was better than the first-order model at re-sampling the wet and dry spells. Chin also concluded that the geographical location of the studied area affects the selection order of the Markov Chain model. Another drawback associated with generating precipitation is the amount generated. The parametric approach samples the amount of precipitation from a probability distribution function. The main challenge associated with this model is its ability to reflect the features found in precipitation data, including bimodality, skewness, and long tail [28]. In addition, the parametric approach assumes weather data to be normally distributed, so that the multivariate generation model can be used to generate temperature and relative humidity variables. However, weather data from different locations may exert different distribution behaviour. Doubrovsky [29] constructed a stochastic weather generator called Met&Roll using the classical approach presented by Richardson [9], and conducted validation by comparing the generated monthly means with observed means from historical weather records. He concluded that weather variables such as solar radiation, maximum temperature, and minimum temperature did not follow a normal distribution. Another drawback associated with the parametric approach used in the universal weather generator framework is created by the gap between the large time scale (on a daily basis) of the generated weather variables and the time scale required by the application at which the weather generator is used. Most construction operations consider the effect of changes in weather conditions on a daily basis. However, other operations, such as earthmoving in the mining industry, which often takes place in cold regions, require hourly weather monitoring. This case adds complexity to the generation of weather variables. Bridging this gap represents a challenging problem in assessing such operations. Although parametric approaches are expected to improve generated weather series, they still have several inadequacies: (1) the choice of model is subjective (e.g., modelling weather variables by fitting them into their distribution independently or using multivariate models) and rarely tested on a site-by-site basis [30], (2) the distribution of weather variables used at one site may not be appropriate for all sites [30], and (3) the multivariate models require data to be normally distributed. In case in which they are not normally distributed, a transformation to normality is required. This is a difficult task that may negatively affect model performance [31].

Detailed records of historical weather data for almost all locations in the world are publicly available. Using such high quality records, it is possible to directly sample realistic weather parameters for different times and locations. Realistic extreme cases can also be generated from these records. This paper illustrates a simplified, non-parametric weather generation approach that uses the classical bootstrapping technique to generate synthetic weather series.

Unlike the parametric approach, the non-parametric approach does not require a theoretical probability distribution function for weather variables. This approach preserves serial dependence between weather variables by using a block-resampling scheme that considers a block of observations as a single observation and generates daily and hourly weather variables. However, the generated weather series in the non-parametric approach is limited by historical records, as simulated samples are selected from available (past) weather data. Therefore, an experiment on both parametric and non-parametric approaches is conducted to highlight differences between both approaches from two perspectives: the generated weather series and their performance when applied on weather-sensitive construction models.

For the purposes of comparison, this experiment uses a weather generator framework developed by Shahin et al. [32] to simulate construction operations. The framework applies the parametric approach to generate weather variables. The parametric approach used by Shahin et al. [32] shares the same drawbacks discussed previously. In addition, wind speed is modelled independently with no correlation to other weather variables and the generated weather data is limited to a daily scale.

This chapter is organized as follows. In Section 2.2, a detailed description of the experiment applied to both parametric and non-parametric weather generators is presented. Section 2.3 describes how both generators are developed. Section 2.4 illustrates a comprehensive weather generation evaluation process, which tests the weather generators' performances from the perspectives of the assumptions applied and outputs generated. It also assesses the generators' performances when applied on construction simulation models. The conclusions of this chapter are outlined in Section 2.5.

2.2 Experimenting with the Parametric and Non-Parametric Approaches

Here, a simplified non-parametric weather generator is developed and its performance compared to a previous weather generator using historical records as a baseline. Figure 2- 1 shows a summary of the study methodology, which begins by selecting the location of study. This step is performed to determine the weather variables that may directly affect construction operation performance at that location. For the purpose of this investigation, Fort McMurray, Alberta is selected as the location of study. However, a different location may be chosen, provided that weather records are available. Fort McMurray is located in the northern part of the province of Alberta, Canada (56°44' N, 111° 23' W) and is characterized by large seasonal temperature differences. Known worldwide for its oil sands, this region has witnessed tremendous industrial activities, including oil extraction, mining, and construction, which are currently driving the province’s economy.

The second step consists of importing historical weather data for the location of study. Most countries have their own meteorological agencies that record and save weather time series. In Canada, Environment Canada maintains historical weather records that can be used to construct a weather generator. Moreover, historical weather data about most locations can be found in the National Climatic Data Center on the National Oceanic and Atmospheric Administration (NOAA) website [33].

After an historical weather database is created, two weather generators are constructed (see Appendix A). The first is constructed using the classical parametric approach and the second is constructed using a non-parametric approach. The two generators’ outputs are evaluated based on a defined testing scenario. For the scenario, two synthetic weather series data sets are created, each corresponding to a 10-year period from both generators. A statistical analysis is performed to compare both datasets against historical records. Comparison with historical records will determine the degree of similarity between the synthetic and the real weather time series. Another weather generator evaluation test will also be conducted. The second test measures how imperfections associated with the weather generators’ outputs affects the results obtained from a model that uses weather series as an input. A similar evaluation was conducted by Dubrovsky et al. [34]. This evaluation assumes that model outputs fed by synthetic weather series should have similar characteristics to those fed by historical records. The discrepancies between outputs from two different sources of input (historical and synthetic) are due to the sensitivity of the model to certain characteristics of weather variables. Accordingly, low discrepancies indicate that weather variables are perfectly reproduced by the weather generator and vice versa.

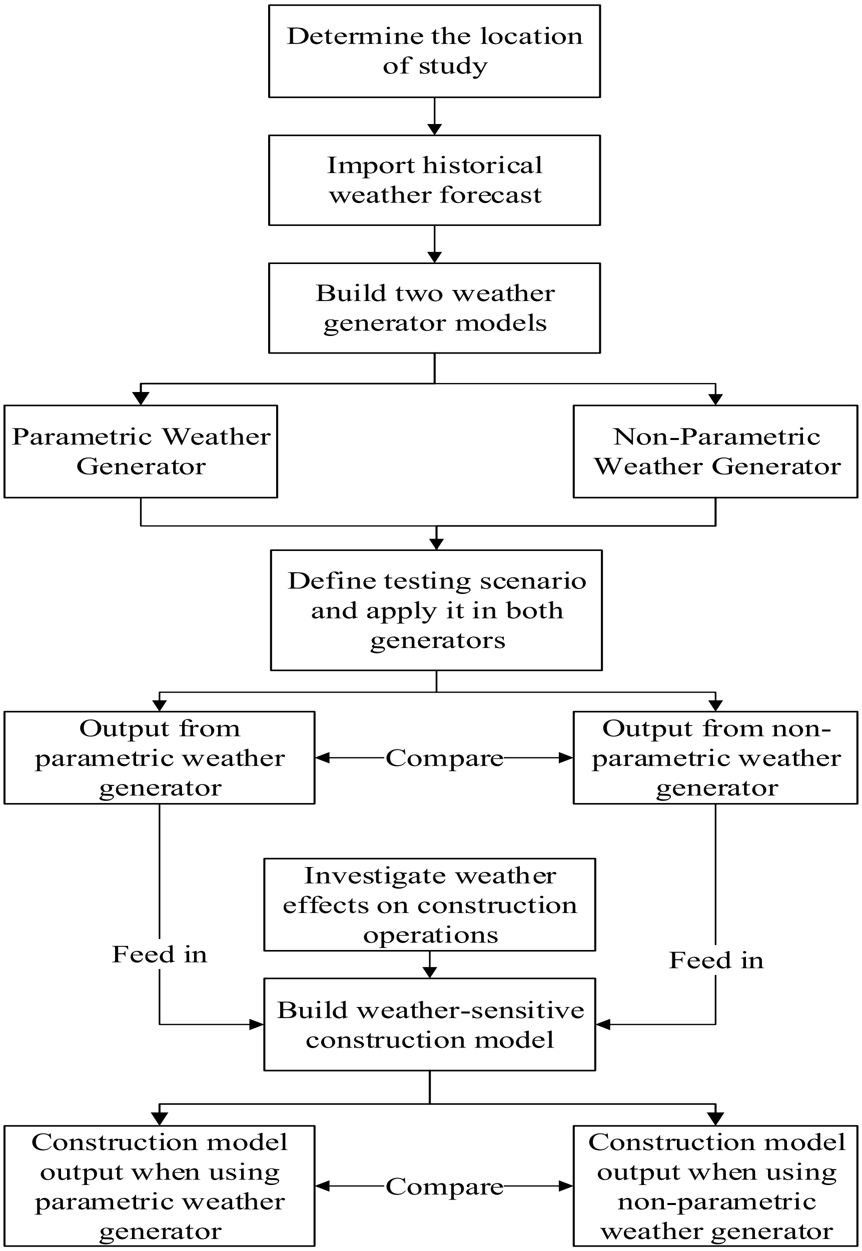

Determine the location of study
Import historical weather forecast
Build two weather generator models
Parametric Weather Generator
Non-Parametric Weather Generator
Define testing scenario and apply it in both generators
Output from parametric weather generator
Compare
Output from non-parametric weather generator
Investigate weather effects on construction operations
Feed in
Feed in
Build weather-sensitive construction model
Construction model output when using parametric weather generator
Compare
Construction model output when using non-parametric weather generator

Figure 2- 1 Study methodology

2.3 Construction of Weather Generators

2.3.1 Parametric weather generator

In classical weather generation, the stochastic relationships underlying meteorological processes are always considered in modelling weather variables. Normally two main relationships are considered in any weather modelling: (1) the time dependence within each variable, and (2) the interdependence among the weather variables [9]. Richardson's [9] stochastic simulation approach to weather generation represents the foundation of most weather modelling studies. Its general generation process flow chart is shown in Figure 2- 2.

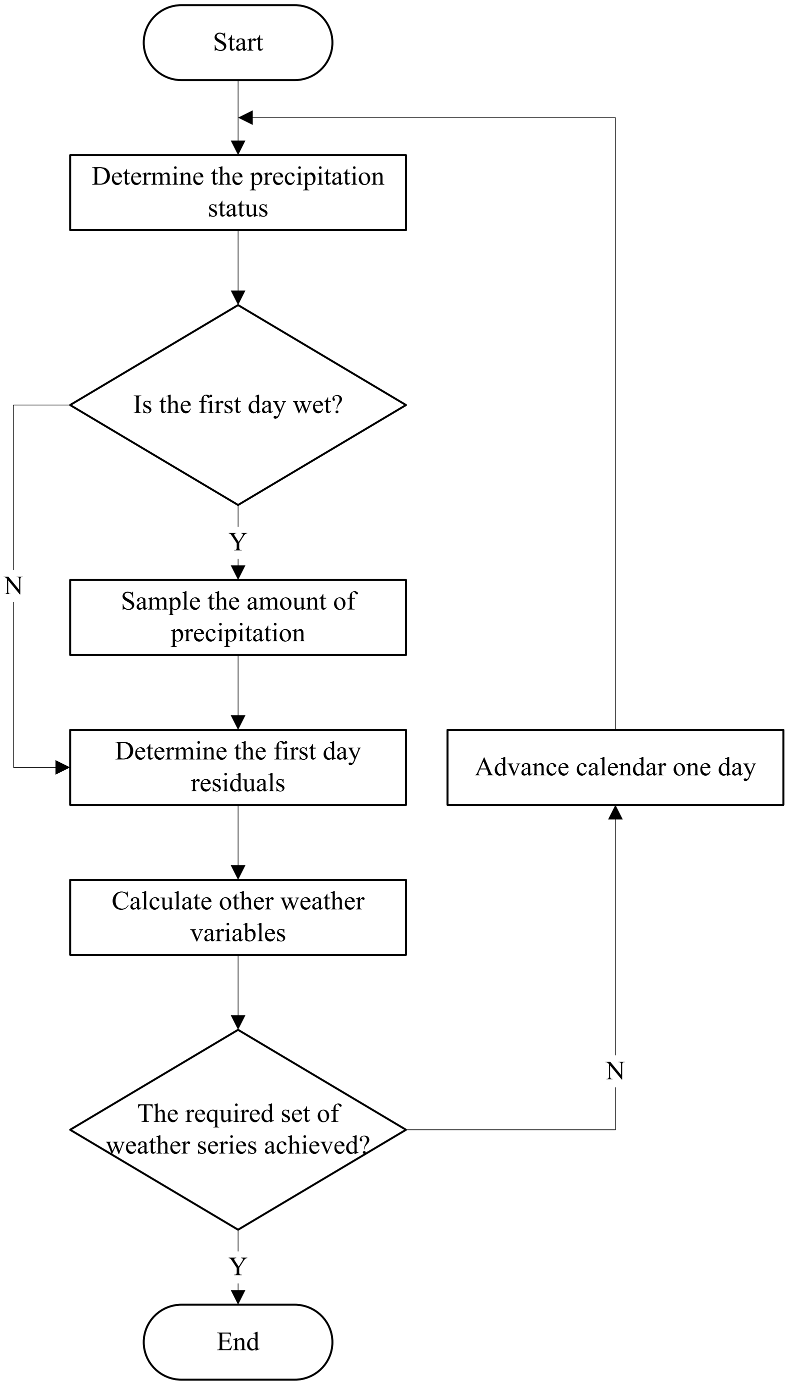

Start
Determine the precipitation status
Is the first day wet?
Y
N
Sample the amount of precipitation
Determine the first day residuals
Calculate other weather variables
The required set of weather series achieved?
N
Advance calendar one day
Y
End

Figure 2- 2 A parametric weather generation flow chart

In Richardson's seminal work, precipitation, which serves to label days as dry or wet, is used to construct any other relevant weather parameter required by the model. The amount of precipitation is determined independently using a distribution function that represents the amount of rainfall throughout a year. Prior to determining the amount of precipitation, a first-order, two-state Markov chain concept is used to describe the occurrence of wet and dry days, as shown in Figure 2- 3.

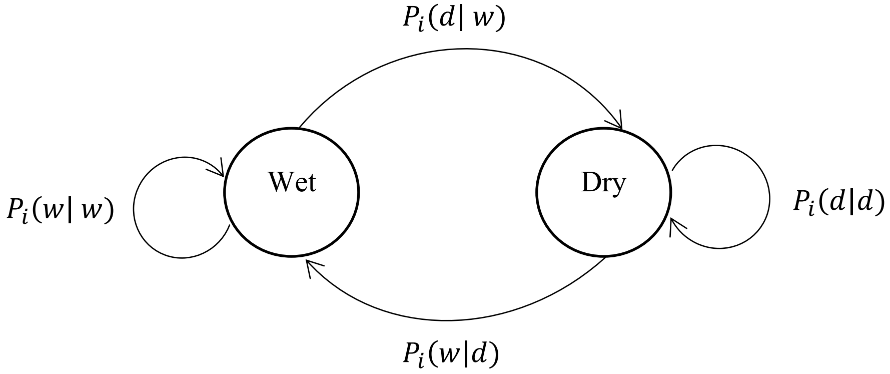


Figure 2- 3 Two-state Markov chain precipitation model

According to Figure 2- 3, given a wet day (or dry day), the conditional transition probability to a dry day (or a wet day) satifies the following equations:

$$\begin{cases} P_i(d\mid w)+P_i(w\mid w)=1 \\ P_i(d\mid d)+P_i(w\mid d)=1 \end{cases} \qquad 2\text{-}1$$

To initialize the computation, the state of the first day (i.e., wet or dry) is determined using the unconditional probability $P_m(w)$ associated with month m (or equivalently $P_m(d) = 1 - P_m(w)$) in conjunction with the algorithm given in equation (2-2):

$$\text{if } \left[r_n \leq P_m(w)\right] \Rightarrow \text{ Wet} \qquad 2\text{-}2$$

In which r_n is a randomly generated number from a uniform distribution. As for the daily precipitation, it is usually modeled independently by means of an appropriately selected distribution function. For instance, Richardson [9] used the exponential distribution function, $f_n(x) = \lambda_n \exp(-\lambda_n x)$, for its simplicity but stated that mixed exponential, $f_n(x) = a_n \exp(-b_n x) + c_n \exp(-d_n x)$, and gamma distribution functions are better at describing the amount of precipitation. Wales and AbouRizk [22]and Shahin [26] used a two-state gamma distribution, due to its flexibility in using two parameters to describe the distribution.

Once the wet or dry day state condition is determined, the other weather variables are calculated using a continuous multivariate stochastic process with daily mean and standard deviations conditional on the day (wet/dry). This technique was described by Yevjevich [35] and it begins by reducing the time series of each variable to time series of residual elements by removing the periodic means and standard deviations. It is performed by first determining the mean and standard deviation for wet and dry days for all variables from the historical weather records. Then, the Fast Fourier Transform method is performed in order to smooth the daily means and standard deviations. Finally, the following equations are utilized to calculate the residual elements:

$$\begin{cases} x_d(i) = \dfrac{X_d^0(i) - \mu_d^0(i)}{\sigma_d^0(i)}, & \text{if} \quad P(d) = 0 \\ x_d(i) = \dfrac{X_d^1(i) - \mu_d^1(i)}{\sigma_d^1(i)}, & \text{if} \quad P(d) > 0 \end{cases} \qquad 2\text{-}3$$

where:

$x_d(i)$ = the residual element of parameter i for day d in the records,

$X_d(i)$ = the value of parameter i for day d in the records,

$\sigma_d^0(i)$ = the periodic standard deviation of parameter i for a dry day d in the historical records,

$\overline{X_d^0}(i)$ = the periodic mean of parameter i for a dry day d in the historical records,

$\sigma_d^1(i)$ = the periodic standard deviation of parameter i for a wet day d in the historical records,

$\overline{X_d^1}(i)$ = the periodic mean of parameter i for a wet day d in the historical records, and

$P(d)$ = the amount of precipitation for day d in the records.

A weakly stationary generating process is then used to generate residual elements of the weather parameters. The weakly stationary generating process used in this approach was defined by Matalas [36] and its equation for n weather parameters is as follows:

$$x_d = \boldsymbol{A}x_{d-1} + \boldsymbol{B}\varepsilon_d \qquad 2\text{-}4$$

where:

x_d= the (nx1) matrix of residual elements for day d for parameters 1 to n,

x_{d-1} = the (nx1) matrix of residual elements for d-1 for parameters 1 to n,

A and **B** = the (nxn) matrices defined so that the correlations within and among the residual series are preserved, and

ε_d = the (nx1) matrix of random components sampled from a standard normal distribution with a mean of 0 and a standard deviation of 1.

This approach implies that the weather parameters are normally distributed and that the serial correlation within each parameter can be described by a first-order linear autoregressive model. Therefore, matrices **A** and **B** may be determined from the following matrix equations:

$$\mathbf{A} = \boldsymbol{M_1 M_0^{-1}} \tag{2-5}$$

$$\boldsymbol{BB^T} = \boldsymbol{M_0} - \boldsymbol{M_1 M_0^{-1} M_1^T} \tag{2-6}$$

where:

$\boldsymbol{M}_0$ = the (nxn) lag0 covariance matrix of the residual series, and

$\boldsymbol{M_1}$ = the (nxn) lag1 covariance matrix of the residual series.

A full description of the construction of parametric weather generators can be found in Shahin [26]. Shahin applied his framework in two different locations, one of which was Fort McMurray. All parameters used in the parametric weather generator are extracted from his work and are listed in Table 2- 1.

Table 2- 1 Parameters used in constructing the parametric weather generator

<table>
<tr><th>Weather variable</th><th>Mathematical model</th><th>Parameter values used in the generation process</th></tr>
<tr><td>Wet and dry states of the day</td><td>First-order two-state Markov chain</td><td>
<table>
<tr><th>Month (m)</th><th>Jan</th><th>Feb</th><th>Mar</th><th>Apr</th><th>May</th><th>Jun</th></tr>
<tr><td>$P_m(w)$</td><td>0.406</td><td>0.369</td><td>0.316</td><td>0.262</td><td>0.338</td><td>0.452</td></tr>
<tr><td>$P_m(w|w)$</td><td>0.536</td><td>0.545</td><td>0.48</td><td>0.429</td><td>0.476</td><td>0.56</td></tr>
<tr><td>$P_m(w|d)$</td><td>0.317</td><td>0.265</td><td>0.24</td><td>0.203</td><td>0.27</td><td>0.364</td></tr>
<tr><th>Month (m)</th><th>Jul</th><th>Aug</th><th>Sept</th><th>Oct</th><th>Nov</th><th>Dec</th></tr>
<tr><td>$P_m(w)$</td><td>0.498</td><td>0.422</td><td>0.412</td><td>0.349</td><td>0.415</td><td>0.41</td></tr>
<tr><td>$P_m(w|w)$</td><td>0.581</td><td>0.548</td><td>0.577</td><td>0.508</td><td>0.574</td><td>0.534</td></tr>
<tr><td>$P_m(w|d)$</td><td>0.416</td><td>0.331</td><td>0.296</td><td>0.264</td><td>0.302</td><td>0.324</td></tr>
</table>
</td></tr>
<tr><td>Precipitation</td><td>Fitted to a two-state gamma distribution</td><td>
<table>
<tr><th>Month</th><th>Jan</th><th>Feb</th><th>Mar</th><th>Apr</th><th>May</th><th>Jun</th></tr>
<tr><td>α</td><td>0.731</td><td>0.736</td><td>0.611</td><td>0.53</td><td>0.46</td><td>0.556</td></tr>
<tr><td>β</td><td>2.085</td><td>1.999</td><td>2.764</td><td>5.151</td><td>7.848</td><td>9.432</td></tr>
<tr><th>Month</th><th>Jul</th><th>Aug</th><th>Sept</th><th>Oct</th><th>Nov</th><th>Dec</th></tr>
<tr><td>α</td><td>0.518</td><td>0.41</td><td>0.431</td><td>0.481</td><td>0.747</td><td>0.666</td></tr>
<tr><td>β</td><td>10.024</td><td>12.795</td><td>9.389</td><td>5.516</td><td>2.52</td><td>2.422</td></tr>
</table>
</td></tr>
<tr><td>Maximum temperature, minimum temperature, maximum relative humidity, and minimum relative humidity</td><td>Weekly stationary generation process</td><td>
$$\mathbf{A} = \begin{bmatrix} \mathbf{0.368} & \mathbf{-0.014} & \mathbf{0.077} & \mathbf{-0.058} \\ \mathbf{0.221} & \mathbf{0.004} & \mathbf{0.004} & \mathbf{0.037} \\ \mathbf{0.085} & \mathbf{-0.005} & \mathbf{0.406} & \mathbf{0.246} \\ \mathbf{0.020} & \mathbf{0.002} & \mathbf{0.095} & \mathbf{0.411} \end{bmatrix}$$
$$\mathbf{B} = \begin{bmatrix} \mathbf{0.923} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0.393} & \mathbf{0.894} & \mathbf{0} & \mathbf{0} \\ \mathbf{-0.016} & \mathbf{-0.005} & \mathbf{0.827} & \mathbf{0} \\ \mathbf{-0.252} & \mathbf{0.135} & \mathbf{0.304} & \mathbf{0.784} \end{bmatrix}$$
The mean and standard deviation of each day are divided into two values representing a wet and a dry status and their values are calculated from the historical weather records.
</td></tr>
<tr><td>Average daily wind speed</td><td>Fitted to a two-state gamma distribution</td><td>
<table>
<tr><th>Month</th><th>Jan</th><th>Feb</th><th>Mar</th><th>Apr</th><th>May</th><th>Jun</th></tr>
<tr><td>α</td><td>2.319</td><td>3.579</td><td>4.522</td><td>6.525</td><td>6.431</td><td>5.904</td></tr>
<tr><td>β</td><td>3.622</td><td>2.519</td><td>2.176</td><td>1 .68</td><td>1.707</td><td>1.634</td></tr>
<tr><th>Month</th><th>Jul</th><th>Aug</th><th>Sept</th><th>Oct</th><th>Nov</th><th>Dec</th></tr>
<tr><td>α</td><td>5.176</td><td>4.693</td><td>4.588</td><td>4.651</td><td>3.794</td><td>2.019</td></tr>
<tr><td>β</td><td>1.735</td><td>1.856</td><td>2.076</td><td>2.215</td><td>2.337</td><td>4.146</td></tr>
</table>
</td></tr>
</table>

2.3.2 Non-parametric weather generator

Most non-parametric methods use resampling techniques to generate samples repeatedly and randomly from a given dataset. One of these techniques, which came to be known as bootstrapping, was introduced by Efron [37] and has been widely used by practitioners to construct confidence intervals and/or approximate sampling distributions. In the context of this approach, the original dataset plays the role of a population from which samples of equal size are randomly drawn with replacement.

In generating weather series, the time dependence within each variable and interdependence among the weather variables should be preserved. However, since the resampled observations are selected independently, the serial dependence may not be preserved. This issue was resolved by using the block-resampling scheme introduced by Kunsch [38], which considers a block of observations as a single observation. In our case, a block that has a daily weather forecast (e.g., temperature, precipitation, relative humidity) is considered a single observation. Using the block-resampling scheme, the serial dependence can be preserved within the block, but not across. Assuming there are no dramatic climatic change effects, the dependence across the blocks can be preserved by considering each year's record as a single independent sample. The bootstrapping resampling technique can then be performed by:

- First, constructing an empirical probability distribution,F_n, from the observed sample by placing a probability of $1/n$ (where n= the number of years in the record) at each year.
- Second, drawing a random sample of size n with a replacement.

- Third, calculating a statistic of interest from the resampled set.
- Finally, repeating the second and the third steps until the required number of sets is achieved [39].

The construction of the non-parametric weather generator starts with the creation of a database of historical weather forecasts of the studied location. Weather parameters created in the database are those that have a direct effect on construction operations. However, all weather parameters can be added to the database for the purposes of covering most of the weather requirements in modelling construction operations. Once the database is created, a computer model is developed to generate random weather parameters. The random generation of weather parameters in this simplified approach begins with the random selection of the year from the database. When the user defines the day and month that represent the construction operation date, the generator begins sampling directly from the initialized date of operation. The weather generator reflects the generated weather in the form of a block containing all of the construction operation's weather parameters of interest. This block represents weather parameters that have been recorded and saved by weather stations, ensuring that correlations and dependencies amongst meteorological variables are well preserved. For this simplified approach to generating large-time and small-time-scale weather variables, daily and hourly historical weather forecast tables were created in the database. As shown in Figure 2- 4, after initializing the weather generation starting date, the generator provides the user with the flexibility of choosing between hourly and daily time intervals. If the construction operation of study requires daily weather forecasts, the generator samples directly from the daily forecast table in the database and advances the calendar one day before generating a second weather update. Where an hourly weather forecast is required, weather variables are sampled from the hourly forecast table, with the exception of certain weather variables, such as precipitation and snow depth, whose parameters are recorded in the daily weather forecast table. In this case, the weather update is performed after the time is advanced one hour and is moved to the second day in the calendar when 24-hour weather forecasts are generated.

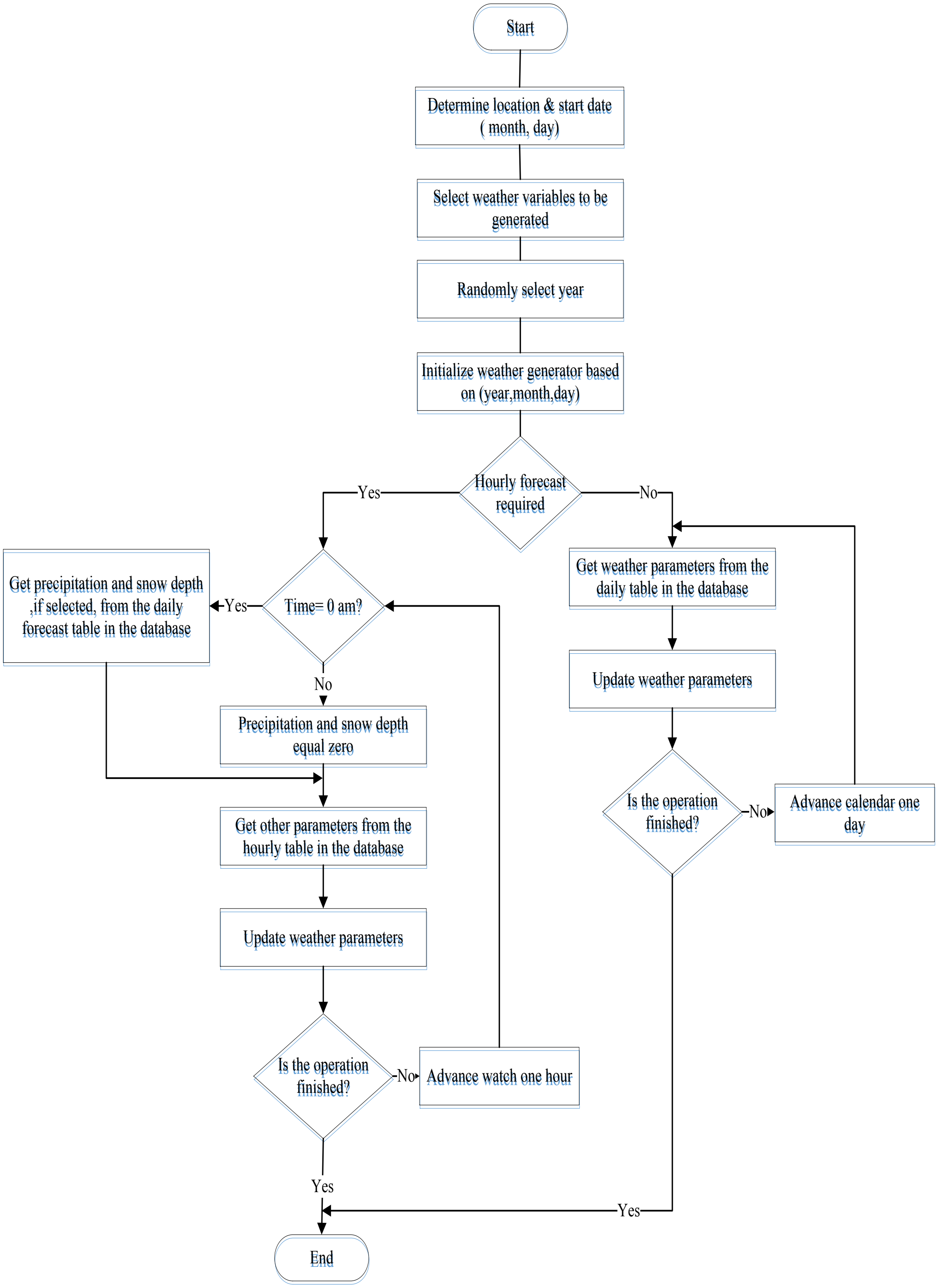

Start
Determine location & start date (month, day)
Select weather variables to be generated
Randomly select year
Initialize weather generator based on (year,month,day)
Hourly forecast required
Yes
No
Time= 0 am?
Yes
Get precipitation and snow depth ,if selected, from the daily forecast table in the database
No
Precipitation and snow depth equal zero
Get other parameters from the hourly table in the database
Update weather parameters
Is the operation finished?
No
Advance watch one hour
Yes
Get weather parameters from the daily table in the database
Update weather parameters
Is the operation finished?
No
Advance calendar one day
Yes
End

Figure 2- 4 Non-parametric weather generation flow chart

2.4 Weather Generators’ Evaluation

The evaluation of the parametric and non-parametric weather generators is performed in three stages as follows:

- Evaluation of weather generators’ assumptions.
- Evaluation of weather generators’ outputs.
- Evaluation of weather generators’ performance in weather-sensitive construction models.

2.4.1 Evaluation of weather generators’ assumptions

This stage is performed by applying the conceptual model validation approach [40], which determines if content, theories, and assumptions are correct and if the problem representation in the model logic, structure, and mathematical relationships is reasonable. In addition, a comparison with historical records is used to assess the reliability of the generated weather variables.

In the parametric approach, it was assumed that temperature and relative humidity were normally distributed, so that the serial correlation within these variables could be described using a first-order linear autoregressive model. This assumption is tested by calculating mean, standard deviation, skewness, and kurtosis and then comparing them to the normal distribution values. Assuming the residual series are normally distributed means that the skewness and kurtosis of the data should have values of 0 and 3 respectively. Moreover, the mean and standard deviation of the data should be values of 0 and 1 respectively in order to satisfy the standard normal distribution characteristics. Therefore, data from the historical records were grouped in a monthly basis and the required four statistical measures were calculated. This calculation was applied on four weather variables: maximum and minimum temperature, and maximum and minimum relative humidity. Wind speed and precipitation were not included in this test because they were generated independently by sampling from their distribution functions which were actually driven by the historical records. Table 2- 2 to Table 2- *5* show results of mean, standard deviation, skewness and kurtosis of maximum temperature, minimum temperature, maximum relative humidity, and minimum relative humidity. Results show that the mean and standard deviation for most of weather variables are close to the standard normal distribution values; however, the skewness and kurtosis of weather variables showed a deviation from normality. In addition, maximum relative humidity somehow showed a different distribution behaviour throughout the year. For example, it shows almost normal distribution behaviour from July to November, but is not normally distributed in the rest of the year. This result contradicts the assumption made on the distribution behaviour of weather data, which means that weather variables may have different distribution behaviour throughout the year.

Table 2- 2 Mean, standard deviation, skewness coefficient, and kurtosis coefficient of the residuals of maximum temperature (MAXTEMP)

Month	MAXTEMP			
	Mean	STD	Skewness	Kurtosis
Jan	0	1	0.32	-0.58
Feb	0.1189	1.0323	-0.04	-0.89
Mar	-0.0937	1.016	-0.17	-0.78
Apr	0.0001	0.9857	-0.32	0.32
May	0	1	0.03	-0.17
Jun	-0.0699	1.059	-0.27	-0.34
Jul	0	1	-0.25	0.07
Aug	0	1	0.02	-0.6
Sep	-0.0308	1.0009	0.13	-0.37
Oct	0	1	-0.25	0.06
Nov	0.0517	1.025	-0.12	-0.41
Dec	0	1	0	-0.52

Table 2- 3 Mean, standard deviation, skewness coefficient, and kurtosis coefficient of the residuals of minimum temperature (MINTEMP)

Month	MINTEMP			
	Mean	STD	Skewness	Kurtosis
Jan	0	1	0.06	-0.81
Feb	0.0987	1.019	-0.27	-0.61
Mar	0.0409	0.9294	-0.48	-0.42
Apr	-0.0183	1.0023	-0.8	0.83
May	0	1	-0.46	0.9
Jun	-0.1393	1.2777	-1.6	5.11
Jul	0	1	-0.19	0.05
Aug	0	1	-0.38	0.4
Sep	-0.0754	1.0898	-0.93	1.45
Oct	0	1	-0.64	0.37
Nov	0.0216	0.9999	-0.54	-0.06
Dec	0	1	-0.21	-0.7

Table 2- 4 Mean, standard deviation, skewness coefficient, and kurtosis coefficient of the residuals of maximum relative humidity (MAXRH)

Month	MAXRH			
	Mean	STD	Skewness	Kurtosis
Jan	-0.0004	0.9989	-0.75	-0.05
Feb	0.0432	1.0078	-0.76	0.22
Mar	0.0757	1.6941	-0.79	0.44
Apr	0.08	1.903	-0.71	0.01
May	0	1	-0.73	-0.18
Jun	-0.0105	1.0045	-1.09	0.58
Jul	0	1	-1.29	2.13
Aug	0	1	-1.37	2.7
Sep	-0.0335	1.0346	-1.58	2.95
Oct	0	1	-1.31	2.21
Nov	-0.0185	1.0259	-1.25	2.22
Dec	-0.0013	1.0008	-0.97	0.49

Table 2- 5 Mean, standard deviation, skewness coefficient, and kurtosis coefficient of the residuals of minimum relative humidity (MINRH)

Month	MINRH			
	Mean	STD	Skewness	Kurtosis
Jan	-0.0011	0.9991	-0.33	0.22
Feb	-2.6161	1.3498	-0.23	-0.22
Mar	0.0572	1.0545	0.16	-0.61
Apr	6.644	2.5539	0.86	0.11
May	0	1	1.33	1.85
Jun	0.0236	1.0013	0.91	0.22
Jul	0	1	0.83	0.35
Aug	0	1	0.84	0.3
Sep	0.0025	0.9921	0.6	-0.31
Oct	0	1	0.25	-0.87
Nov	-0.0441	1.0288	-0.65	0.16
Dec	-0.0006	0.9998	-0.39	-0.24

Two normality tests were applied: Anderson-Darling, and Kolmogorov-Smirnov tests. These are two of the Empirical Distribution Function (EDF) tests that are based on the measure of discrepancy between the empirical and the hypothesized distributions [41]. Both tests start by defining the null hypothesis (H_o), which presumes that the data are normally distributed, and the alternative hypothesis, (H_1) which presumes they are not. The acceptance and rejection of the null hypothesis is made using the corresponding p-value; if the p-value is less than α (significance level) then the null hypothesis is rejected and vice versa. Table 2- 6 to Table 2- *9* show results of p-values of maximum temperature, minimum temperature, maximum relative humidity, and minimum relative humidity (a significance level of 5% was used in both tests). Results show that both tests rejected the null hypothesis, which means that the residuals of weather variables are not normally distributed. In addition, normal probability plots of the historical records were plotted to assess the normality assumption. These plots are shown in Figure 2- 5 to Figure 2- *8*. The weather variables exerted some deviation from normality, especially with the maximum relative humidity, which largely deviated from normality.

Table 2- 6 P-values of Anderson-Darling and Kolmogorov-Smirnov tests of the residuals of maximum temperature (MAXTEMP)

Month	MAXTEMP			
	Anderson-Darling	**Normality**	**Kolmogorov-Smirnov**	**Normality**
Jan	4.97E-14	✕	0.0003871	✕
Feb	3.73E-15	✕	1.47E-11	✕
Mar	6.77E-14	✕	9.03E-06	✕
Apr	2.20E-16	✕	1.83E-07	✕
May	0.000112	✕	0.003748	✕
Jun	2.20E-16	✕	1.99E-08	✕
Jul	2.76E-12	✕	0.0001548	✕
Aug	6.36E-08	✕	0.02612	✕
Sep	8.53E-11	✕	6.89E-08	✕
Oct	4.13E-14	✕	3.15E-08	✕
Nov	0.0003603	✕	0.003747	✕
Dec	9.83E-06	✕	0.04865	✕

Table 2- 7 P-values of Anderson-Darling and Kolmogorov-Smirnov tests of the residuals of minimum temperature (MINTEMP)

Month	MINTEMP			
	Anderson-Darling	**Normality**	**Kolmogorov-Smirnov**	**Normality**
Jan	2.20E-16	✕	1.44E-06	✕
Feb	2.20E-16	✕	8.55E-14	✕
Mar	2.20E-16	✕	3.01E-14	✕
Apr	2.20E-16	✕	5.78E-11	✕
May	1.68E-10	✕	0.00196	✕
Jun	2.20E-16	✕	0.006914	✕
Jul	0.0003931	✕	0.002608	✕
Aug	3.87E-05	✕	0.03094	✕
Sep	2.20E-16	✕	0.06237	✓
Oct	2.20E-16	✕	9.69E-13	✕
Nov	2.20E-16	✕	6.66E-16	✕
Dec	2.20E-16	✕	0.0001827	✕

Table 2- 8 P-values of Anderson-Darling and Kolmogorov-Smirnov tests of the residuals of maximum relative humidity (MAXRH)

Month	MAXRH			
	Anderson-Darling	Normality	Kolmogorov-Smirnov	Normality
Jan	2.20E-16	✕	2.11E-15	✕
Feb	2.20E-16	✕	2.20E-16	✕
Mar	2.20E-16	✕	2.20E-16	✕
Apr	2.20E-16	✕	2.20E-16	✕
May	2.20E-16	✕	6.18E-12	✕
Jun	2.20E-16	✕	2.20E-16	✕
Jul	2.20E-16	✕	2.20E-16	✕
Aug	2.20E-16	✕	2.20E-16	✕
Sep	2.20E-16	✕	2.20E-16	✕
Oct	2.20E-16	✕	2.20E-16	✕
Nov	2.20E-16	✕	1.44E-15	✕
Dec	2.20E-16	✕	2.20E-16	✕

Table 2- 9 P-values of Anderson-Darling and Kolmogorov-Smirnov tests of the residuals of minimum relative humidity (MINRH)

Month	MINRH			
	Anderson-Darling	Normality	Kolmogorov-Smirnov	Normality
Jan	7.16E-05	✕	0.001314	✕
Feb	2.88E-06	✕	2.20E-16	✕
Mar	3.19E-09	✕	1.14E-05	✕
Apr	2.20E-16	✕	2.20E-16	✕
May	2.20E-16	✕	2.20E-16	✕
Jun	2.20E-16	✕	5.55E-11	✕
Jul	2.20E-16	✕	2.05E-11	✕
Aug	2.20E-16	✕	8.87E-14	✕
Sep	2.20E-16	✕	1.77E-08	✕
Oct	2.20E-16	✕	1.20E-06	✕
Nov	2.20E-16	✕	0.00021	✕
Dec	9.76E-15	✕	9.46E-06	✕

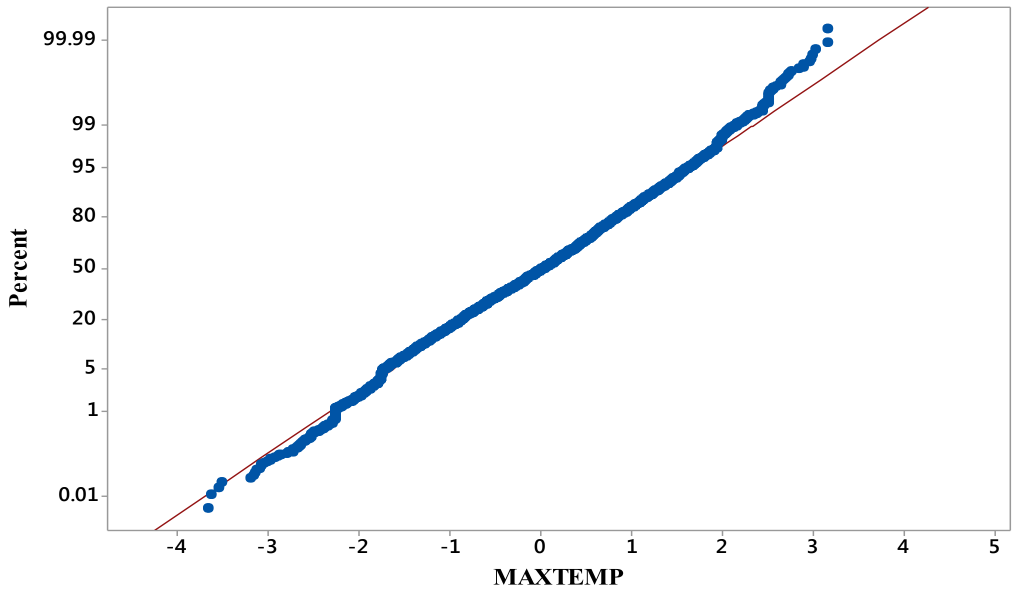


Figure 2- 5 Normal probability plot of the residuals of maximum temperature (MAXTEMP)

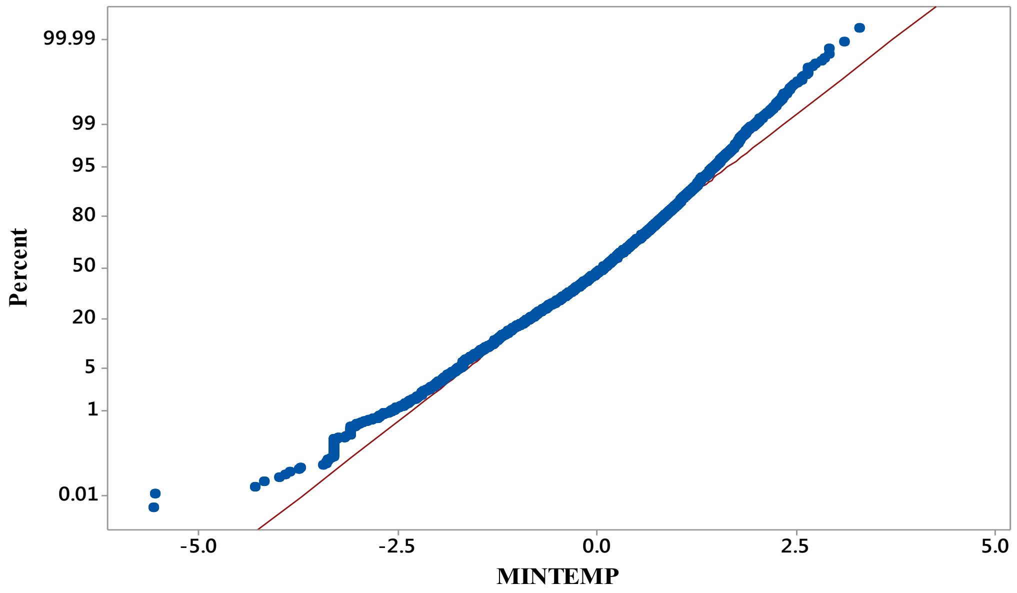


Figure 2- 6 Normal probability plot of the residuals of minimum temperature (MINTEMP)

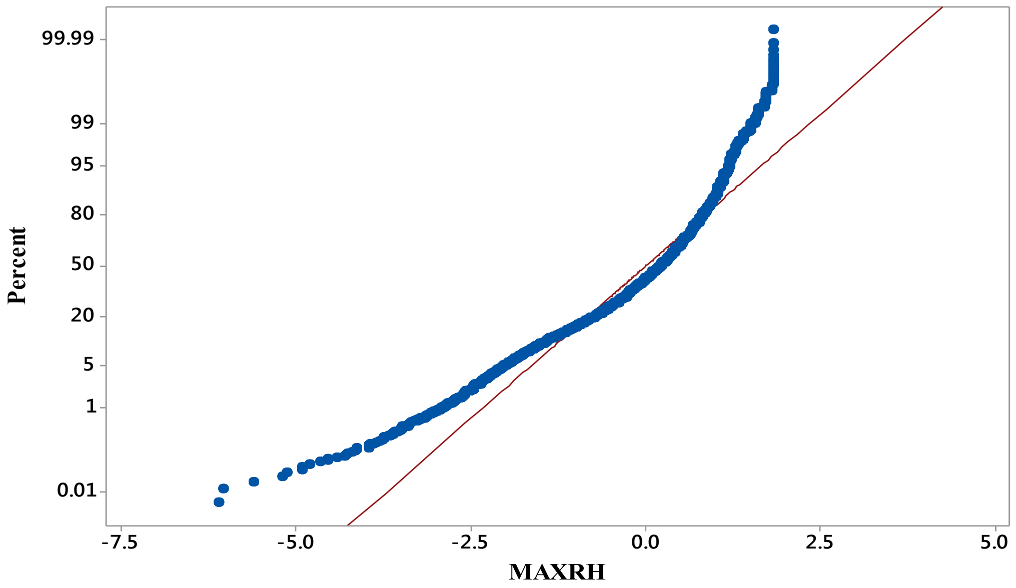


Figure 2- 7 Normal probability plot of the residuals of maximum relative humidity (MAXRH)

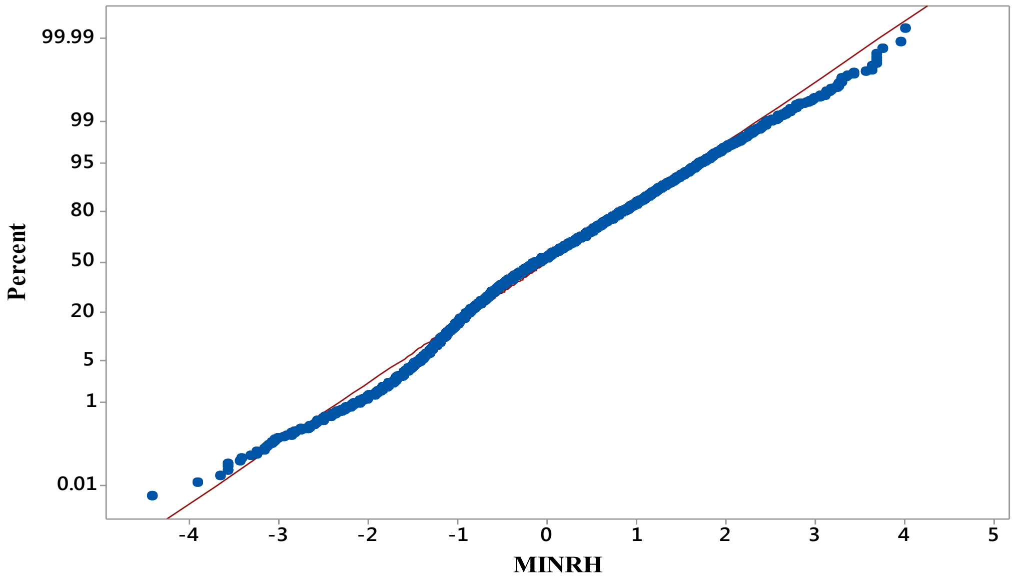


Figure 2- 8 Normal probability plot of the residuals of minimum relative humidity (MINRH)

In addition to the normality assumption, another assumption was that the first-order autoregressive model in the parametric approach and the block-resampling scheme in the non-parametric approach can approximate the serial dependence of weather variables. Therefore, serial correlation coefficients of lags up to five days for each residual series were calculated and then compared with serial correlation coefficients of residual series of the historical records. Figure 2- 9 to Figure 2- *12* illustrate the analysis results. The parametric weather generator provided a better serial correlation coefficients' approximation for the maximum and minimum temperature than did the non-parametric weather generator. Meanwhile, the non-parametric weather generator performed better in approximating the serial correlation coefficients of the maximum and minimum relative humidity. In general, both generators provided acceptable approximations of serial correlation coefficients.

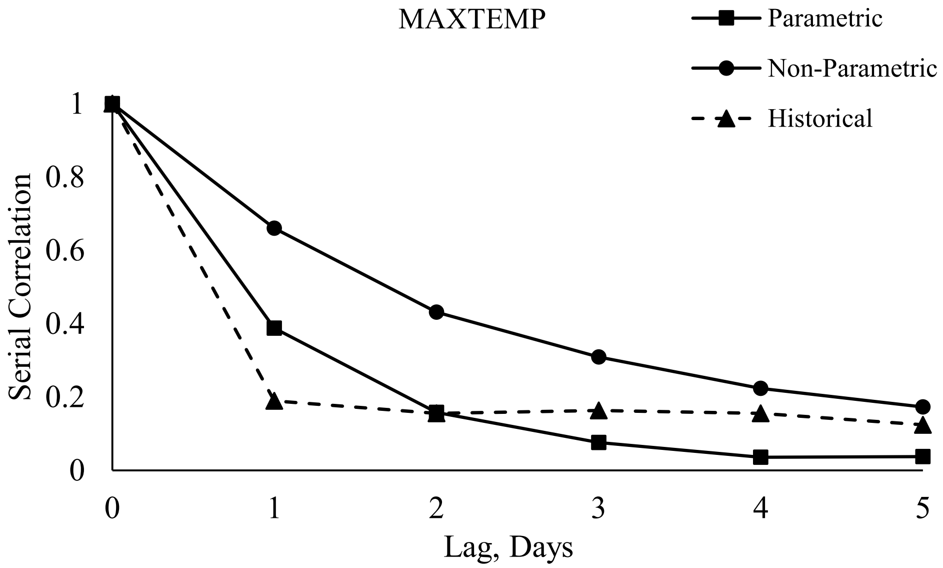


Figure 2- 9 Serial correlation coefficients of maximum temperature

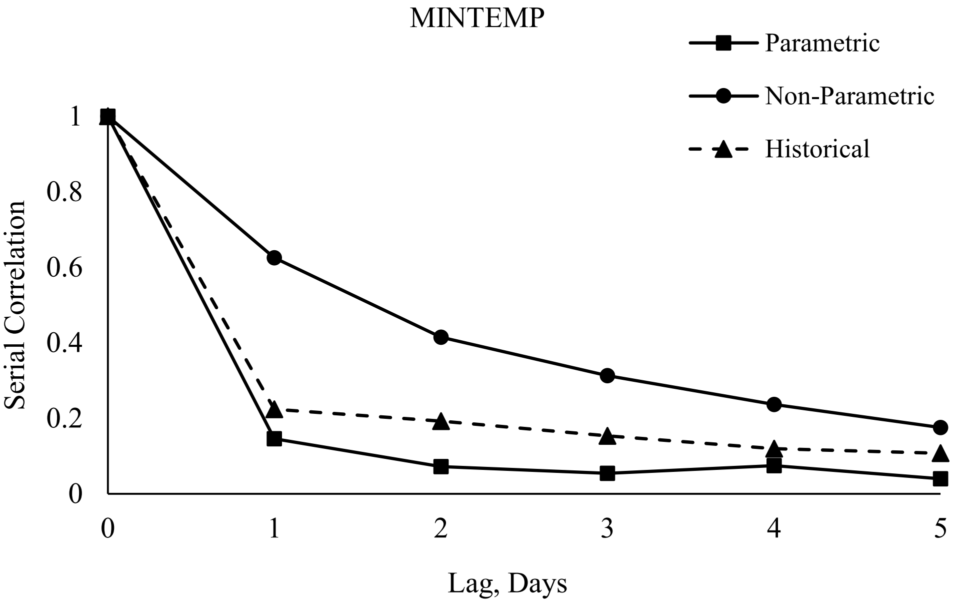


Figure 2- 10 Serial correlation coefficients of minimum temperature

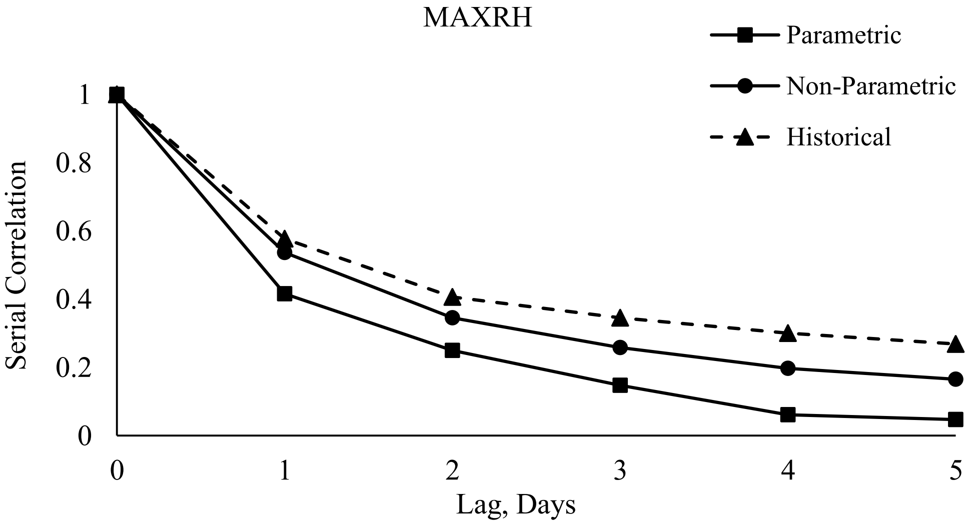


Figure 2- 11 Serial correlation coefficients of maximum relative humidity

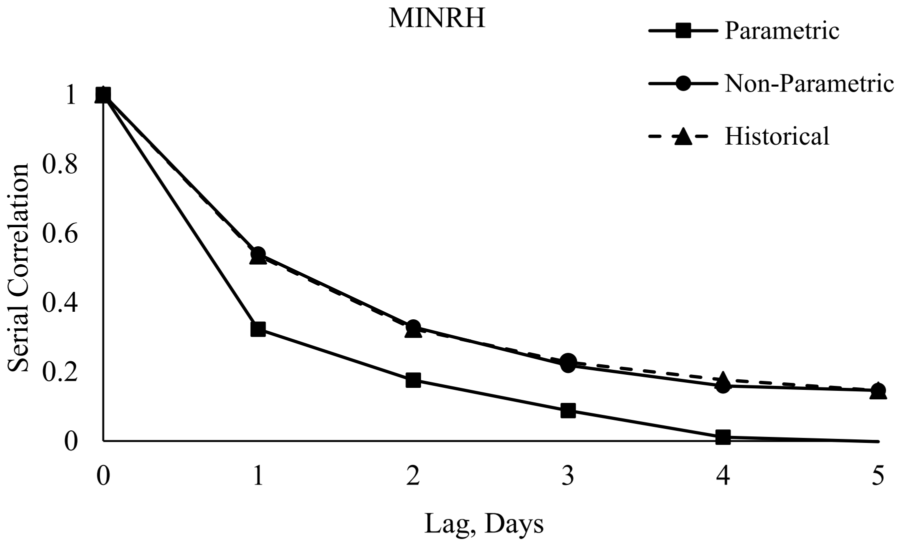


Figure 2- 12 Serial correlation coefficients of minimum relative humidity

Wind speed in the parametric approach was assumed as an independent weather variable which has no correlation with other variables; it was modeled by fitting its daily average values to a two-state gamma distribution. However, a relationship between wind speed, temperature, and relative humidity may exist. Therefore, a correlation test based on Pearson's product moment correlation coefficient was applied between wind speed, temperature and relative humidity. The test was applied on weather series generated from both generators and on the historical records as well. Correlation coefficients were calculated for each generated year and 10-year values were plotted as shown in Figure 2- 13 to Figure 2- 16. The historical averages of correlation coefficients indicate the existence of a positive relationship between wind speed and temperature, and a negative relationship between wind speed and relative humidity. The non-parametric weather generator provided the same relationship with higher correlation coefficients. Likewise, the parametric weather generator, although the wind speed was modeled independently, also maintained the same relationships.

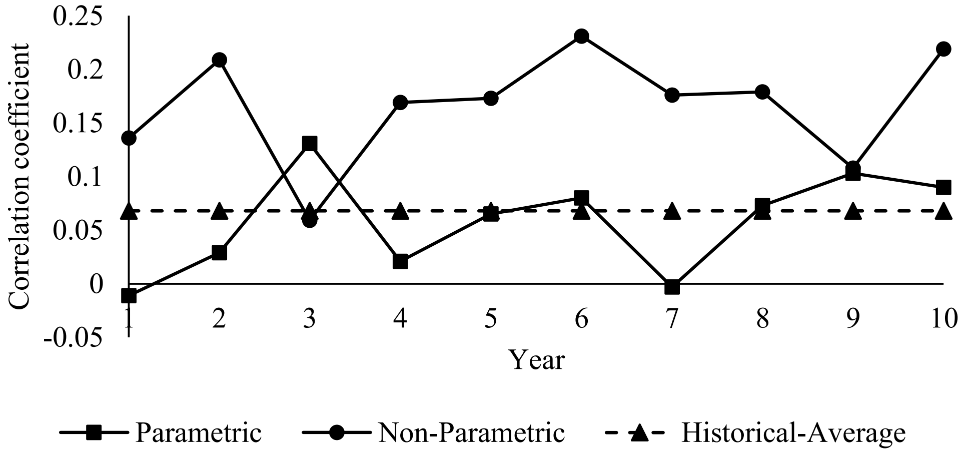


Figure 2- 13 10 years' distribution of correlation coefficients between maximum temperature and wind speed

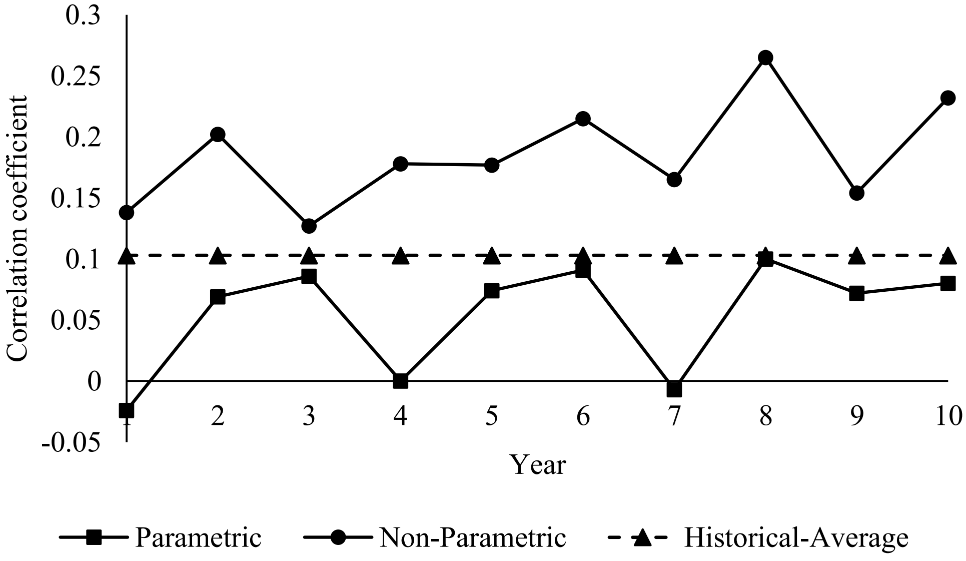


Figure 2- 14 10 years' distribution of correlation coefficients between minimum temperature and wind speed

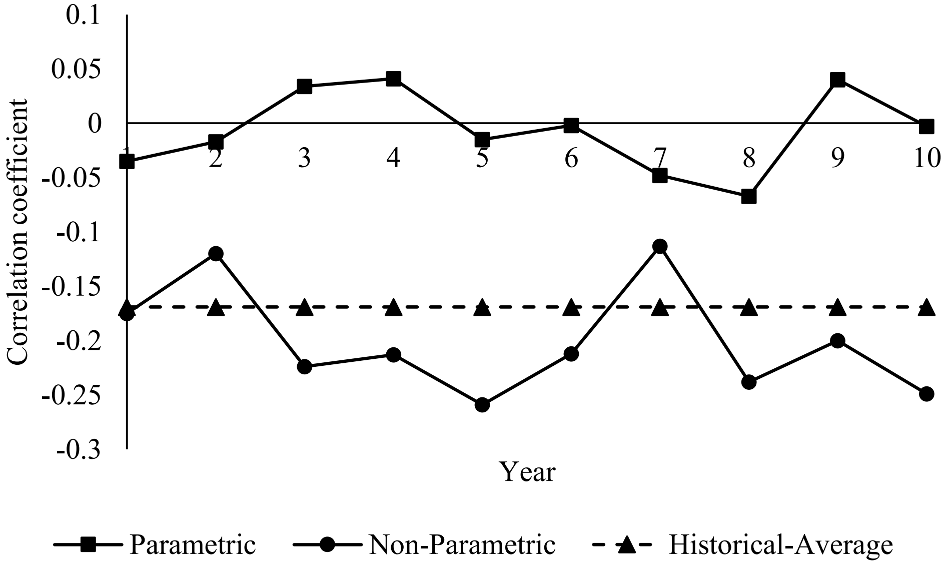


Figure 2- 15 10 years' distribution of correlation coefficients between maximum relative humidity and wind speed

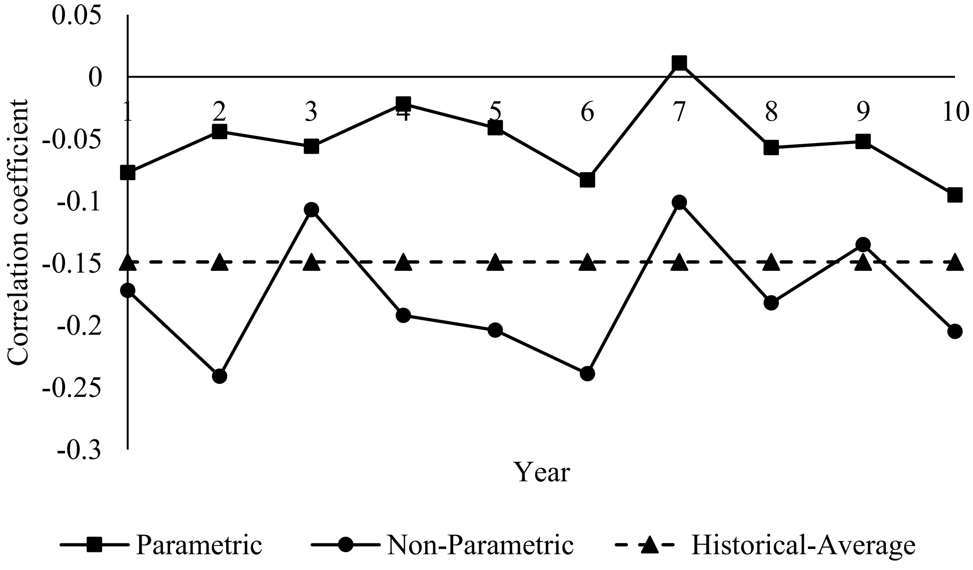


Figure 2- 16 10 years' distribution of correlation coefficients between minimum relative humidity and wind speed

2.4.2 Evaluation of weather generators' outputs

The second stage in the evaluation process is concerned with testing the reliability of weather generators' outputs in terms of monthly means and standard deviations. Figure 2- 17 to Figure 2- 22 illustrate the comparison between the monthly averages of maximum temperature, minimum temperature, maximum relative humidity, minimum relative humidity, precipitation, and wind speed generated from both the parametric and non-parametric weather generator against the historical averages. In the context of maximum and minimum temperatures, both the parametric and the non-parametric weather generators provided almost similar averages compared to the historical records. The parametric weather generator provided more accurate averages than the non-parametric weather generator (see Appendix B for more details on the output results). This result was expected because in the parametric weather generation mechanism the residuals of weather variables are generated and added to historical monthly averages. However, while the parametric model is expected to perform better than its non-parametric counterpart, the differences between the synthetic time series, with respect to historical records, is approximately a fraction of a degree Celsius apart, which, from a practical view, is acceptable. The same result was also observed in the averages of relative humidity.

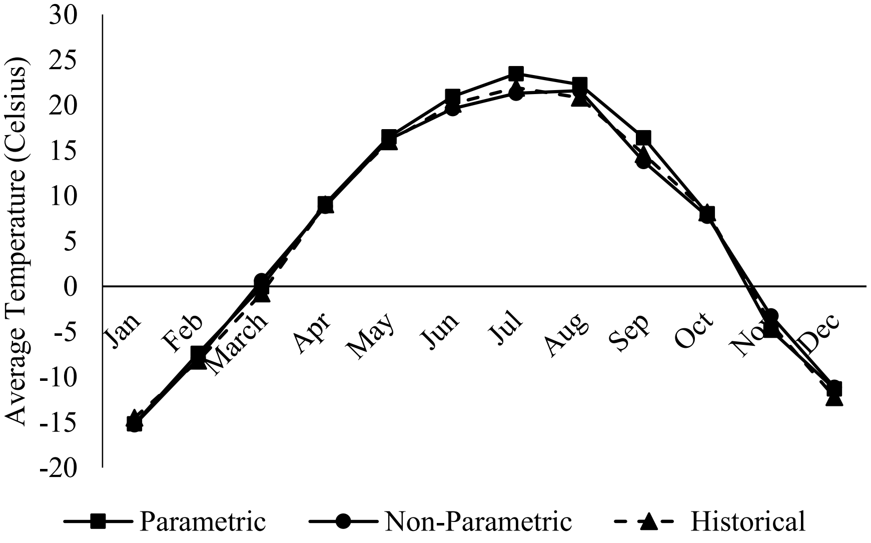


Figure 2- 17 Monthly averages of maximum temperature

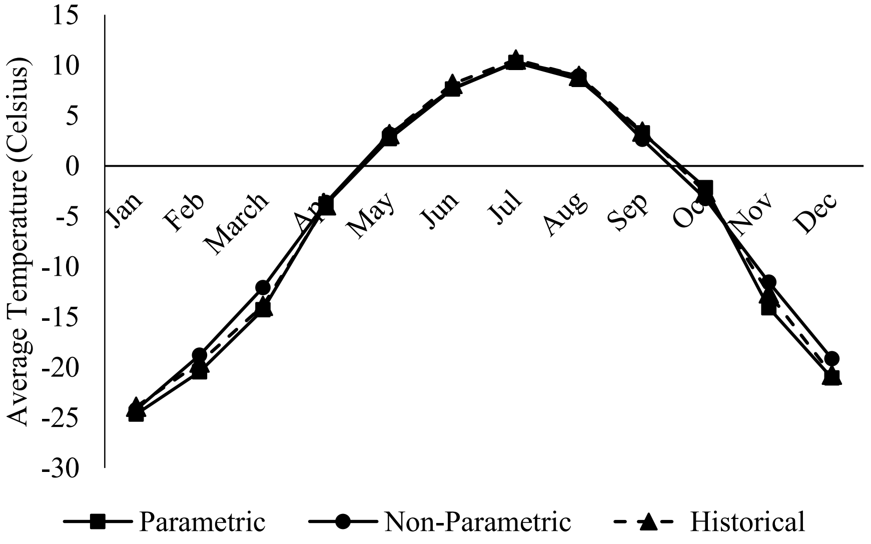


Figure 2- 18 Monthly averages of minimum temperature

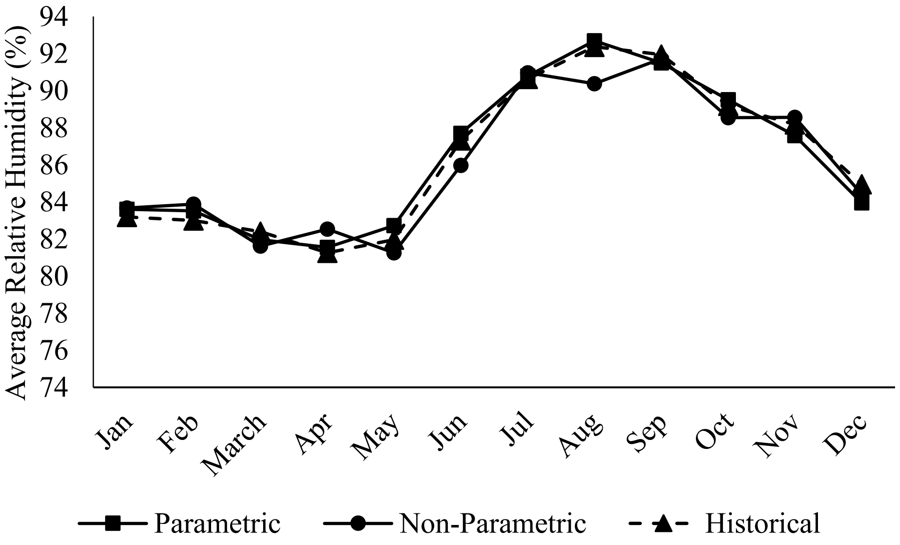


Figure 2- 19 Monthly averages of maximum relative humidity

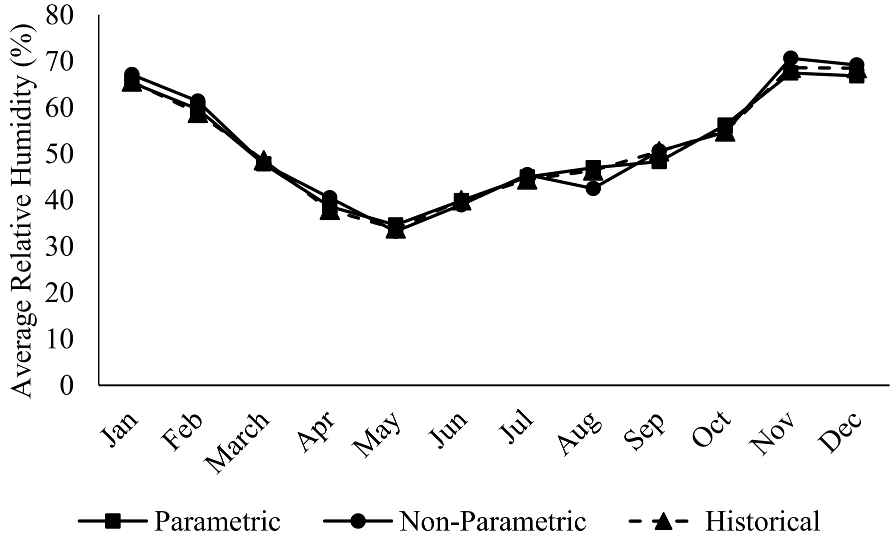


Figure 2- 20 Monthly averages of minimum relative humidity

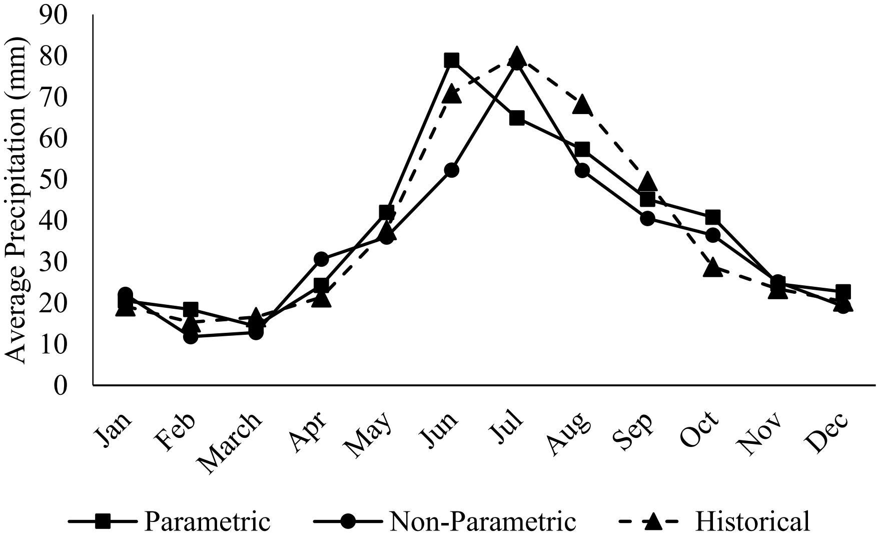


Figure 2- 21 Monthly averages of precipitation

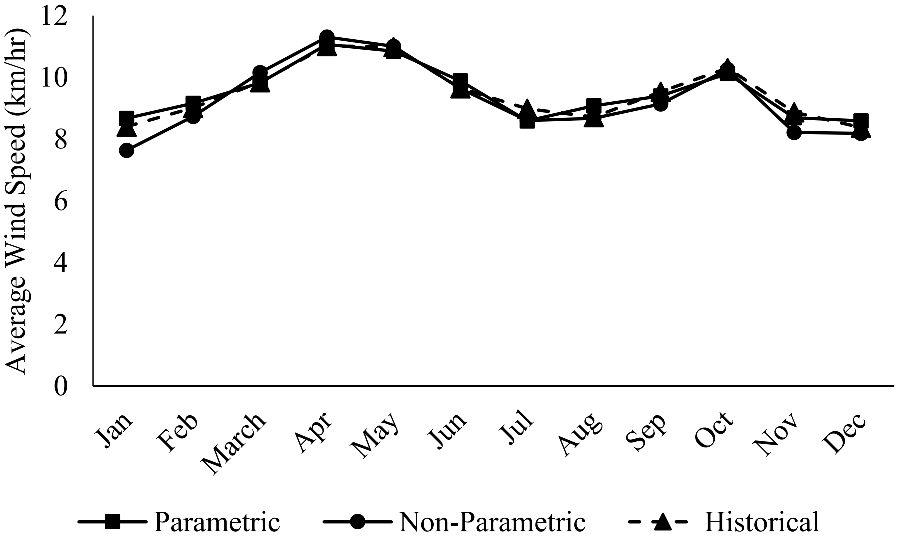


Figure 2- 22 Monthly averages of wind speed

Regarding the amount of generated precipitation (see Figure 2- 21), the non-parametric generator led to the same maximum average precipitation at the same period of time while the parametric generator produced a skewed maximum precipitation peak. Such discrepancy maybe related to the shape parameters (α) and the inverse scale parameter (β) used in the two-state gamma distribution function. In addition to comparing weather generators based on the amount of monthly precipitation, a comparison based on the length of wet spells (shown in Table 2- 10) was conducted on outputs from both generators. The comparison is based on three measures: (1) the average number of wet days, (2) the average number of wet intervals that lasted for more than two days, and (3) the average duration of wet intervals.

Table 2- 10 Analysis of length of wet spells

Month	**Number of wet days**			**Number of wet intervals**			**Average duration of wet intervals**		
	P*	NP*	H*	P*	NP*	H*	P*	NP*	H*
Jan	12.4	13.9	12.6	2.9	3.4	3.0	2.7	3.3	3.4
Feb	10.4	8.9	10.4	3	2.1	2.6	2.7	3.3	3.2
March	7.7	8.3	9.8	1.1	2	2.6	2.6	2.9	3.0
Apr	7.6	10.3	7.8	1.6	2.9	2.0	2.4	2.7	2.6
May	11.2	10.7	10.4	3	2.9	2.6	2.2	2.7	2.8
Jun	13.6	12.2	13.5	3.9	3.8	3.6	2.6	2.6	3.1
Jul	14.6	15.5	15.4	3.6	4.6	3.9	2.8	2.9	3.1
Aug	11.6	11.2	12.95	3.1	2.2	3.2	2.4	3.5	3.3
Sep	12.1	13.1	12.3	3.1	3.4	3.2	2.6	3.1	3.3
Oct	11.9	11.9	10.8	2.4	3	2.6	2.6	3.2	3.1
Nov	12.2	12.5	12.4	2.9	3	3.05	2.5	3.3	3.3
Dec	13.3	13	12.6	3.4	3.5	3.3	3.0	3.1	3.1

**(P: Parametric, NP: Non-Parametric, H: Historical)*

Table 2- 10 demonstrates that both weather generators produced a very similar monthly average number of wet days. While the parametric generator outperforms the non-parametric, in the case of the average number of wet intervals, the non-parametric weather generator provided more accurate durations of wet intervals than did the parametric counterpart. Finally, in the case of wind speed, both weather generators (see Figure 2- 22) generated almost identical wind speed averages when compared to the historical records.

In contrast to the output averages, when comparing the standard deviation of each weather variable from both generators, the non-parametric weather generator had a better performance (see Figure 2- 23 to Figure 2- 28). This is clearly noticeable in Figure 2- 25. In this figure, the parametric weather generator showed large differences when compared to the standard deviation of the historical records of maximum relative humidity. Meanwhile, the behaviour of the non-parametric weather generator came close to matching the historical records. For that reason, additional tests such as t and F tests were conducted for each month to obtain better insight into the behaviour of the mean and variance of the generated weather series compared to the historical record. These tests were conducted to determine whether the generated weather series from both generators was significantly different from those in the historical record. The two tests were applied on four weather variables: maximum and minimum temperature, and maximum and minimum relative humidity. Table 2- 11 shows the test results.

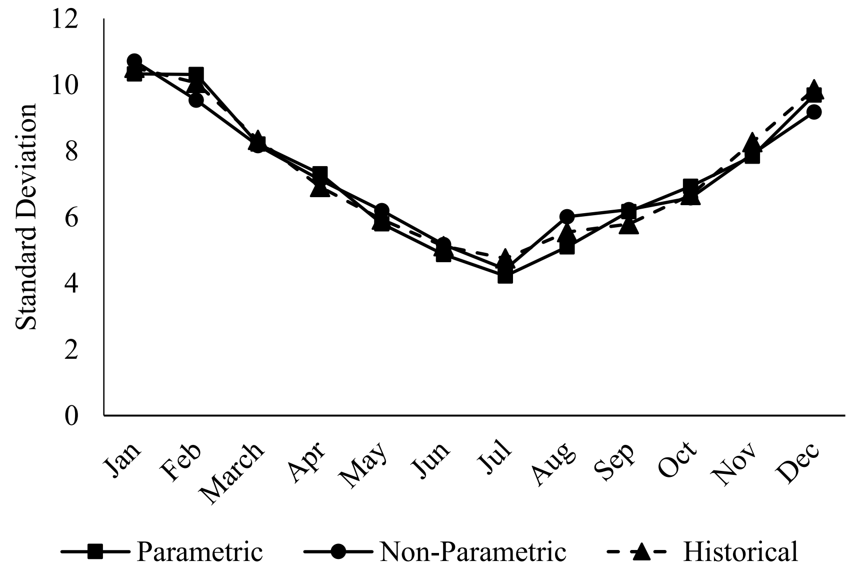


Figure 2- 23 Standard deviation of maximum temperature

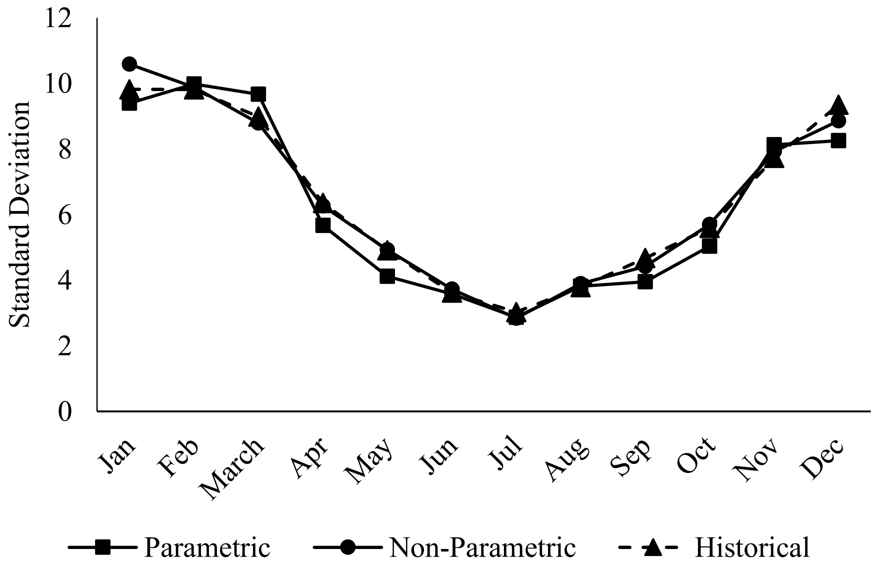


Figure 2- 24 Standard deviation of minimum temperature

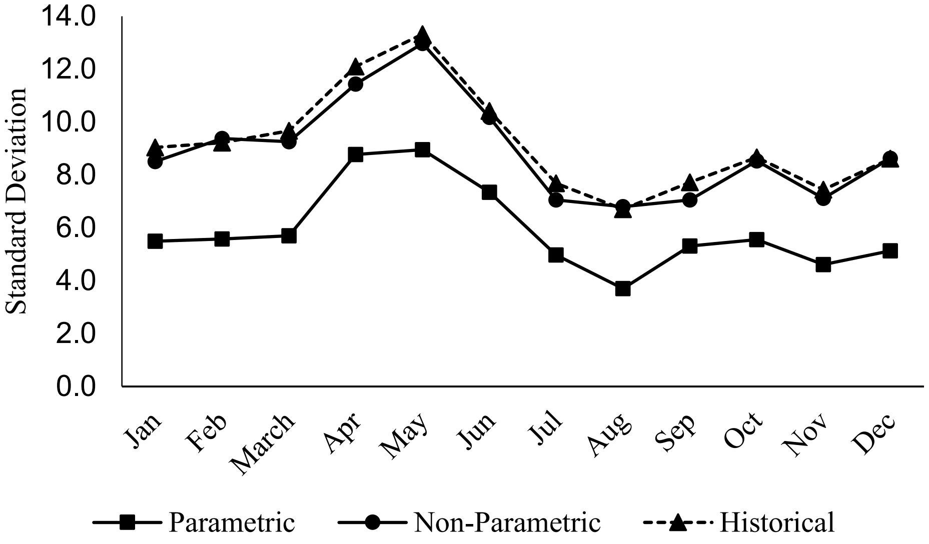


Figure 2- 25 Standard deviation of maximum relative humidity

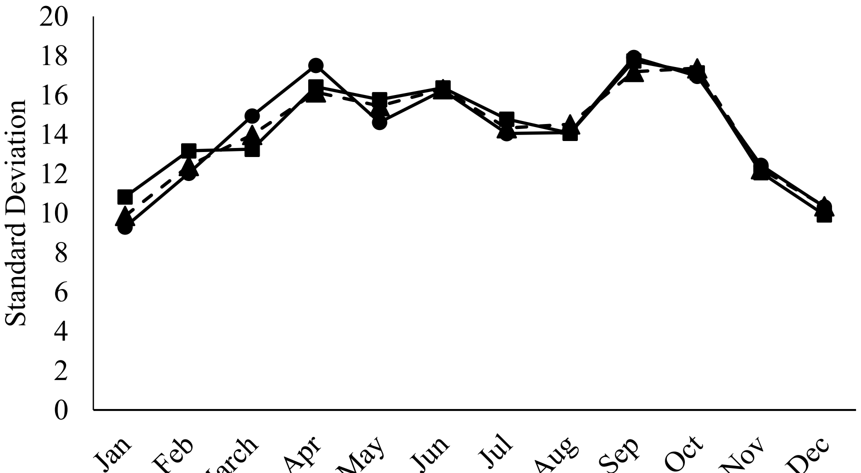


Figure 2- 26 Standard deviation of minimum relative humidity

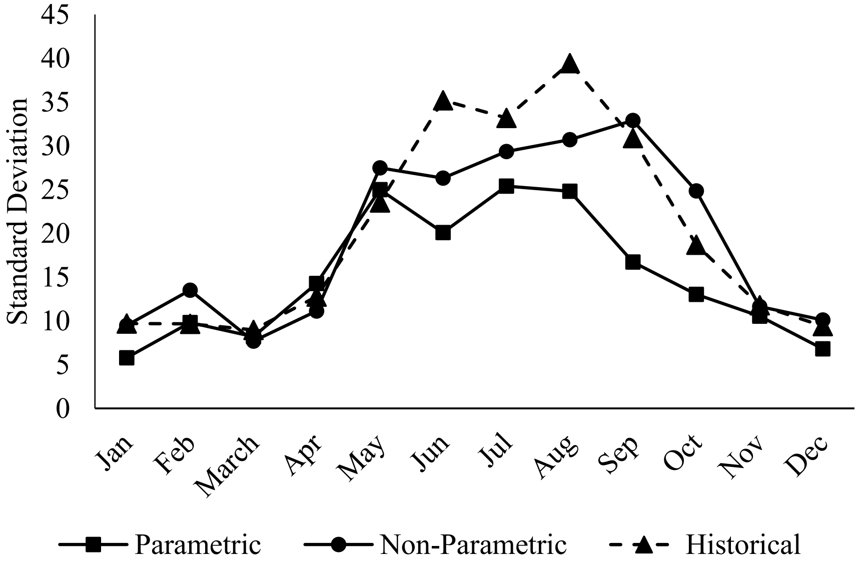


Figure 2- 27 Standard deviation of precipitation

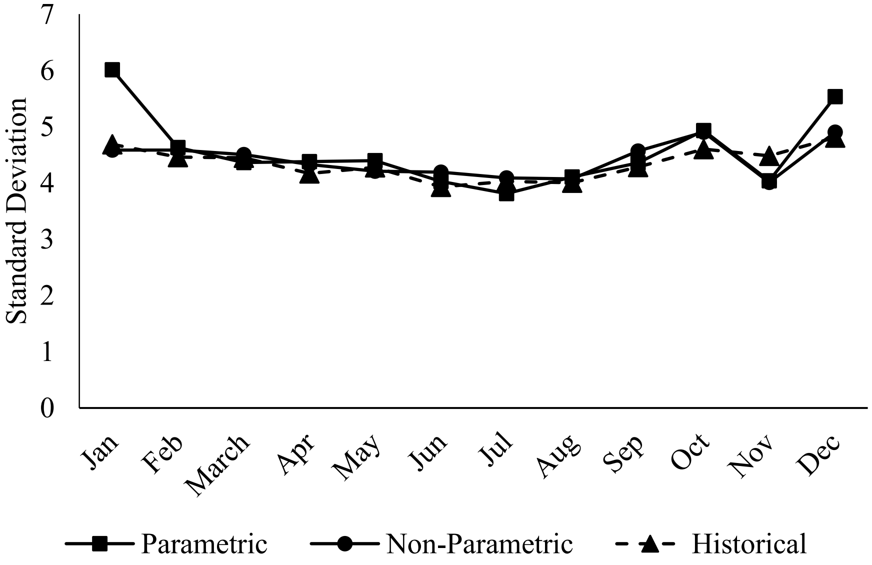


Figure 2- 28 Standard deviation of wind speed

Table 2- 11 Number of months rejected by t and F tests for four weather variables

Test	MAXTEMP		MINTEMP		MAXRH		MINRH	
	P*	NP*	P*	NP*	P*	NP*	P*	NP*
t-test (rejection)	4	4	4	5	2	2	2	4
F-test (rejection)	0	0	5	2	12	0	1	0

**(P: Parametric, NP: Non-Parametric)*

Table 2- 1 shows that the non-parametric weather generator outperforms the parametric weather generator, which means that the non-parametric weather generator provides a data spread similar to those in the historical records. The parametric weather generator showed poor results in maximum relative humidity; the data spread generated is significantly different. Failure in generating a similar data spread can be interpreted as the parametric weather generator failing to provide wider possible relative humidity scenarios that may exist in reality.

In addition to average and standard deviation-based analysis, a cross correlation test was applied to measure the similarity between the generated time series from both generators with the time series from the historical records. This test was performed by first randomly selecting a baseline year for the comparison from the historical records. Two independent years were generated from both generators and, finally, a cross correlation test was applied for each weather variable in the time series. The test was performed 10 times to show the cross correlation coefficients' behaviour of the parametric and non-parametric weather generators against the historical year (see Figures 2-29 to 2-34). Both generators have almost the same behaviour when compared to the historical baseline year, except in some runs. For instance, in

MAXTEMP, the parametric and non-parametric weather generators provided different cross correlation coefficients through the first three runs and were almost the same in the rest of the runs. Likewise, MAXRH showed that for the non-parametric weather generator, no correlation exists with the historical year in the eighth and ninth runs, but the parametric weather generator provided better cross correlation coefficients for the same runs. In general, both generators have performed well in the cross correlation test with the preference given to the parametric weather generator.

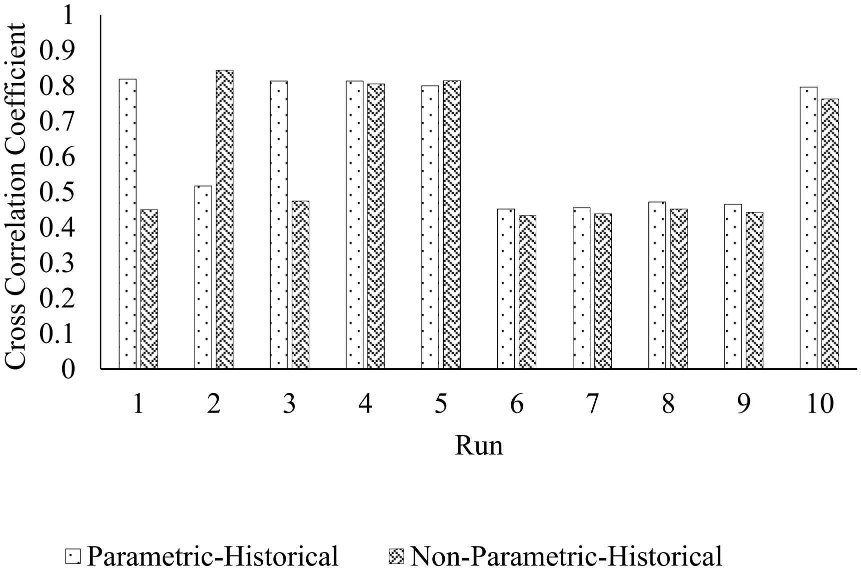


Figure 2- 29 Distribution of 10 runs of cross correlation coefficient of maximum temperature

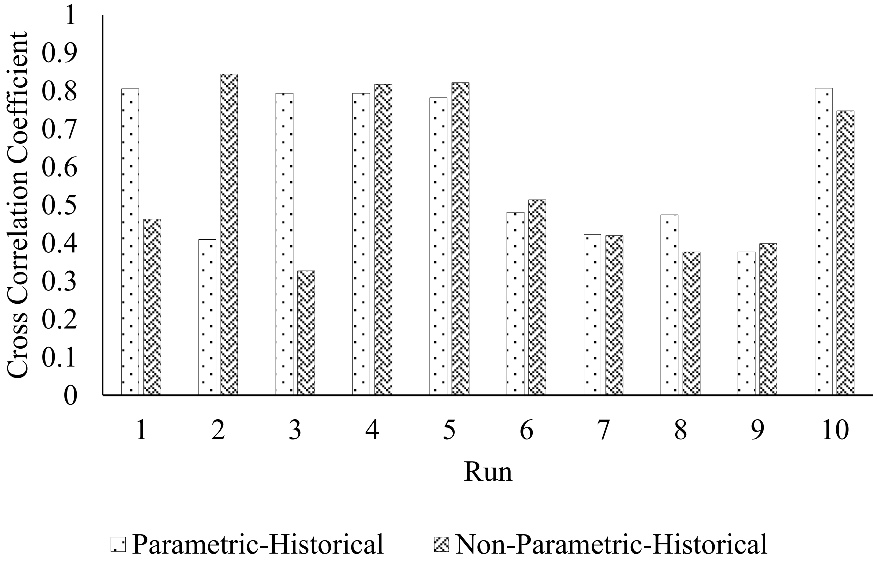


Figure 2- 30 Distribution of 10 runs of cross correlation coefficient of minimum temperature

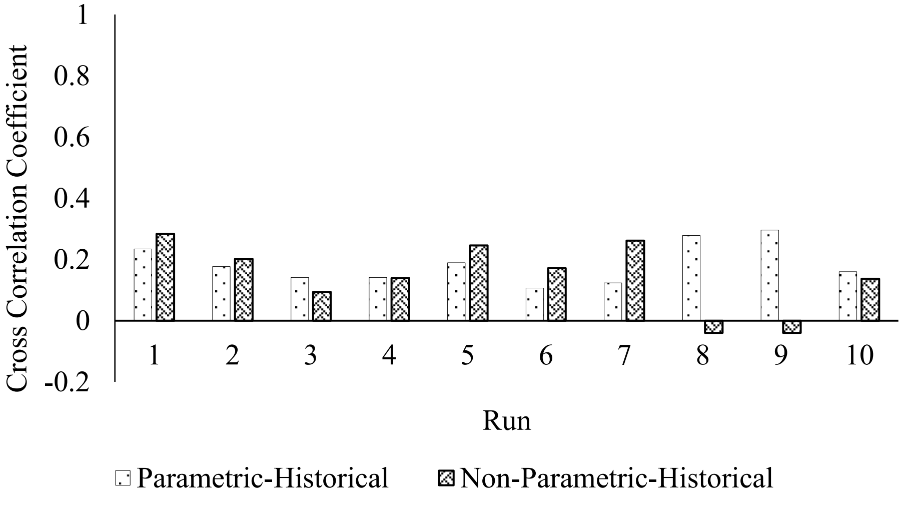


Figure 2- 31 Distribution of 10 runs of cross correlation coefficient of maximum relative humidity

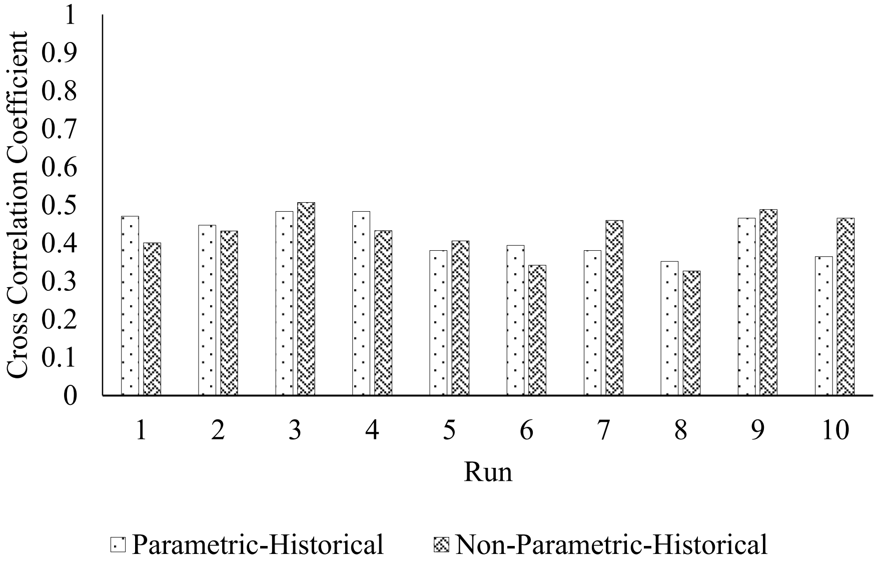


Figure 2- 32 Distribution of 10 runs of cross correlation coefficient of minimum relative humidity

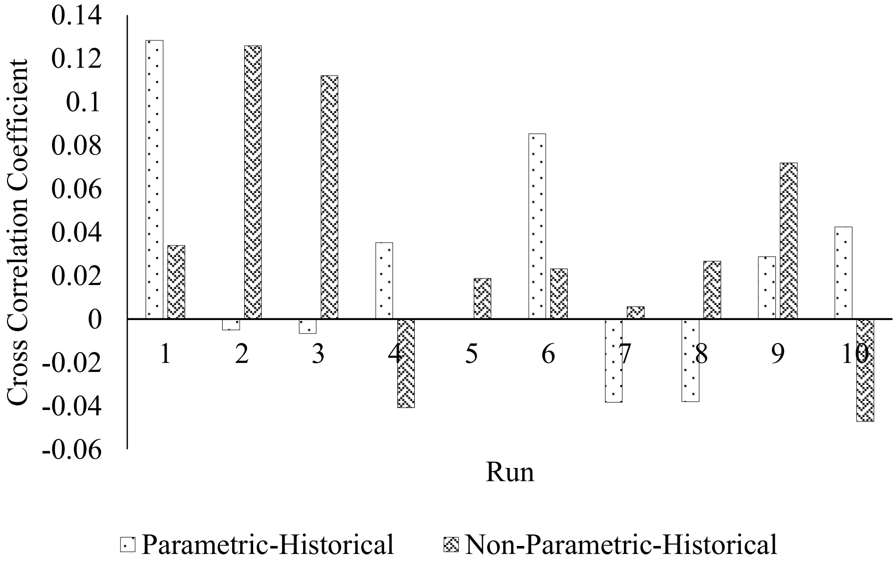


Figure 2- 33 Distribution of 10 runs of cross correlation coefficient of precipitation

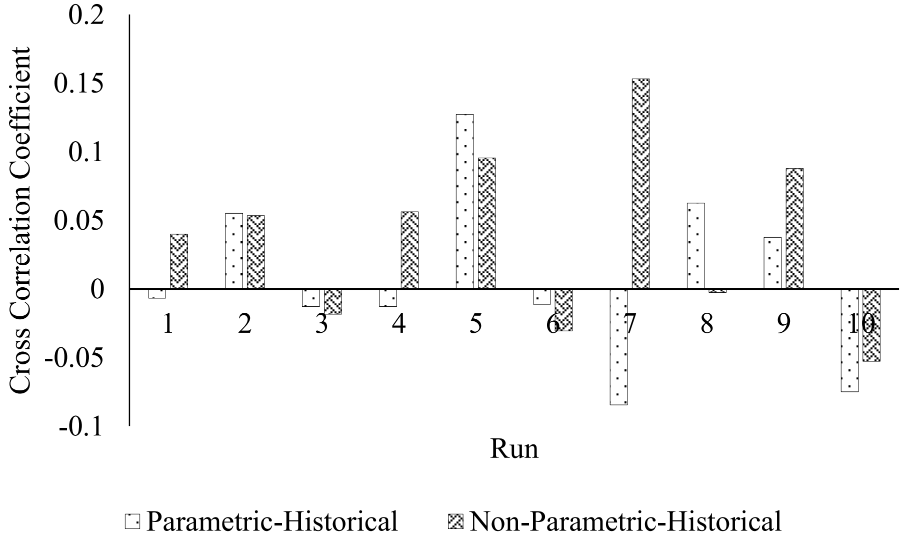


Figure 2- 34 Distribution of 10 runs of cross correlation coefficient of wind speed

At the end of this section and despite the inadequacies associated with the parametric weather generator such as the assumption of distribution and the missing correlation between wind speed and other weather variables, the above monthly based weather analysis showed that both weather generators performed well in generating synthetic weather variables.

2.4.3 Evaluation of weather generators' performance in weather-sensitive construction models

This stage of evaluation is performed to investigate the way in which imperfections associated with the generated weather series from both weather generators affect weather-sensitive construction models. In Figure 2- 13 to Figure 2- 16, wind speed is correlated to temperature. Both generators maintained the relationship to different degrees, although the non-parametric weather generator provided higher correlation coefficients than the parametric model. While the differences in correlation coefficients caused no effect when the monthly averages of generated weather series were compared to the historical records, it is expected that differences would emerge when a certain simulation output were co-dependent on temperature and wind speed. This was investigated by developing a construction model that uses temperature and wind speed as input variables to predict construction performance. Results from Table 2- 11 showed that there is a difference in the minimum temperature data spreads of both generators when compared to historical records. Therefore, another weather sensitive construction model that uses minimum temperature as an input variable was developed to assess the performance of both weather generators in modelling construction operations.

2.4.3.1 Estimating temperature-wind speed effects in construction labor

In the construction industry, projects are executed in an open work environment, which can directly affect productivity and efficiency. The performance of construction operations changes depending on the working season because changes in weather conditions affect construction manpower through work-related disorders that occur as a result of extreme weather conditions. In hot regions, high temperature, high relative humidity, heat, and ultraviolet radiation are examples of weather variables that affect construction labourers. These variables produce injuries such as heat stroke, sunburn, and heat exhaustion and their associated risk level varies from dehydration to fatality. In cold regions, injuries are often due to a combination of low temperature and wind speed. Frostbite is the most common injury in this working environment. Its injurious effect is enhanced on wet skin, as wet skin has a higher effective temperature for freezing than dry skin, as described by Kohen and Brown [17] in Table 2-12. In cold weather regions, the combination of weather variables such as temperature and wind speed can be used to quantify the effect of weather changes on labour productivity. Occupational health and safety regulation guidelines are a reliable source that can be used in modelling weather-sensitive construction models.

Table 2- 12 Minimum wind speed[a] required for freezing exposed skin *[17]*

Temperature (C^o)	Wet skin (miles/hour)	Dry skin (miles/hour)
-1	15	-
-7	7	30
-12	4	15
-18	3	10
-23	2	7
-29	1	5
-34	-	3

The Canadian Center for Occupational Health and Safety (CCOHS) has a cold exposure guideline for workers. These guidelines include a work warm-up schedule for outdoor activities, as shown in Table 2- 13. The schedule is adopted from the American Conference of Governmental Industrial Hygienists (ACGIH) [18] and recommends maximum work periods and break numbers for four-hour shifts, conditional on the temperature and wind speed values. CCOHS also identified the effect of the combination of relative humidity and temperature on construction working periods. CCOHS provided a recommendation for the number of breaks to be taken in accordance to the humidex reading.

Table 2- 13 is transformed to Table 2- 14 to simplify the quantification of weather effects on labourers' productivity. Table 2- 14 illustrates minutes lost per four-hour shifts due to temperature and wind speed and is used as a black box model. The model will receive inputs of weather series generated from the parametric and non-parametric weather generators and will provide an output in the form of expected minutes lost in a four-hour shift cycle.

The parametric weather generator generates a daily weather series, whereas its non-parametric counterpart has the flexibility of generating daily and hourly weather series. To maintain test consistency between the two weather generators, only daily weather series were considered. However, two different input testing scenarios were conducted using different combinations of temperature and wind speed:

Input scenario 1*: Daily maximum and minimum temperatures combined with daily average wind speed, generated from both generators with no consideration of construction working period. In this scenario, weather variables represent the average of weather records over 24 hours.*

Input scenario 2*: Daily maximum and minimum temperatures combined with daily average wind speed; the non-parametric weather generator in this case provided weather series for the specified construction working period (from 8:00 to 17:00). This testing scenario is applied to address the effect of considering weather variables from a specific time interval when estimating construction performance.*

Table 2- 13 Work warm-up schedule for outdoor activities *[18]*

T	No Noticeable Wind		Wind 8 km/h		Wind 16 km/h		Wind 24 km/h		Wind 32 km/h	
C°	Max. work period	No. of breaks	Max. work period	No. of breaks	Max. work period	No. of breaks	Max. work period	No. of breaks	Max. work period	No. of breaks
-26 to -28	(Norm breaks)	1	(Norm breaks)	1	75 min.	2	55 min.	3	40 min.	4
-29 to -31	(Norm breaks)	1	75 min.	2	55 min.	3	40 min.	4	30 min.	5
-32 to -34	75 min.	2	55 min.	3	40 min.	4	30 min.	5	Non-emergency work should cease	
-35 to -37	55 min.	3	40 min.	4	30 min.	5	Non-emergency work should cease			
-38 to -39	40 min.	4	30 min.	5	Non-emergency work should cease					
-40 to -42	30 min.	5	Non-emergency work should cease							
- 43 -	Non-emergency work should cease									

Table 2- 14 Minutes lost per four-hour shift

Temperature C		Wind Speed km/h					
From	To	0	1	8	16	24	32
-43	-65	240	240	240	240	240	240
-40	-43	60	60	240	240	240	240
-38	-39	40	40	60	240	240	240
-35	-37	20	20	40	60	240	240
-32	-34	15	15	20	40	60	240
-29	-31	0	0	15	20	40	60
-26	-28	0	0	0	15	20	40
0	-25	0	0	0	0	0	0

The combination of maximum temperature and average wind speed in each scenario represents the minimum expected loss in minutes per shift. The combination of minimum temperature and average wind speed represents the maximum expected loss in minutes per shift. Figure 2- 35 to Figure 2- 38 illustrate the output results of the first testing scenario; Figure 2- 35 demonstrates that the non-parametric weather generator provided a better estimation in terms of the minimum expected loss in minutes per four-hour shift. However, the parametric weather generator outperforms the non-parametric weather generator when it comes to estimating the maximum expected loss in minutes per four-hour shift, as shown in Figure 2- 36. Figure 2- 37 and Figure 2- 38 illustrate the way in which the estimated values of maximum and minimum expected loss in minutes per four-hour shift differ from the historical average. For the minimum expected loss, the trend of the non-parametric weather generator was similar to that of the historical record when compared to the parametric weather generator. However, the parametric weather generator outperformed the non-parametric weather generator in maximum expected loss.

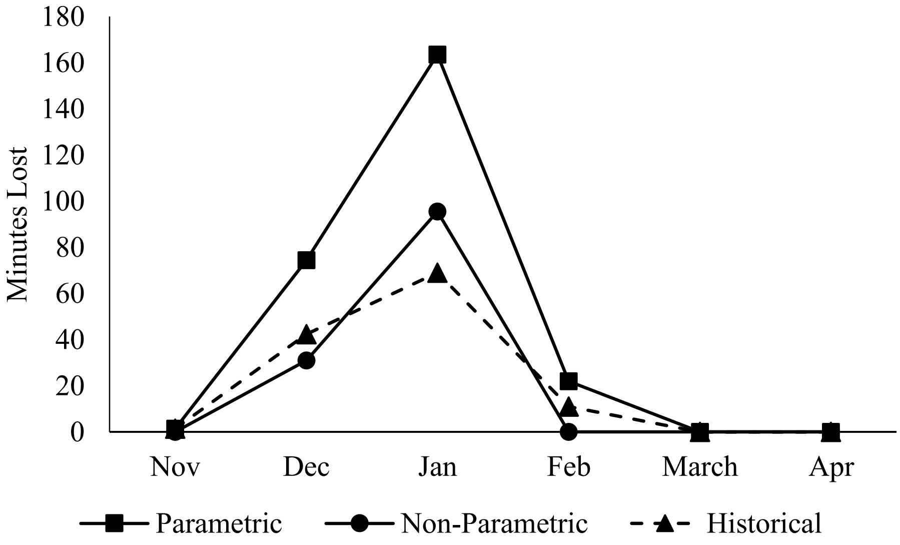


Figure 2- 35 Minutes lost per month due to maximum temperature and average wind speed (with no consideration of construction working period)

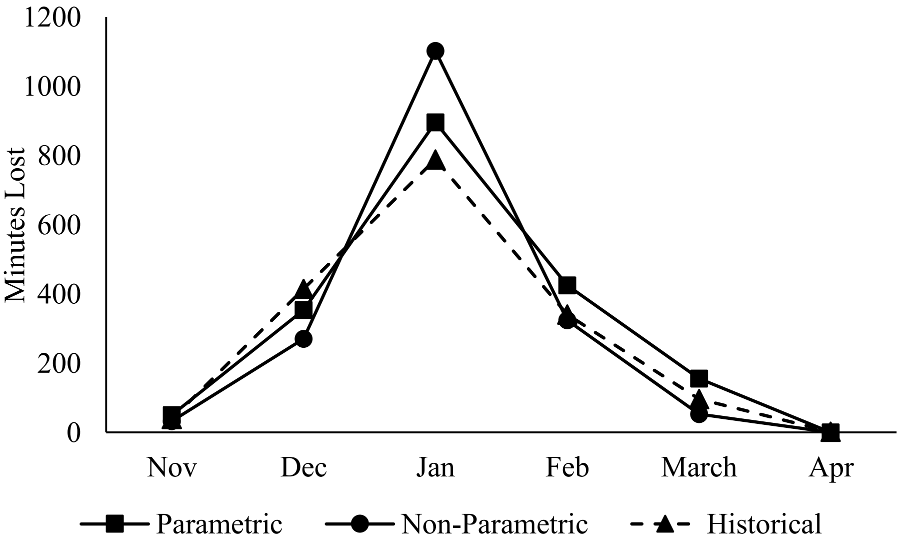


Figure 2- 36 Minutes lost per month due to maximum temperature and average wind speed (with no consideration of construction working period)

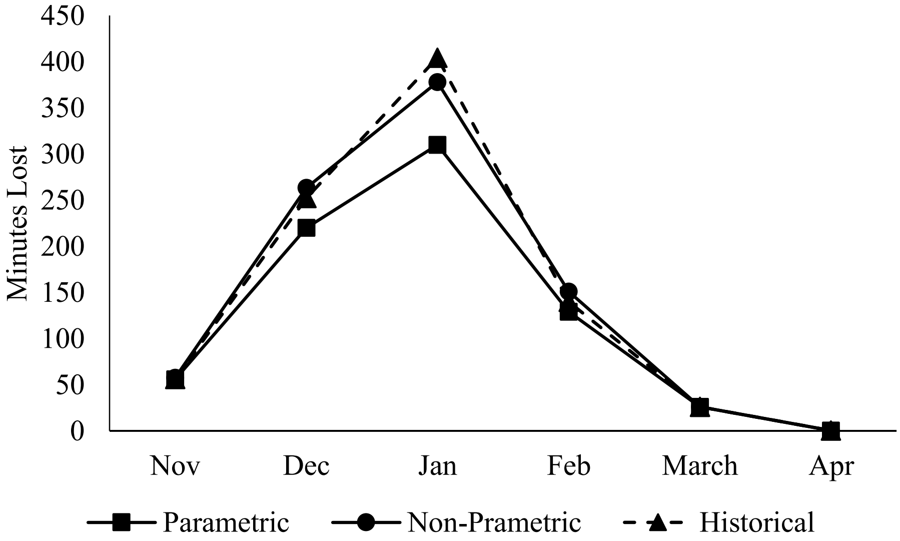


Figure 2- 37 Deviation of minutes lost per month due to maximum temperature and average wind speed from historical average (with no consideration of construction working period)

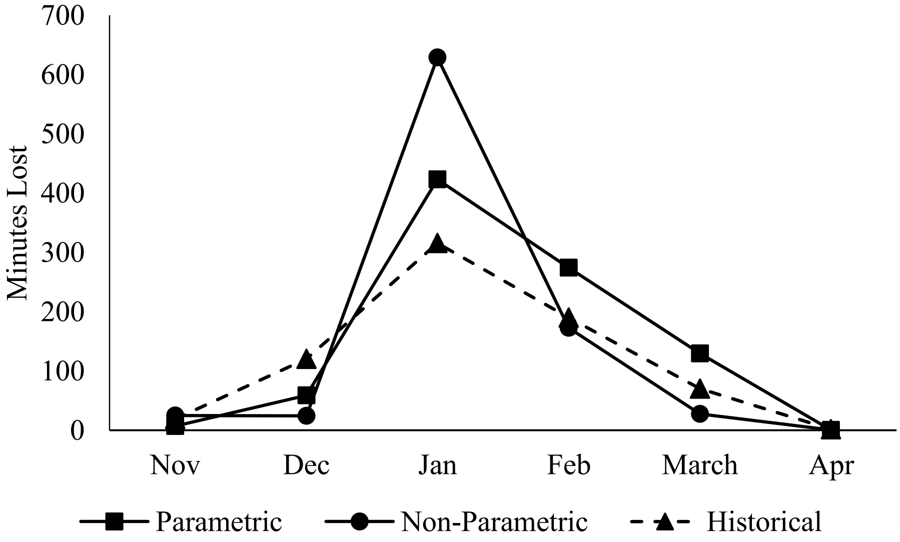


Figure 2- 38 Deviation of minutes lost per month due to minimum temperature and average wind speed from historical average (with no consideration of construction working period)

Figure 2- 39 to Figure 2- 42 illustrate the second testing scenario outputs. Here, the output results contradict those of the first testing scenario. The parametric weather generator performed better than the non-parametric weather generator in estimating the minimum expected loss; however, the non-parametric weather generator provided a better estimation of maximum expected loss. These results indicate that the parametric weather generator provides a better estimation of the maximum effect of temperature and wind speed on construction labour, if no specific construction period is assumed (this means that the average daily values of weather variables are used). However, in the case of estimating the maximum effect of temperature and wind speed in a specified construction period, the non-parametric weather generator provides a better estimation.

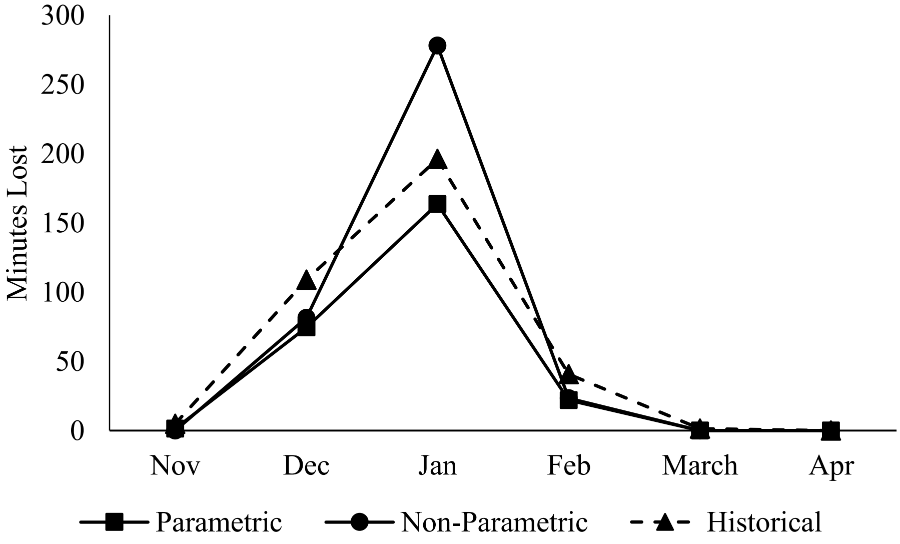


Figure 2- 39 Minutes lost per month due to maximum temperature and average wind speed (with consideration of construction working period)

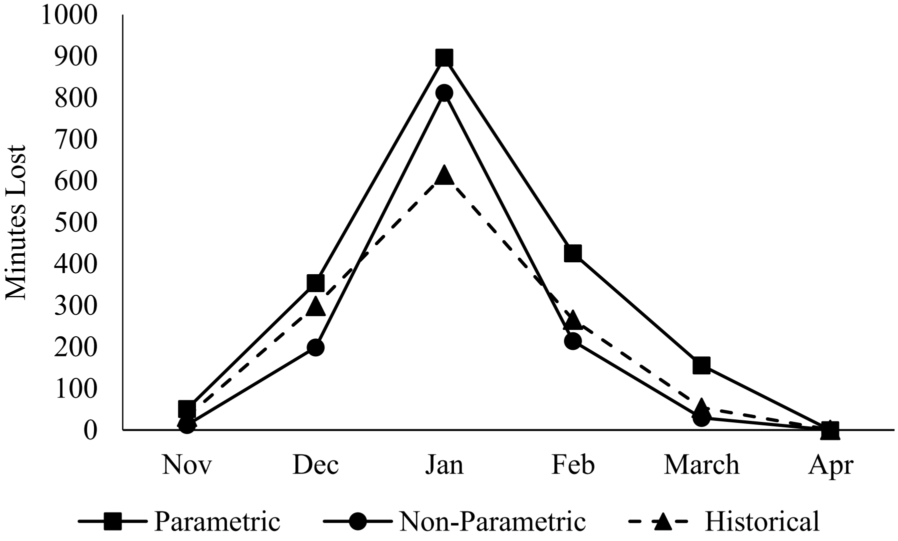


Figure 2- 40 Minutes lost per month due to minimum temperature and average wind speed (with consideration of construction working period)

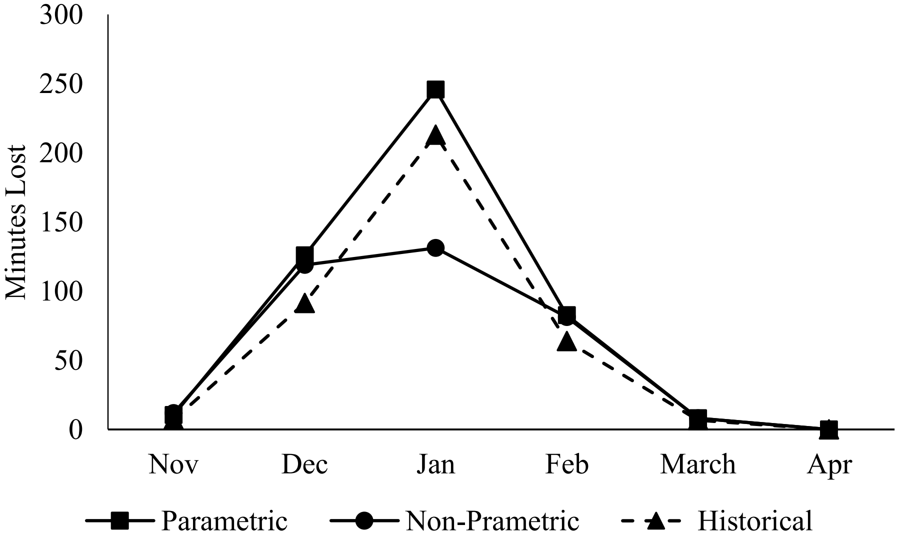


Figure 2- 41 Deviation from historical average of minutes lost per month due to maximum temperature and average wind speed (with consideration of construction working period)

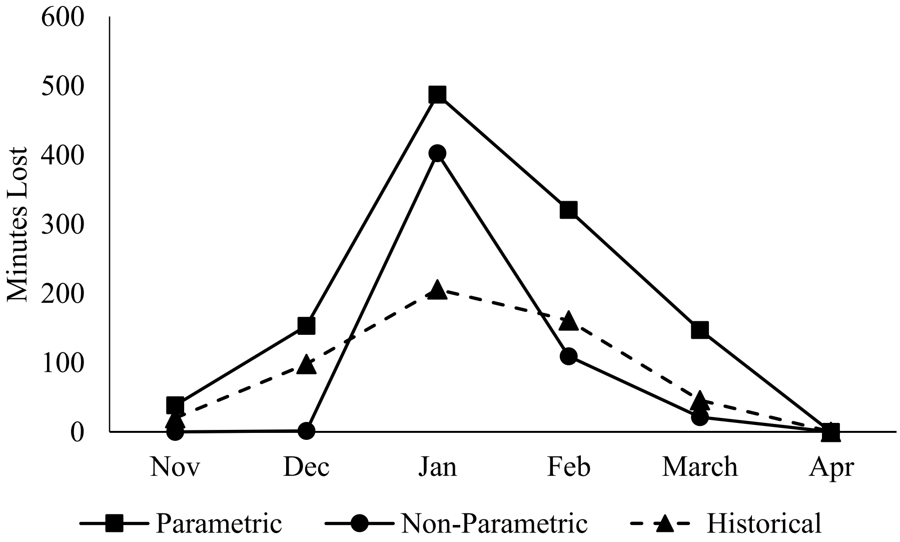


Figure 2- 42 Deviation from historical average of minutes lost per month due to minimum temperature and average wind speed (with consideration of construction working period)

2.4.3.2 Estimating temperature effects on tower cranes

Extreme weather conditions affecting cranes are normally controlled by two weather variables: wind speed and temperature. The wind speed mainly affects the lifted load. The combination of heavy load and high wind speed creates an unpleasant working environment for cranes and construction labourers [42]. This condition may cause either a partial or complete stoppage of crane work onsite. Temperature also causes stoppage due to the allowable operational temperature set for cranes.

There are many types of cranes. All belong to a class of construction equipment. Included in this class mobile and tower cranes. Tower cranes are selected for this study because the occupational health and safety regulations in British Columbia [43] specify winter operational rules: the minimum allowable temperature at which a tower crane can operate is -18 degrees Celsius. This rule is used as a black box model to estimate the loss of operational days in winter. Similar to the previous construction model, two testing scenarios were applied, as follows:

Input scenario 1*: Daily maximum and minimum generated from both generators with no consideration of construction working period.*

Input scenario 2*: Daily maximum and minimum temperatures from both generators; the non-parametric weather generator in this case provided weather series for the specified construction working period (from 8:00 to 17:00).*

Figure 2- 43 to Figure 2- 46 and Figure 2- 47 to Figure 2- 50 illustrate the output results of both scenarios, respectively. In the first testing scenario, both generators provided almost the same estimation of loss in operational days. However, when the differences from historical averages were compared, the parametric weather generator provided a better trend than the non-parametric weather generator. In the case of the second testing scenario, the non-parametric weather generator outperformed the parametric weather generator in both the estimation of loss in operational days and the deviation from historical averages. Therefore, although both generators performed well on a daily basis, the non-parametric weather generator performed better when the construction period was specified.

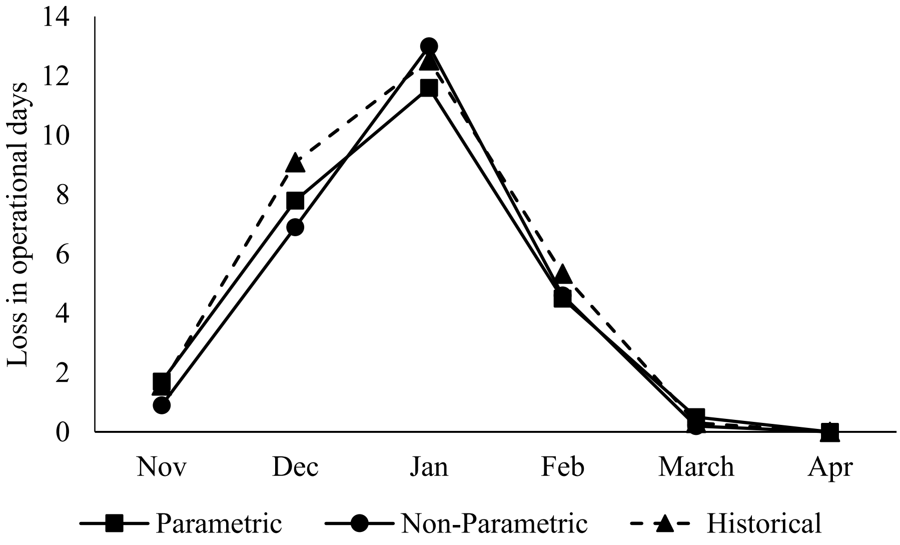


Figure 2- 43 Loss in operational days due to maximum temperature (with no consideration of construction working period)

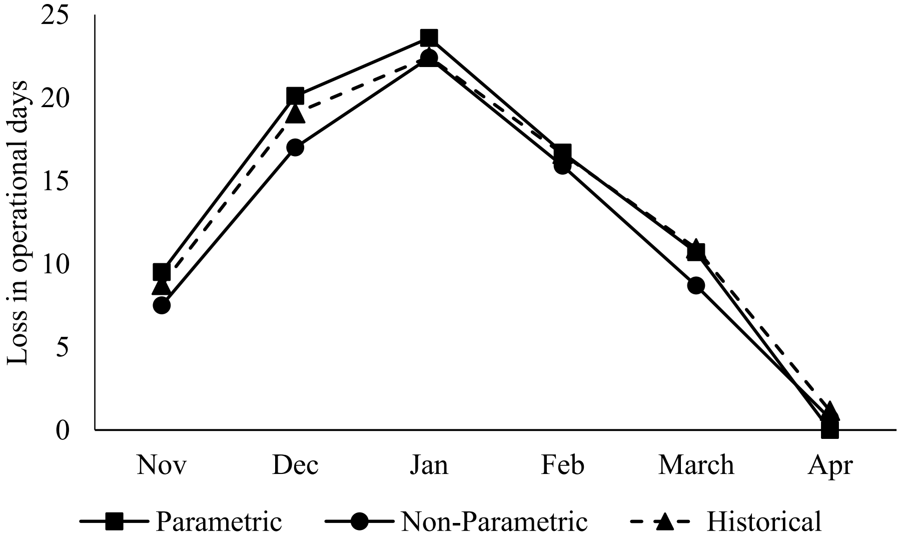


Figure 2- 44 Loss in operational days due to minimum temperature (with no consideration of construction working period)

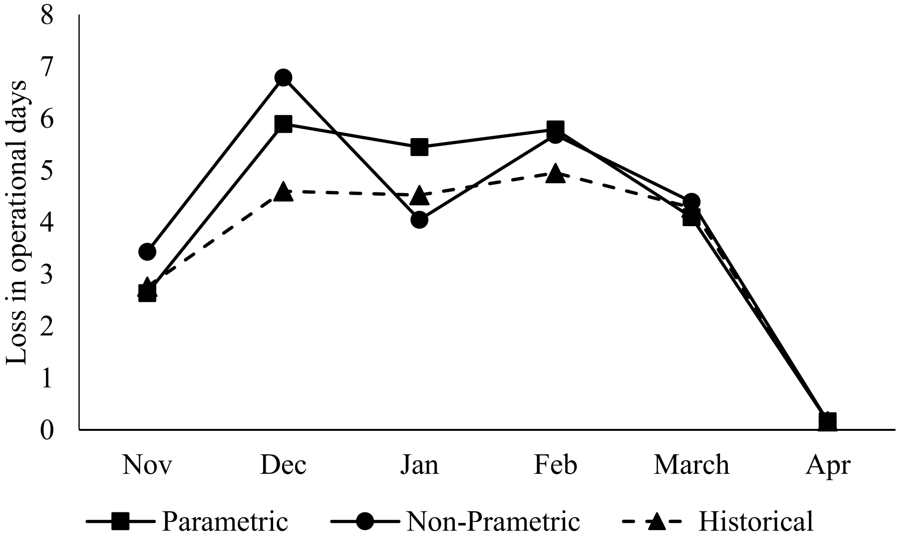


Figure 2- 45 Deviation from historical average of loss in operational days due to maximum temperature (with no consideration of construction working period)

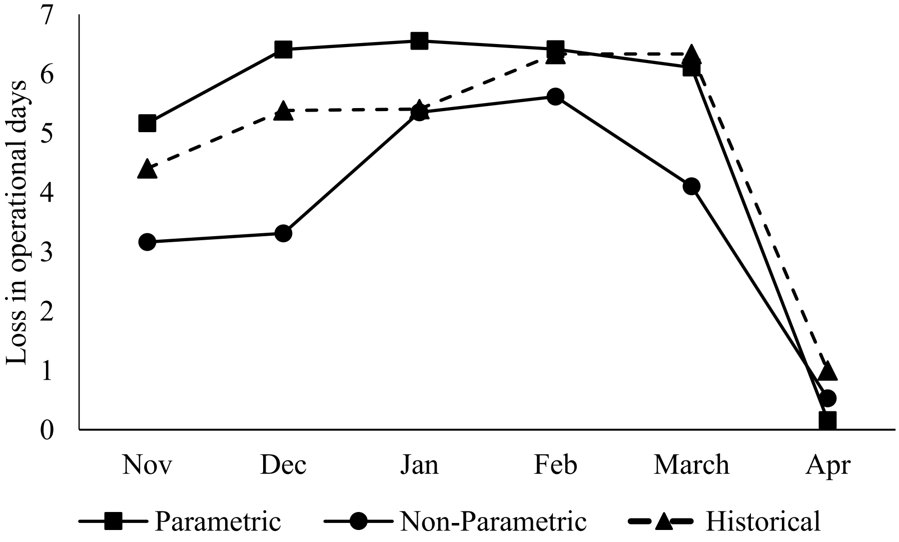


Figure 2- 46 Deviation from historical average of loss in operational days due to minimum temperature (with no consideration of construction working period)

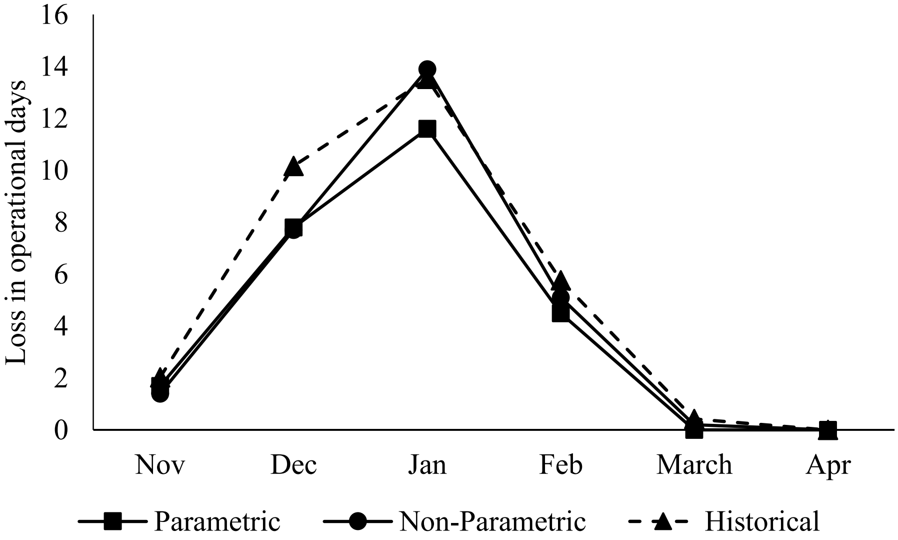


Figure 2- 47 Loss in operational days due to maximum temperature (with consideration of construction working period)

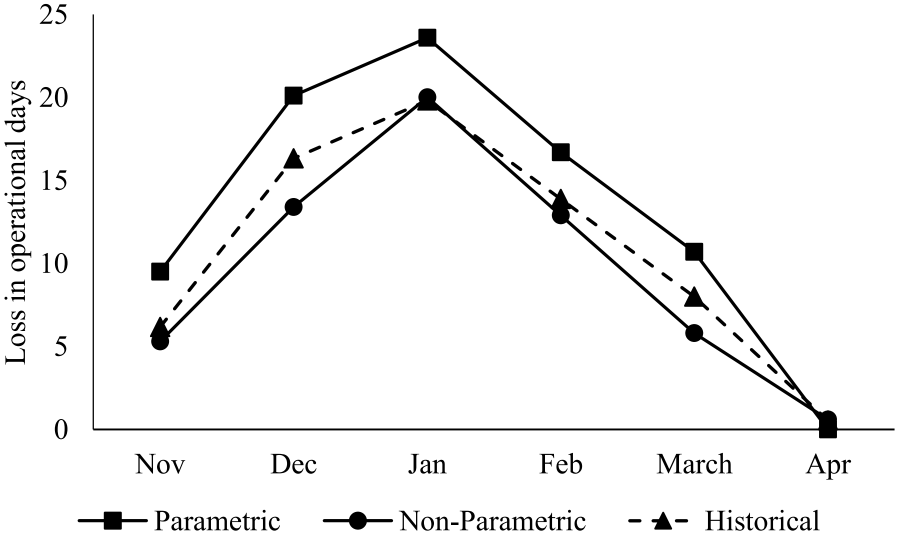


Figure 2- 48 Loss in operational days due to minimum temperature (with consideration of construction working period)

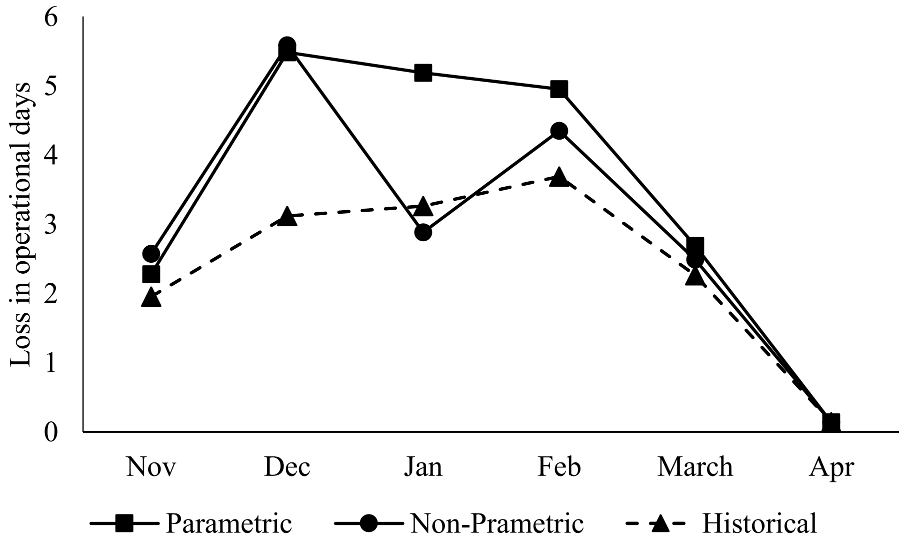


Figure 2- 49 Deviation from historical average of loss in operational days due to maximum temperature (with consideration of construction working period)

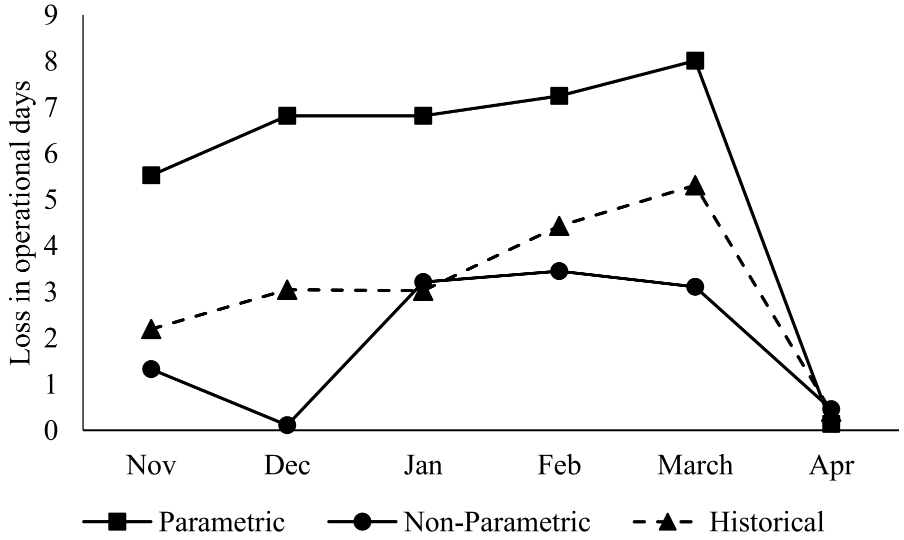


Figure 2- 50 Deviation from historical average of loss in operational days due to minimum temperature (with consideration of construction working period)

2.5 Conclusion

A simplified weather generator using a non-parametric approach, built using the bootstrap sampling technique, was proposed in this study. An experiment was conducted to evaluate the way in which parametric and non-parametric weather generators performed in comparison to each other. Another experiment was performed, this one to evaluate the generators' imperfections when applied on weather-sensitive construction models. In the context of resembling a real weather series, it was found that the proposed approach has the same performance as the parametric weather generator and, moreover, it exhibits better performance than the parametric weather generator for some weather variables, such as maximum relative humidity and minimum temperature. When measuring the generators' imperfections in terms of the weather-sensitive construction model, it was found that the parametric weather generator outperforms the non-parametric weather generator in estimating extreme weather effects in construction labour when compared to outputs generated using historical records. This result applied when there was no specific construction working period identified for the modelling construction operation. However, when a specified construction period was applied, the non-parametric weather generator provided a better estimation of extreme weather effects on construction labour. Likewise, in estimating the extreme weather effects of tower crane operation, the non-parametric weather generator provided a better estimation than the parametric weather generator when a specified construction period was applied. It was found that both generators are reliable in generating synthetic weather series and in modelling construction operations from a daily perspective. However, the non-parametric weather generator outperforms the parametric weather generator when a smaller time-scale flexibility (e.g. hourly) is required in modelling a construction operation.

Chapter 3

Application of the Non-Parametric Weather Generator in Modeling a Construction Operation

3.1 Introduction

Weather variables play different roles in estimating the performance of earthmoving operations. Earthmoving machinery is most affected. For instance, the rolling resistance factor is calculated based on the amount of tire penetration into the ground. Different soils have different tire interaction behaviour. In the winter, snow cover or snow depth affects such interactions and varies the rolling resistance from two perspectives: first, if the snow cover is not packed, the tire penetration rate becomes higher, which results in a higher rolling resistance. Second, if the snow cover is packed, the rolling resistance is lower [44]. In the spring and summer, precipitation is the governing factor for rolling resistance. Precipitation increases the water content in the soil, which in turn increases the tire penetration rate. The increase in rolling resistance decreases hauling truck speed, which affects the productivity of the earthmoving cycle [45]. Furthermore, the increase in rolling resistance has a negative impact on a truck's fuel economy, which means that it has a direct relationship with fuel consumption [46].

Visibility in meteorology is defined as "the greatest distance at which a black object of suitable dimensions can be seen and recognized against the horizon sky during daylight. It could also be seen and recognized during the night if the general illumination were raised to the normal daylight level" [47]. Visibility is an important factor used to estimate a truck's hauling productivity since it controls the maximum speed that can be obtained in the hauling journey [48]. The high wind speed on a snowy or rainy day creates blowing snow that can be a hazard in transportation since it significantly reduces visibility. On a clear summer day, a truck can achieve maximum speed in its hauling journey while on a foggy or snowy winter day truck speed is affected and often limited by visibility on the road [49].

The low temperature in the winter season influences properties of soil, especially its strength. Wet soils are comprised of pores, water, and soil particles. When water in soil pores is subjected to a temperature below the freezing point, it results in changing the state of water from a liquid to solid state. This change gives the soil some ice characteristics like strength, increasing its strength by one or two orders of magnitude [50]. For example, a coarse-grained soil with a medium to high unit weight becomes very strong when water in the soil freezes. This change in soil strength has a direct effect on the excavation process [51]. In contrast, an increase in temperature initiates the thawing process which reduces soil strength [52].

The variation in temperature from one season to another has a direct effect on a truck's operational cost, specifically breakdown repairs and maintenance costs. For example, a truck's performance depends on tire life, and the most significant environmental factor that affects tire life is temperature [53]. Li, Liu and Frimpong [53] identified that dump truck tires are subjected to high stress and deformation when working under severe temperature conditions ranging from 40 to -40 ^{o}C. As a result, tire failure may occur in earthmoving operations [54]. Heavy rain and snow are also considered to be among the environmental factors affecting tires [55]. Temperature also has a negative effect on a wide range of equipment operations. For instance, low temperature has an adverse effect on engine performance; extreme low temperatures increase the number of engine failures and reduce fuel efficiency [56]. Furthermore, low temperatures significantly change the brittleness of metals, resulting in an increase in vehicle breakdowns and machine part failures [57].

Weather variables effects, based on the above discussion, on earthmoving operations is contained within three processes: (1) excavation, (2) loading, and (3) hauling. In the context of a simulation of an earthmoving operation, it is important to integrate such effects to create a realistic simulation environment. In each simulation run, weather variables should be generated carefully so that time dependency between weather variables is preserved. This way each weather variable generated for a certain earthmoving operation (e.g., excavation) will be correlated to other weather variables generated for other operations (e.g., hauling). Furthermore, all earthmoving operations should be controlled by a simulation time step (e.g., hourly or daily), allowing the simulation model to preserve the operational dependency among all resources. Getting all earthmoving operations and a weather generator working in harmony based on a specified time step is a complex task. To accomplish this task, a high-level architecture (HLA) standard is one of the solutions offered in the field of simulation research. This chapter illustrates the integration of weather effects in earthmoving operations through the use of a distributed simulation with HLA standards. The modeled earthmoving operation is related to oil sands' mining operations and the non-parametric weather generation approach described in Chapter 2. This chapter also focuses on demonstrating the effect of temperature on trucks and on the time it takes to repair breakdowns in trucks and excavators. The effect of temperature on breakdown repair durations is analysed using different weather scenarios generated by the embedded weather generator in the simulation model. The results are reported accordingly.

3.2 Overview of distributed simulation and HLA standards

Distributed simulation or parallel/distributed simulation technology is defined as a technology that enables a program to be executed on a system composed of multiple computers [58]. Fujimoto [58] identified four principal benefits for applying distributed simulation: (1) reduced execution time, (2) geographical distribution of simulation or computer components, (3) integrating multiple simulators, and (4) fault tolerance. The first principal benefit, "reduced execution time," was the actual main objective of proposing the distributed simulation. Thus a time effect technology to develop and execute large simulation programs was highly desired.

In real world applications, the flexibility requirements of the distributed simulation extended beyond the geographical distribution of simulation components and the integration of multiple simulators. An interaction between simulation components was of great interest. The US military initiated such a requirement because it wanted effective and economical ways to train personnel. The military's main objective was to develop a virtual environment capable of allowing interactions between geographically distributed hardware and personnel in a real-time framework [59]. This led to a proposal for a Distributed Interactive Simulation (DIS). DIS is "an infrastructure that enables heterogeneous simulators to interoperate in a time and space coherent environment" [60]. Despite the ability of DIS to enable interaction between different simulation components, it was associated with challenges related to uncontrolled latencies and lack of time management services [61]. Furthermore, building simulation components in different environments limit the reusability of the simulation system. As a result related to these challenges, an HLA standard was developed to improve the concept of standardization of simulation building and to improve data processing and acquisition through a time management infrastructure.

The HLA standard is a general purpose framework that supports the simulation of a system composed of multiple simulation components working independently [62]. It was developed by the United States Department of National Defense with the main objectives to incorporate interoperability (the ability to integrate different simulation components created in different development environments), modularity (the standardization of the framework so it can be adopted in different applications), and reusability (the ability to use the simulation component in different scenarios or applications) into long-term simulation objectives [63]. It includes three core components: federation and federates rules, the federate object model (FOM), and HLA interface specification.

Federation and federates rules are a set of rules or conventions that must be followed to regulate interactions between different federates (a federate represents an independent simulation component) during the execution stage. Each federate may contain objects of different attributes, interactions or both and all are standardized by the FOM. The FOM describes all sets of objects, attributes, and interactions which are shared across the federation (a federation contains multiple federates). The Simulation Object Model (SOM) identifies what objects, attributes, or interactions are required by each federate in the simulation. These two objects models FOM and SOM are documented using a standard form called the Object Model Template (OMT), which is shared by all federates [64]. Federates can either publish or subscribe objects or interactions from other federates. This process is controlled by the HLA interface specification that describes the runtime services. These runtime services is provided by the Run Time Infrastructure (RTI) which is federates coordinator capable of synchronizing different federates time models and coordinate the exchange of events (objects and interaction) between them at a predefined point in time.

3.3 Oil sands mining process in Alberta, Canada

The oil sands regions, mainly located in the province of Alberta, Canada, are the fastest growing area in the world of developing petroleum resources [65]. Two main production processes are applied to oil extraction: (1) in-situ and (2) surface mining [66]. The in-situ technique uses a steam injection method to heat the oil, and after which it can be pumped. Meanwhile, in surface mining, large shovels and trucks are used to extract oil sands from the surface. Open pit mining processes, as shown in Figure 3- 1, are usually performed when the oil sand is located near the surface, which is the case in Alberta; 20 % of oil sands reserves are located less than 75 meters underground [67]. The open-pit mining process used in the oil sand mining involves the following:

1. Ore material collection trucks: in this step, the oil sand is loaded into hauling trucks via large shovels and then transported to crushers.
2. Material handling (includes crushing conveying): in this step, the oil sand is stored as earth clumps and is moved to crushers on conveyors to produce small sizes of oil sand material.
3. Slurry conditioning and transfer: in this step, hot water is added to the crushed oil sand to produce oil sand slurry, which is then transferred to the extraction process.
4. Extracting: in this step, the bitumen is separated from the oil sand slurry by adding more hot water. The bitumen is then allowed to settle in the separation vessel.
5. Tailings, froth treatment: in this step, the by-product of the oil sand separation process is transferred to the oil sand tailings ponds and the extracted bitumen froth is further diluted and refined.
6. Upgrading: in this step, the extracted bitumen is transformed into a synthetic crude oil so it can be transferred to refineries to produce oil products.

The ore material collection step involves an earthmoving operation; it uses shovels and trucks as the main resources. These resources are exposed to the environment which means that changes in weather conditions may affect their productivity. Therefore, this step has been selected to be modeled and used as an illustration of the application of the non-parametric weather generator in an earthmoving operation. In order to better understand the earthmoving operation in oil sand mining and its resources, and to clearly define the simulation components of the operation, a discussion was conducted with experts in this field.

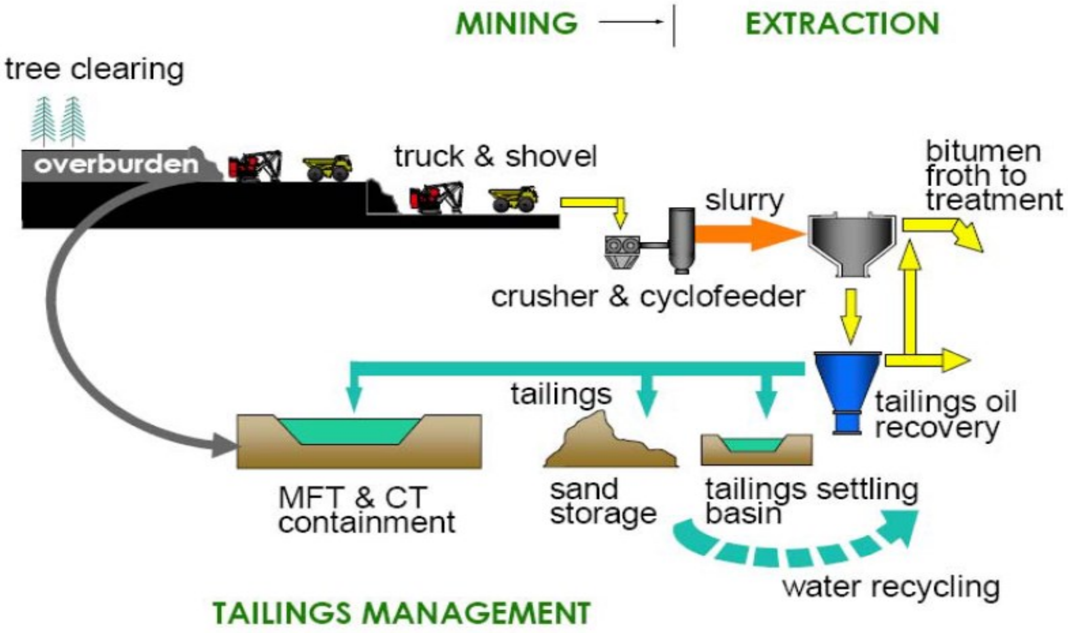


Figure 3- 1 Oil sand open pit mining process [65]

3.4 The development of the mining earthmoving operation model

The earthmoving process cycle, as per the discussion with experts, involves loading ore material into the hauling trucks using shovels/loaders. Trucks move the ore material from the excavation pit to the dump pit at the extraction facility, unload the ore material in the dump pit, and then return to the excavation pit. The experts agreed that the simulation model should contain three major components: excavators, trucks, and equipment breakdown and maintenance.

The earthmoving operation is comprised of resources integrated to develop an earthmoving process cycle. Each resource has its own earthmoving operational cycle. For instance, hauling soil from an excavation pit to the dumping site and back again to excavation pit represents a full hauling cycle for a truck. On the other hand, excavators/shovels operate in two different locations, each with a different operational cycle. The first cycle is located at the mining pit where excavators excavate oil sand and load it into the trucks. The second cycle is located at oil sand earth clumps where excavators/shovels move the oil sand to crushers. These earthmoving operational cycles are operated continuously by different types of trucks and loaders to maintain high productivity and lower operational cost. The major factor influencing the equipment performance is the unanticipated equipment's breakdown event. Therefore, the simulation model of the mining earthmoving operation should be allowed to test the operation under different resource scenarios.

The modeled mining operation is located in a cold, harsh environment; therefore, the experts asked to integrate a weather effect into the model of the mining earthmoving operation. Since weather can change dramatically throughout the day and the mining operation runs nonstop (i.e., for 24 hours), it was agreed that the simulation time step should represent a working hour and the weather generator should be developed in a way to provide hourly basis weather variables. Moreover, it was decided that all mining earthmoving operational cycles should run and interact with each other on an hourly basis. This was achieved by applying HLA standards to regulate the processing time. It was concluded from the discussion with experts that (1) the simulation model should have six simulation components interacting with each other, and (2) each simulation components should consider the weather variable that affects its performance. These components with the simulation structure are shown in Figure 3-2.

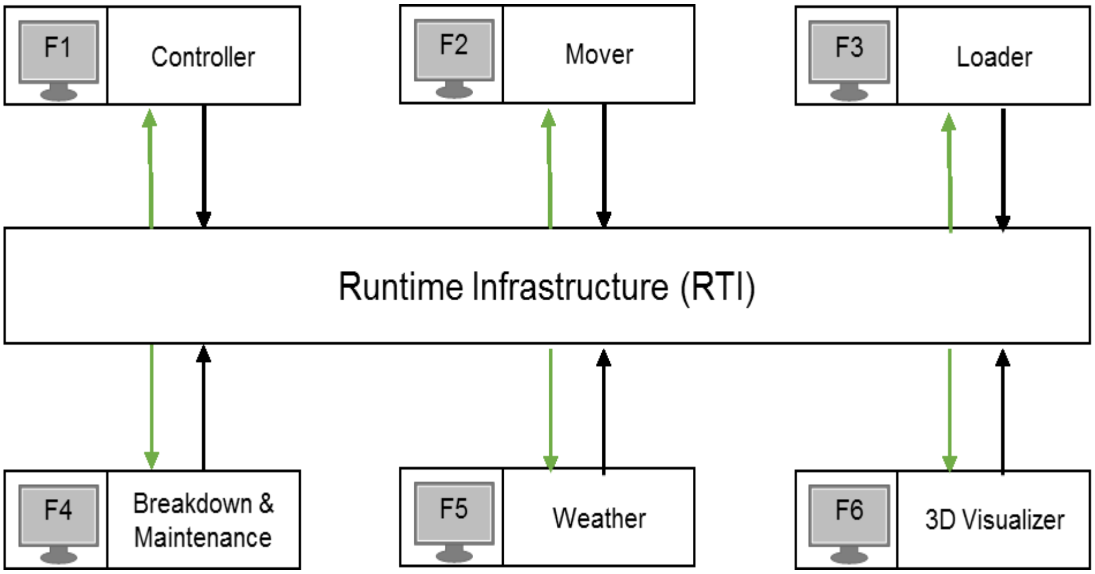


Figure 3- 2 Earthmoving simulation structure

Each simulation component represents a federate in the mining earthmoving federation and is developed by a different team of researchers. The work description of each federate is as follows:

1. Controller: responsible for initializing the simulation model, defining testing scenarios, and analysing the operation performance.

2. Mover: responsible for simulating the hauler behaviour in its cycle.
3. Loader: responsible for simulating the loader behaviour such as loading and dumping.
4. Equipment breakdown and maintenance: responsible for simulating the trend of breakdown and maintenance in the truck and loader cycles.
5. Weather: responsible for generating dynamic and realistic weather variables.
6. 3D visualizer: responsible for animating the earthmoving operation and visually identifying the location of the trucks and loaders.

After all federates of the mining earthmoving federation are identified, the SOM for each federate is constructed so that the object class and its attribute, whether needed or provided by the federate, is clearly identified. Also, an investigation was conducted to determine which weather variables are affecting each federate and how. Once all SOM are created, they are all combined to create the mining earthmoving FOM. Table 3- 1, below, shows a sample of the FOM containing weather variables as an interaction class called "CurrentWeather." This interaction class is published by the weather federate to provide others with a block containing all weather variables required in the simulation process.

Table 3- 1 "CurrentWeather" interaction class and its attributes in the FOM (S=subscribe, P=publish, PS= publish and subscribe)

Attribute	Controller	Mover	Loade r	Breakdown & Maintenance	Weathe r	Visualize r
Interaction Class						
Temperature WindSpeed Visibility Snowfall Precipitation	S	S	S	S	P	S

3.5 Weather federate

3.5.1 Historical weather database

The bootstrapping approach to randomly generate weather variables is integrated into the weather federate. The weather federate is linked to an historical weather database containing the location of operation. It reflects weather parameters to other federates. The location of the oil sand mining operation is Fort McMurray, Alberta, and the historical weather database is extracted from the Environment Canada website. The weather variables listed in the database are those requested by other federates and listed in the FOM. The weather variables are as follows:

- Temperature
- Wind speed
- Visibility
- Precipitation

- Snow depth

Figure 3- 1, below, shows the weather database breakdown structure and two tables created in the database to maintain hourly and daily weather variables in accordance with the need for the mining earthmoving simulation model. The daily weather forecast table classifies the temperature into three groups: maximum, minimum and average temperature of each day. Wind speed is classified into two groups: maximum and average. This classification of weather parameters is important to create different testing scenarios of weather conditions and to study their impact on the mining earthmoving operation.

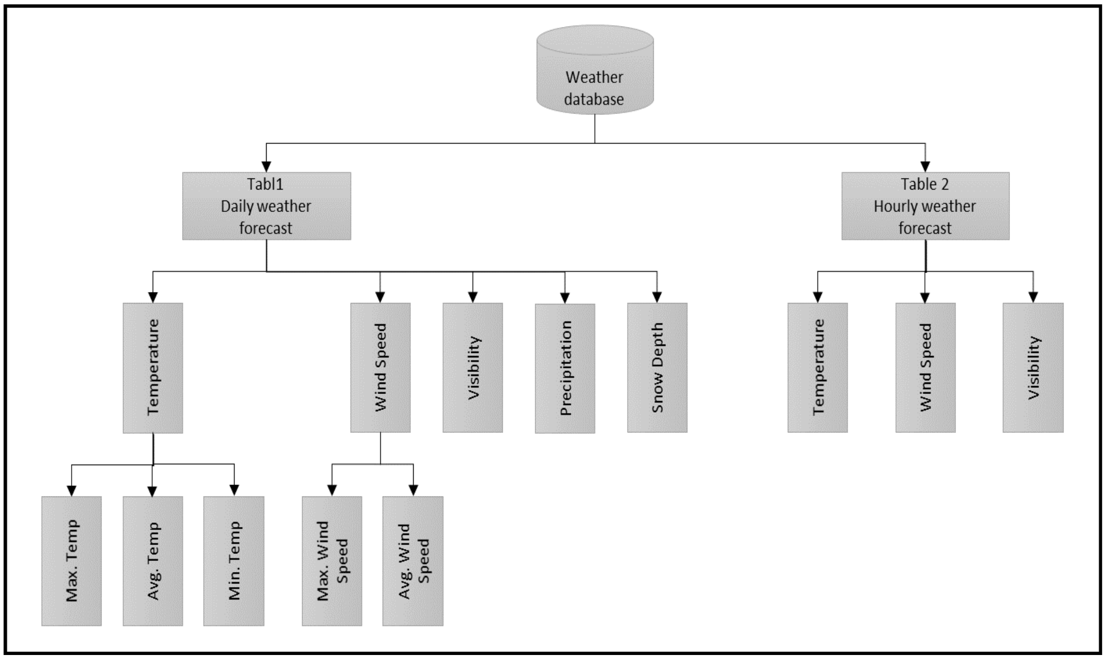


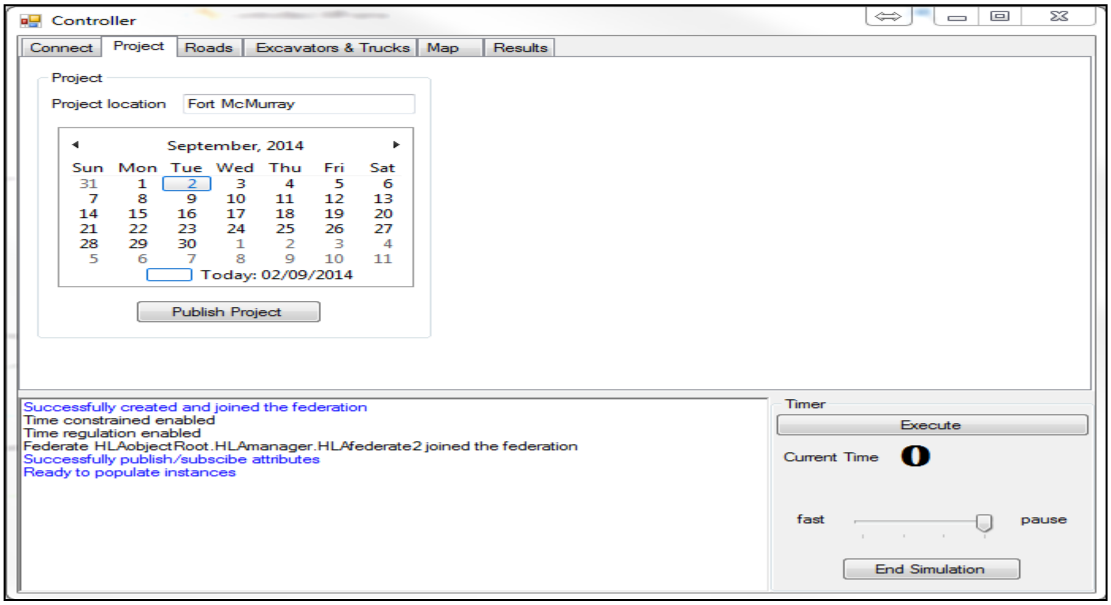
Figure 3- 3 Weather database breakdown structure

3.5.2 The weather generation process

The weather generation in this study extracts real historical weather variables from the database for the purpose of preserving correlations and dependencies among meteorological variables. Hourly and daily forecast tables in the Fort McMurray weather database provide flexibility to generate different scenarios. In this stage of the research, the weather federate provides other federates with weather parameters in the following three scenarios:

1. The first scenario (SC1) generates weather variables from the database based on the random selection of a year between 1961 and 2002. The hourly generated weather variables represent the daily average. This scenario is considered the conservative daily scenario.
2. The second scenario (SC2) is also based on generating weather variables from randomly selected years. However, the generated hourly weather variables represent the minimum daily values in winter and maximum daily values in summer. This scenario is considered the extreme daily scenario.
3. The third scenario (SC3) is based on generating weather variables from randomly selected years. The generated hourly weather variables in this scenario represent the actual values of weather variables which have been experienced at that particular hour. This scenario is considered the actual hourly scenario.

The weather generation process starts, as shown in Figure 3- 4, by first determining the location and the expected starting date of the operation. After the user enters the location and the date of operation, the weather generator randomly selects the year of operation from the database (see Figure 3- 5). The generator is initialized based on year, month, day, and location. These values are used to indicate the weather records to be generated for other federates. All weather variables are provided on an hourly basis and in accordance with the selected testing scenario, except for snow depth and precipitation, because the measured records of those variables are normally presented in the form of accumulated amounts per day. Snow depth and precipitation are generated at hour 0 AM of each simulated day. The model assumes that the operation works for 24 hours, and each simulation run represents one minute. Therefore the weather federate updates its weather interaction class values after the run time of every 60 simulations and moves to the second day forecast when a full day of operation is completed.



Project location: Fort McMurray

Today: 02/09/2014

Current Time 0

```
Successfully created and joined the federation
Time constrained enabled
Time regulation enabled
Federate HLAobjectRoot.HLAmanager.HLAfederate2 joined the federation
Successfully publish/subscibe attributes
Ready to populate instances
```

Figure 3- 4 Federation interface

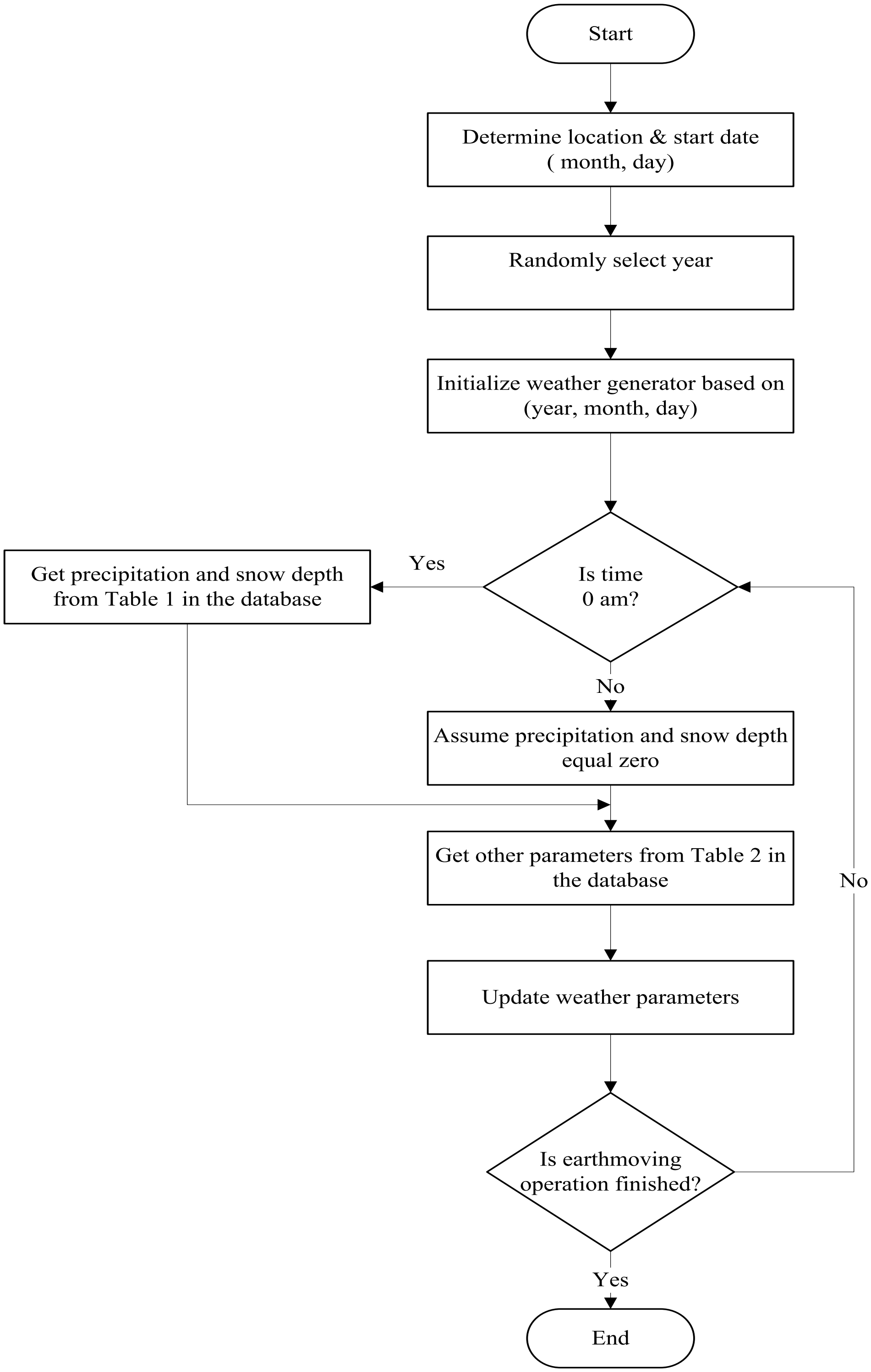

Start
Determine location & start date (month, day)
Randomly select year
Initialize weather generator based on (year, month, day)
Is time 0 am?
Yes
Get precipitation and snow depth from Table 1 in the database
No
Assume precipitation and snow depth equal zero
Get other parameters from Table 2 in the database
No
Update weather parameters
Is earthmoving operation finished?
Yes
End

Figure 3- 5 Weather generation flow chart

3.6 Simulation run and testing scenario results

3.6.1 Simulation run

The simulation run of the operation is controlled and initiated by the controller federate. The controller federate is composed of two parts: (1) the interface and (2) the federate. The interface allows the user to enter the operation attributes such as project information (includes project location and starting date, seen in Figure 3- 4), road condition, truck types and their quantity, excavator types and their quantity, and the topological map of the site. This information is provided by the user and represents the attributes that are subscribed by the controller federate. These attributes are published by the controller to the other federates in the simulation model. The controller federate uses this information to start the simulation by populating the number of instances of each object class. For example, nine instances of object class “Truck” and three instances of object class “Excavator” are populated, each associated with their attributes such as the truck model and its capacity to simulate the mining operation. Following the population stage of the resources class objects, the road section that hosts the hauling and returning routes of trucks is initiated. The road section is composed of multiple segments, each of which has different attributes such as road segment materials and layouts. The road section length used to haul and return trucks is equal to 2.2 km.

The mover federate subscribes road conditions in terms of layouts and materials, truck models, and breakdown and maintenance states from other federates. The mover federate publishes a truck’s location dynamically using road layouts and truck speed.

The loader federate subscribes both trucks and excavator to load trucks with soil. If all excavators are busy loading pre-arrived trucks, the mover federate will acquire the ownership of the newly arrived truck until it is filled with soil and released for hauling. Both the mover and the loader federates performances are controlled by the availability of trucks and excavators. These resources are associated with breakdown and maintenance events that control the percentage of utilization for each resource. These events are controlled by the breakdown and maintenance federates, which employ four crews for repair and maintenance. It uses information (shown in Table 3- 2) to create a breakdown and maintenance events in the mining earthmoving operation.

Table 3- 2 Durations of expected breakdown and maintenance events for trucks and excavators

Event	Duration
Truck breakdown interval	Exponential (200)
Truck maintenance interval	Constant (400)
Truck repair time	Uniform (20,24)
Truck maintenance time	Triangular (18,21,24)
Time increase in truck repair due to a specified temperature threshold value (T)	20%
Excavator breakdown interval	Exponential (250)
Excavator maintenance interval	Constant (350)
Excavator repair time	Uniform (20,24)
Excavator maintenance time	Triangular (18,21,25)
Time increase in excavator repair due to a specified temperature threshold value (T)	20%

Table 3- 2 shows that the breakdown repair time is controlled by the temperature experienced in the operation location. The repair time for both the trucks and excavators is expected to increase by 20% depending on a pre-specified temperature threshold value(T). This percentage is experts-driven for a temperature less than or equal to -30 C^{o}. However, Nguyen et al. [68] highlighted that in the context of construction contracts, no single temperature threshold value exists because different construction projects have different working characteristics. Kohen and Brown [17], in the context of labour productivity, identified -29 C^{o} as the temperature threshold value at which labors must stop work is at and -18 C^{o} as the threshold value at which labour productivity starts deteriorating at. A sensitivity analysis which will apply a range of temperature threshold values (T) from -18 C^{o} to -30 C^{o} will be used to study the effect of temperature on breakdown events.

The breakdown and maintenance federate subscribes temperature so that a temperature-based analysis can be performed. As previously described, the temperature is published within the interaction class called “CurrentWeather” that is provided by weather federate (refer to Table 3- 1). The weather federate publishes the “CurrentWeather” interaction class based on three scenarios: (1) SC1 (the conservative daily scenario), (2) SC2 (the extreme daily scenario), and (3) SC3 (the actual hourly scenario. To analyze these scenarios combined with the temperature limit sensitivity analysis, a performance benchmark result for the trucks and excavators is generated based on the assumption that the working condition will not achieve any temperature threshold value. Figure 3- 6 and Figure 3- *7* show the performance benchmark results for both trucks and excavators running for a total duration of 8760 hours (a one-year working period). The average working duration of all trucks represents 83% and their breakdown and maintenance averages are 10% and 7% respectively. A similar percentage is found for excavators. Refer to Appendix C for a detailed results description for each truck and excavator working duration, number of breakdowns and maintenance, and breakdown repair and maintenance durations.

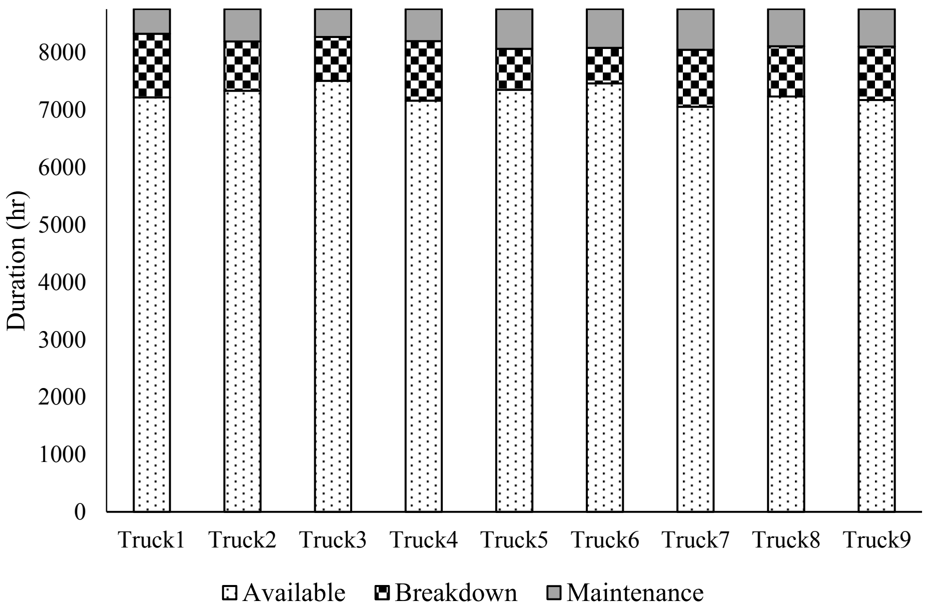


Figure 3- 6 Performance benchmark results for trucks

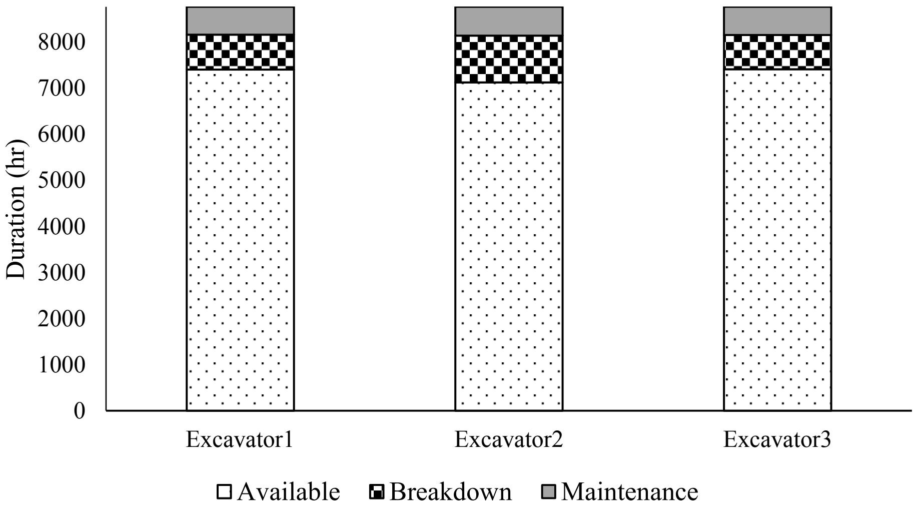


Figure 3- 7 Performance benchmark results for excavators

3.6.2 Results of scenarios

The mining earthmoving operation was tested under three different scenarios. The weather federate generated 10 randomly selected years, and an hourly based "CurrentWeather" interaction class was published to other federates. The effect of temperature on truck and excavator breakdown events was tested under different temperature limit values (T) ranging from -18 C^{o} to -30 C^{o}. The analysis was individually performed for each truck and excavator participating in the simulation. However, the results are summarized to reflect the overall breakdown repair duration expected for a truck or an excavator. The result includes the minimum, average and maximum expected breakdown repair durations for a truck and excavator in the operation.

Figure 3- 8 and Figure 3- 9 show the breakdown repair durations for a truck and an excavator respectively. Figure 3- 8 (a) shows the minimum breakdown repair duration expected for a truck. The results show that using the daily minimum temperature (SC2) generates on average a repair duration that is 1.2 % more than SC1, SC3, which is equivalent to 7.5 more hours. Comparing SC1 (average daily temperature) to SC3 (actual hourly temperature), both scenarios provided the same breakdown repair durations when the temperature limit value (T) was less than or equal to -21 C^{o} and SC3 generated a slightly higher repair duration when (T) was greater than -21 C^{o}. Comparing the total averages of the minimum expected repair duration of each scenario with the truck breakdown benchmark result (616.5 hrs) led to the following:

(1) SC1 generated a 0.05% more repair duration, (2) SC2 generated a 1.32 % more repair duration, and (3) SC3 generated a 0.16% more repair duration.

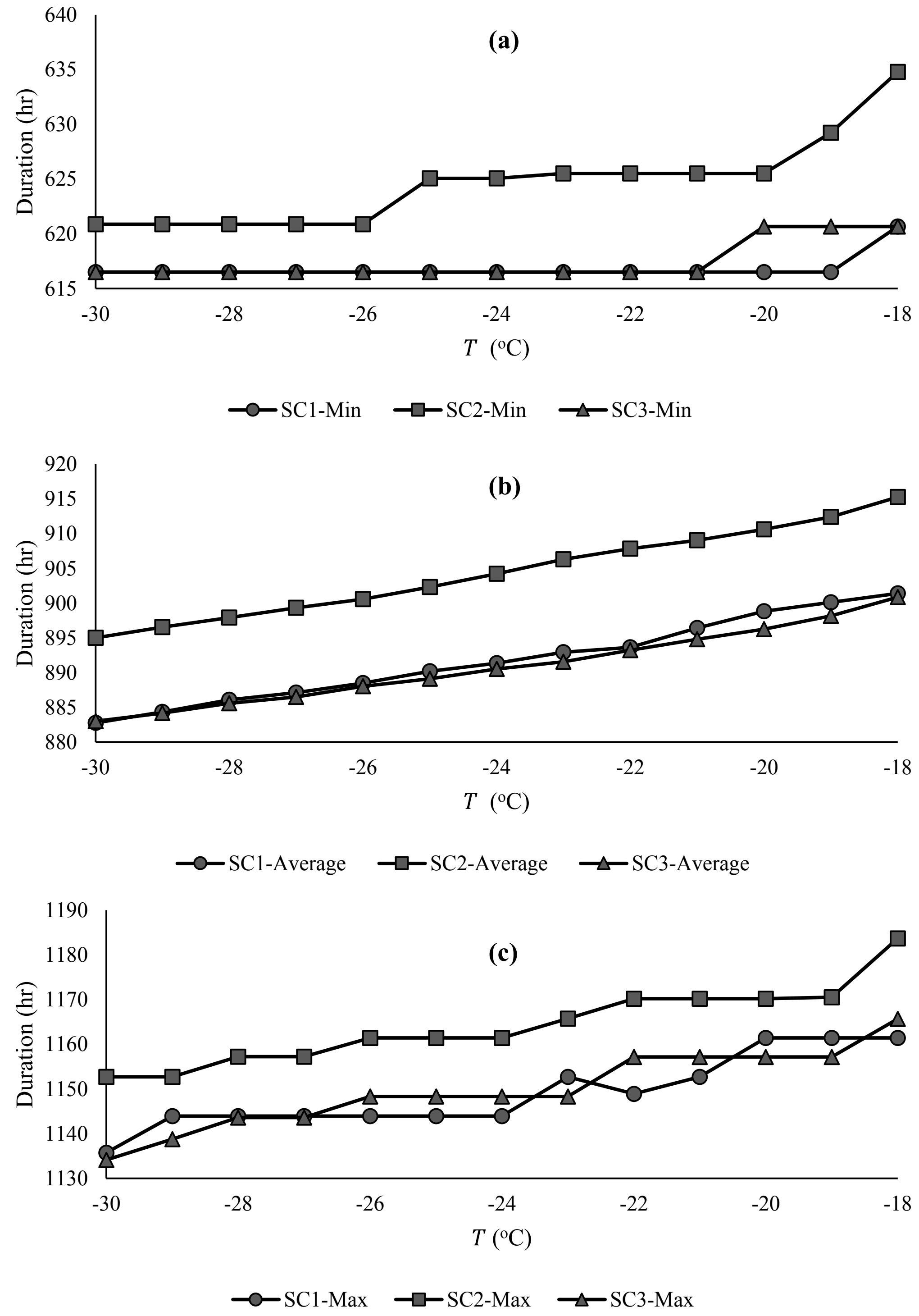


Figure 3- 8 Truck breakdown repair durations under three testing scenarios (SC1, SC2, and SC3) and on different temperature limit values (T); (a) the expected minimum repair durations, (b) the expected average repair durations, and (c) the expected maximum repair durations

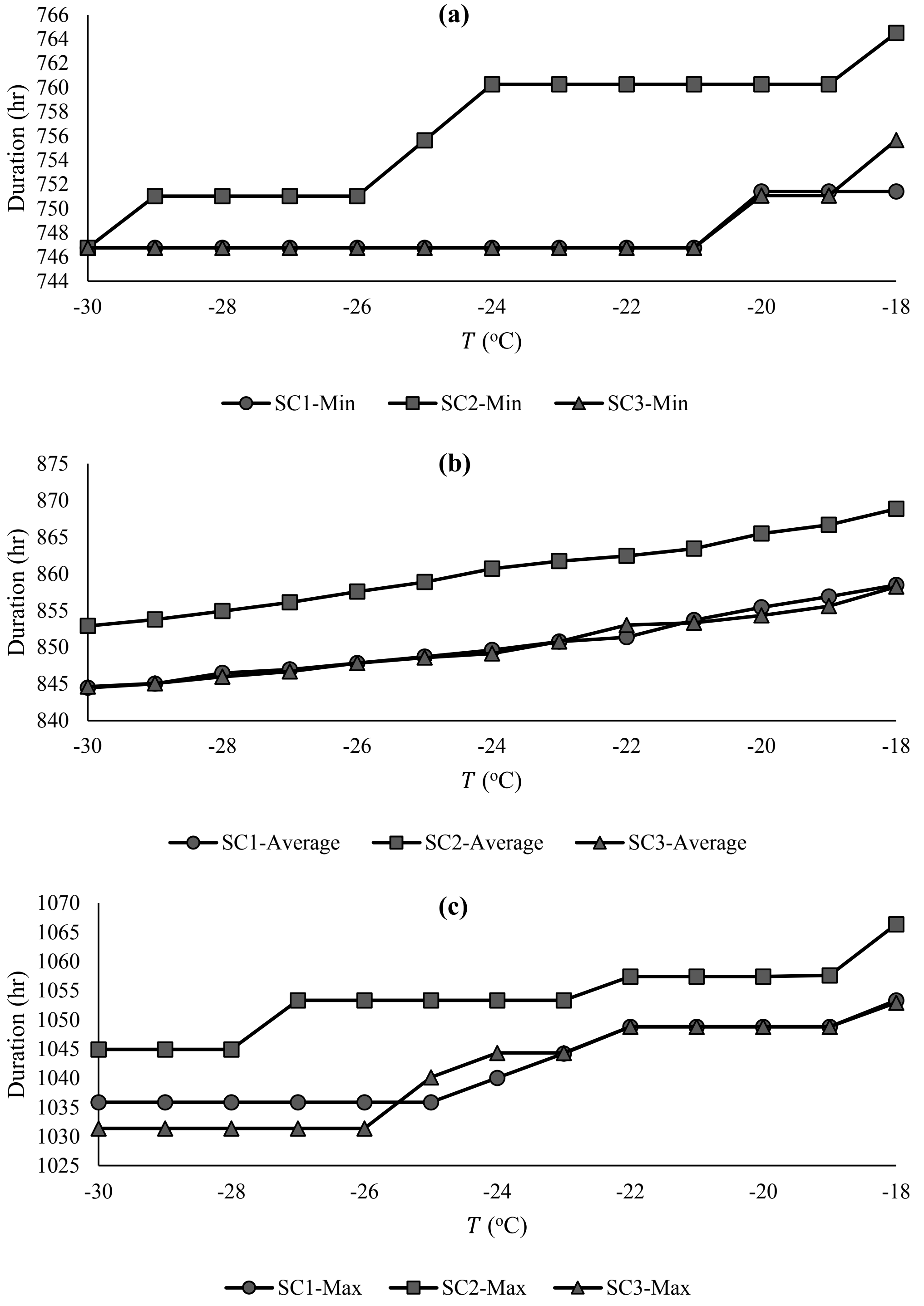


Figure 3- 9 Excavator breakdown repair durations under three testing scenarios (SC1, SC2, and SC3) and on different temperature limit values (T); (a) the expected minimum repair durations, (b) the expected average repair durations, and (c) the expected maximum repair durations

Figure 3- 8 (b) shows the average breakdown repair durations for a truck in the mining earthmoving operation. SC2 generates a 1.5 % more repair duration than SC1 and SC3, which is equivalent to 13.5 more repair hours. Both SC1 and SC3 generated the same breakdown repair duration, equal to 892 hours when tested against different temperature limit values (T). Comparing the total averages of the expected average breakdown repair duration with the truck breakdown benchmark result (876.7 hrs) led to the following: (1) SC1 generated a 1.7% more repair duration, (2) SC2 generates a 1.6 % more repair duration, and (3) SC3 generates a 3.1 % more repair duration. In the context of calculating the maximum expected breakdown repair duration, shown in Figure 3- 8 (c), the results of SC1 and SC3 show different behavior: SC3 generates a slightly higher breakdown repair duration. The percentage of differences between SC2 and both SC1 and SC2 is 1.3% and the comparison with the truck breakdown benchmark result (1108.7 hrs) results in (1) SC1 and SC3 generating approximately 3.5% more breakdown repair duration, and (2) SC2 generating 4.8% more breakdown repair duration.

Figure 3- 9 (a), (b), and (c) show the expected minimum, average, and maximum excavator breakdown repair durations. Figure 3- 9 (a) and (b) show that scenarios SC1 and SC3 have a similar behavior with respect to temperature limit values (T). Meanwhile, they exert different behaviour when calculating the maximum breakdown repair duration; SC1 generates higher repair durations when the temperature limit value is less than or equal to -26 C^{o}. On the other hand, as shown in in Figure 3-9 (a), (b), and (c), SC2 generates a 1.1% higher repair durations than SC1 and SC3.

The conclusion results in this section are shown in Table 3- 3. Based on calculating the expected breakdown repair duration with respect to different temperature limit values (T) and based on different temperature scenarios (SC1, SC2, and SC3), the expected contribution of each resource breakdown repair duration in the overall mining earthmoving operation is:

1. Of the total operation duration for each working truck, 7.04 % to 13.29 % of contributed to the breakdown repair duration, and
2. Of the total operation duration for each working excavator, 8.54 % to 12.03 % contributed to the breakdown repair duration.

Table 3- 3 Expected percentages of breakdown repair durations of each scenario in the mining earthmoving operation

Scenario	% of Breakdown Repair Duration					
	Trucks			Excavators		
	Min	Average	Max	Min	Average	Max
SC1	7.04	10.18	13.12	8.54	9.71	11.90
SC2	7.13	10.32	13.29	8.63	9.82	12.03
SC3	7.05	10.17	13.13	8.54	9.71	11.88

3.7 Conclusion

A distributed simulation approach with HLA standards has been used to model the earthmoving operation of oil sand mining. The model integrated different simulation components including trucks, excavators, breakdown and maintenance, and weather simulation components. The weather effect on truck and excavator breakdowns was addressed and modeled. Furthermore, the weather generator provided different weather testing scenarios to analyze the truck and excavator breakdown repair durations. The weather-based breakdown analysis provided a range of the percentage of breakdown repair duration that trucks and excavators may experience in a one-year mining operation.

Chapter 4

Random Generation of Industrial Pipelines' Data Structure using a Markov Chain Model

4.1 Introduction

Pipelines and gas projects have increased rapidly throughout the world to meet energy requirements. In 2016, Pipelines and Gas Journal [69] reported that 33% of pipeline projects are in North America. Thus research in this field increased accordingly. Research in this field is classified in accordance with the components of industrial projects. For example, industrial projects may be composed of two major components: the construction of pipeline facilities and the construction of pipelines connecting two pipeline facilities located in two different locations. The complexity associated with these components differs in terms of the required resources, construction methods, materials supply chain, etc. To study the integration effects of all factors, different modeling approaches have been employed. There are four phases to building a simulation model: (1) product abstraction, (2) process abstraction, (3) modeling, and (4) experimentation [70]. The first three phases involve two major processes, namely systems' knowledge acquisition and data collection. Data collection plays a critical role in simulation modeling, especially in cases where the numerical data required in building the simulation models may not be readily available [71]. Ten to 40% of total time in building simulation models is usually devoted to data collection, data preparation, and validation [72]. The collected data is used in input modeling of the simulated operation because it inherits the randomness associated with the system properties. It is valuable in modeling and in conducting experimental studies for the purpose of understanding the systems' behavior under different circumstances.

Input modeling for simulation purposes can be viewed as the practice of selecting a probability distribution that best represents randomness in input sources [73]. Input modeling is performed by fitting collected data to theoretical probability distributions using the assessment of goodness of fit as a metric for quality. The case where a single event is modeled is called a univariate model. The generated sample from the distribution is a single numerical value representing a single possible event of a specific process. When randomly generating events from a probability distribution function, the collection of these events represents the approximated randomness property of the modeled process. Generally speaking, the univariate model is used to randomly generate independent variables and has been widely applied in the simulation of pipeline projects. For example, each process involved in pipeline construction is represented by a probability distribution function; Tommelein [74] assigned uniform and triangular distribution functions to welding and trenching processes respectively [75]. The probability distribution functions are also applied in simulation modeling of industrial fabrication processes. For example, in modeling pipe spool processes, beta and normal distribution functions were assigned to the fabrication and transportation processes respectively [74]. These examples, as mentioned previously, represent independent events. However, different events/variables in the simulation model of a complex problem may be dependent on each other. Randomly generating a correlated variable is imperative to construct a realistic simulation behavior. Moreover, failure to capture such dependencies may lead to inaccurate input models which eventually results in generating errors in the performance estimates [76] [77]. A multivariate distribution approach has been proposed to preserve dependencies between input variables. Multivariate distributions rely on joint distribution functions to preserve the correlation between randomly generated variables. Each variable is represented by a distribution function called marginal distribution. The correlation between the correlated variables is maintained using the covariance matrix [78]. Such an approach is widely implemented in construction, especially in modeling the total cost of construction projects. The total project cost is considered a vector of correlated variables, each representing the cost of a certain construction component or work package. A joint distribution function is used to randomly generate total project cost vectors [79] [80] [81].

The above description of the random generation of inputs is confined to a numerical type of data. However, in construction engineering research, the complexity of data extends beyond a numerical type of data; it includes a combinatorial data type. The complexity associated with such data types is that the randomly generated variables are not single numerical values or jointly correlated values. Rather, they represent a structure such as graphs or trees. Such cases are present in modeling the construction of industrial projects, specifically, pipe spools fabrication. The nature of industrial spool fabrication in construction is, to some extent, similar to industrial manufacturing. However, it is characterized as a low volume and high product mix production process [82]. As a result, the product routing and the time required for its fabrication may vary widely according to the product features as well as its complexity [83]. Consequently, randomly generating inputs using a probability distribution describing the fabrication time without considering the randomness and complexity associated with the product itself, may result in the improper modeling of fabrication processes [82]. Hence, randomly generating combinatorial data of a product along with its processing time is expected to improve the simulation model accuracy and provide additional flexibility for testing the efficiency of different models under different scenarios.

In modeling a spool fabrication process, spool pipe is defined as a collection of sequenced components with unique attributes such as type, size, and material [11]. Pipe spools, on the other hand, are parts of a high-level product, a pipeline, in industrial construction projects [84]. This study looks at modeling pipelines to randomly generate products of a combinatorial type of data to model construction processes of industrial construction. A pipeline data structure can be represented as a tree structure composed of nodes that represent the pipeline component and edges that represent connectivity between pipeline components. The random generation of trees has been widely covered in literature. Drmota [85] described different classes of random trees including combinatorial, recursive, and search trees. A random generation of a set of combinatorial trees represents a subclass of a structurally defined class of finite trees such as binary trees. The generation of recursive trees is based on randomly generating the number of children (or branches) and then recursively generating the branches of the branches and so on. The major differences between these trees is that a recursive tree does not necessary follow a certain class of trees; it can take the shape of unary-binary tree (a unary-binary tree is a tree whose inner nodes may have a single child or two children [86], this generally known as a k-array tree in general). Search trees are more devoted toward storing and searching data in computer science applications. When projecting definitions of these types of random trees into the random generation of industrial pipeline data structures, the most relevant class of random tree is the recursive tree.

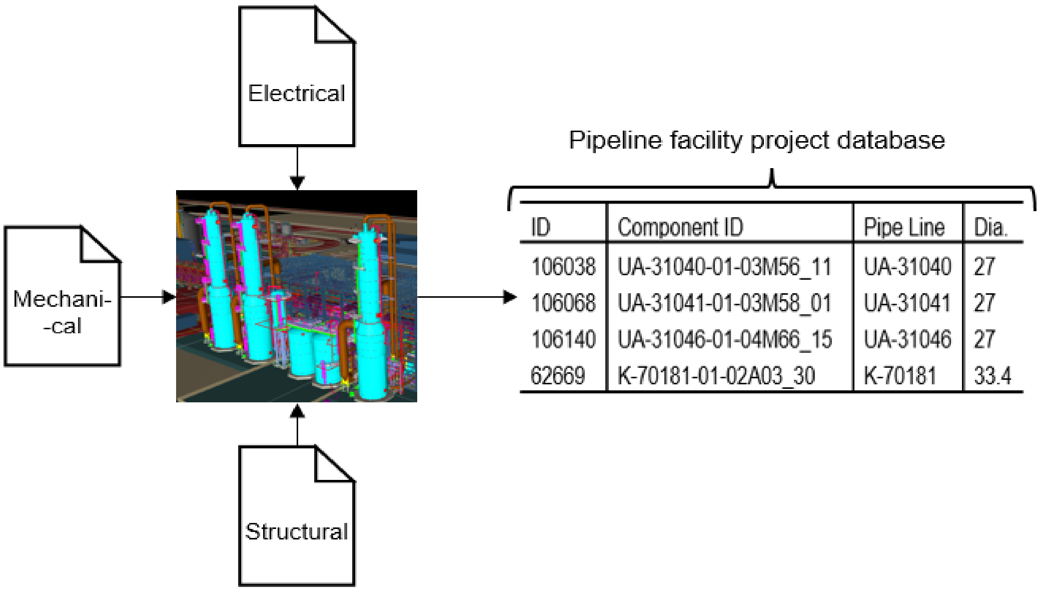
An industrial pipeline, as described previously, is composed of a collection of different types of components. Only a certain type of component has the ability to generate two branches such a tree connection component. Furthermore, the reproduction of each type of pipeline component may depend on the pre-generated component. For this reason, this chapter proposes applying a branching process in terms of a Markov chain generation model to randomly generate industrial pipeline data structure in the form of a recursive tree. This chapter is organized as follows:

1. In Section 4.2, an overview of industrial pipeline data is presented. This includes pipeline data preparation and structuring.

2. In Section 4.3, a statistical analysis of pipeline data is presented. This is performed to gain insight and knowledge about pipeline components, their properties, and correlations.
3. In Section 4.4, the construction of process of the Markov-chain pipelines generation model is presented.
4. In Section 4.5, a comprehensive validation process is conducted. The process starts by converting the topological structure of the pipelines into a feature vector capable of preserving the structural properties. After this, the three-stage validation process is applied.

4.2 Overview of pipeline data

A dataset from an existing industrial construction project located in Alberta, Canada is used in this study. Referring to Figure 4- 1, industrial construction projects are presented in terms of a building information model (BIM) that contains layers showing different project properties for different disciplines such as electrical, mechanical, and structural. The project design data flow starts by combining drawings of and information about different parts of the project from different engineering divisions into a single BIM model. Then, all of the information about project components is transferred to a database so that it can be used in both construction and research activities.



ID	Component ID	Pipe Line	Dia.
106038	UA-31040-01-03M56_11	UA-31040	27
106068	UA-31041-01-03M58_01	UA-31041	27
106140	UA-31046-01-04M66_15	UA-31046	27
62669	K-70181-01-02A03_30	K-70181	33.4

Figure 4- 1 Design data flow of pipeline facility project

The industrial construction project database includes the following component properties:

1. Unique component id number,
2. Pipeline number to which the component belongs
3. Component property that describes the type of component (e.g., *Tube, Valve, Flange,* etc.)
4. Location (minimum (x,y,z), maximum (x,y,z)) of the component in the BIM model
5. Components' diameter
6. Components' length

This information is valuable to study the topology of pipelines (e.g., components' sequential patterns and their properties). However, when exploring the type of components that form a pipeline, it was found that there are many pipelines that have types of components that are not primary components. In this study, pipelines' primary components are defined as a sequence of components attached to one another (e.g., tube, tee, and elbow as shown in Figure 4- 2) for the purpose of directing the flow of fluids such as oil or liquidated gas. In piping systems, the collection of these components is called a pipeline section [84]. Due to the considerations of project design functionality, these pipeline sections have unique configurations when assembled. Therefore, secondary components (e.g. supports, shown in Figure 4- 2) are attached to pipelines to satisfy the design functionality. When pipelines were filtered based on primary components, the result was that the types of components were reduced to 13, and they are listed as follow;

1. Tube*:* a pipeline element that hosts the material flow.
2. Elbow: a pipeline element that changes the flow direction.
3. Flange: a pipeline element that connects two pipeline sections without permanently joining them.
4. Tee: a pipeline element that connects a perpendicular branch.
5. Valve: a pipeline element that regulates the material flow.
6. Fblind/ blind flange: a disk-shaped pipeline component used to block off a pipeline.
7. Ftube/ blind tube: a cylindrically shaped pipeline component used to block off a pipeline.
8. Reducer: a pipeline element that changes the pipeline flow diameter.
9. Closure: a pipeline element that seals the end of the pipe.
10. Cap: a pipeline element that seals the end of the pipe.

11. Instrument: a pipeline element that measures a certain pipeline property such as pressure.
12. Pcomponent: a general pipeline component.
13. Coupling*:* a pipeline element that connects two threaded pipes of the same size.

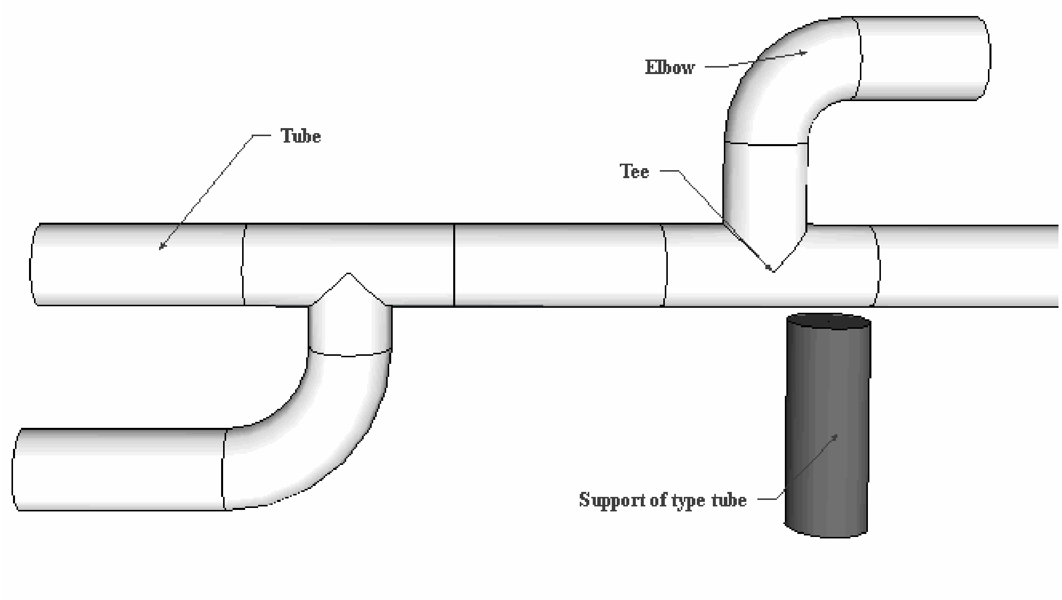


Figure 4- 2 Section of industrial pipeline

Studying the properties of these components provides insight into industrial pipeline structure (i.e., components' formation and their physical properties). However, it does not reflect the unique sequence of the pipeline components; two pipelines may contain the same number of each type of components, but they may differ regarding their sequence. Furthermore, studying component sequencing in pipeline structures is considered the base of the random generation study of pipeline data structures. Therefore, it is important to arrange and prepare a pipeline data table to support the analysis of both pipeline component properties and sequences. A recursive function is employed to extract and generate a data table with components ordered according to their sequence in the pipeline structure.

The recursive function uses two tables from the database to branch and sequence pipeline components. The first table contains information about pipeline component properties such as lengths and diameters and the second table provides information about component connectivity. Figure 4- 3 illustrates the branching process used in the recursive function. The recursive function starts by first randomly selecting a component with one connectivity from the first pipeline set of components (component/node 1 in Figure 4- 3). The next step is to append the first component in the first branch list. Then, using the connectivity table in the database, the second connectivity point (component/node 2 in Figure 4- 3) is determined and appended to the same branch list. In the case of a component branching the pipeline into two branches, such as component/node 2, one of the components will be appended in the first branch and the second will be appended in the second branch. Once all the components are processed, the branches are ordered in the pipeline data table based on the branch number. In the case of a branch having sub-branches, each branch extending from the first branch connection is treated as a block containing all branches ordered according to their position in the pipeline structure. This process is applied on 1052 pipelines with a total number of components equal to 33324. A sample of the final pipeline data table used in this study is shown in Table 4- 1.

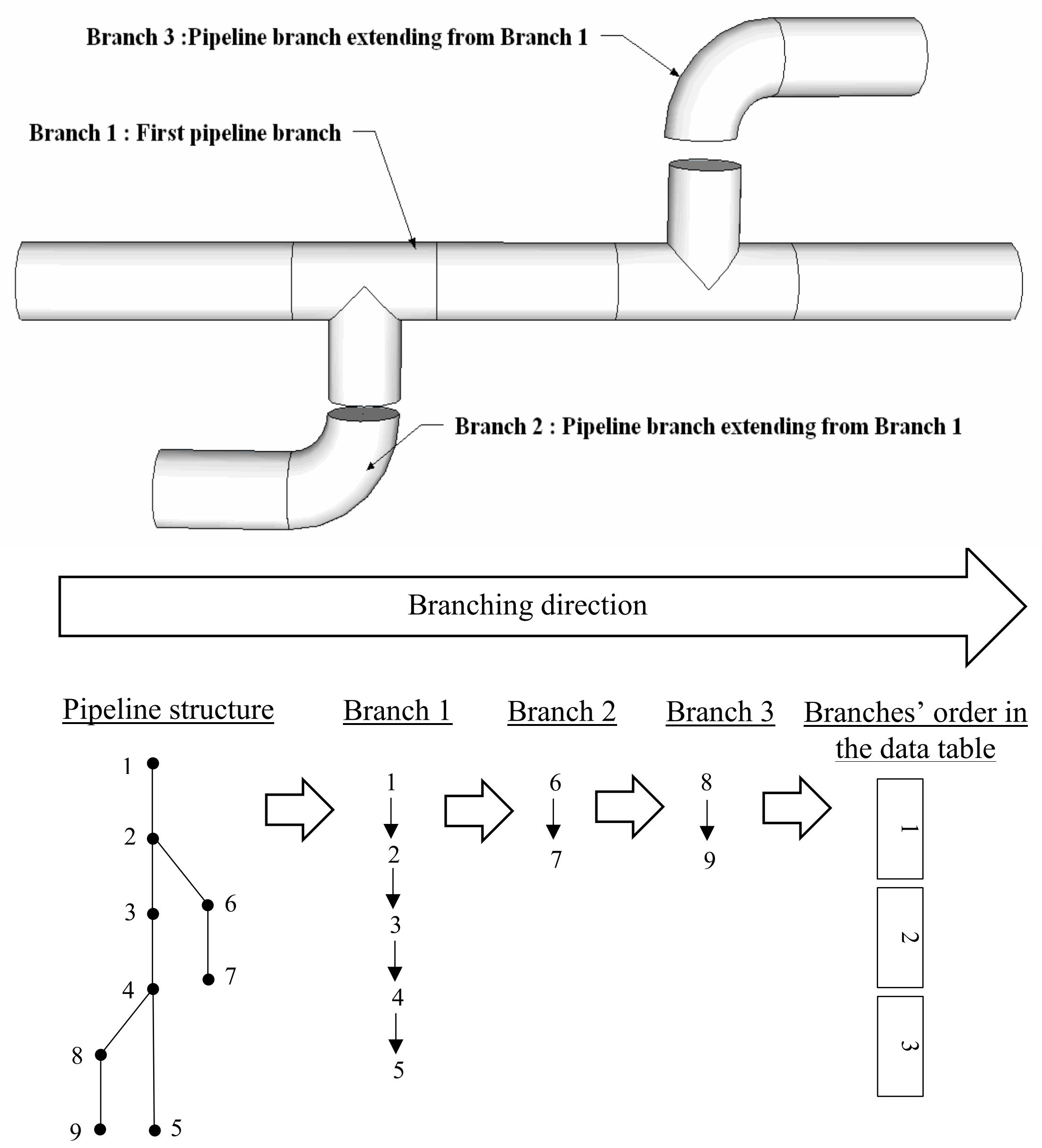

Branch 3 :Pipeline branch extending from Branch 1
Branch 1 : First pipeline branch
Branch 2 : Pipeline branch extending from Branch 1
Branching direction
Pipeline structure
Branch 1
Branch 2
Branch 3
Branches' order in the data table
1
2
3
4
5
6
7
8
9
1
2
3
4
5
6
7
8
9
1
2
3

Figure 4- 3 Pipeline branching process

Table 4- 1 Pipeline data table

Line No.	Branch No.	Seq.	Component ID	Type	Dia.	Length
5	7	1	SB-XXXXX-01A03_33	Elbow	72	106
5	7	2	SB- XXXXX -01A03_34	Tube	60	100
5	7	3	SB- XXXXX -01A03_35	Flange	213	0
5	7	4	SB- XXXXX -01A03_37	Valve	213	0
5	7	5	SB- XXXXX -01A03_39	Fblind	20	213
5	8	1	SB- XXXXX -01A03_48	Elbow	72	106
5	8	2	SB- XXXXX -01A03_49	Tube	60	100
5	8	3	SB- XXXXX -01A03_50	Flange	213	0
5	8	4	SB- XXXXX -01A03_52	Valve	213	0
5	8	5	SB- XXXXX -01A03_54	Fblind	20	213

4.3 Statistical Data Analysis

In this section, a basic statistical analysis is performed to provide a better understanding about pipeline data before structuring the pipeline generator model. Figure 4- 4 to Figure 4- 6 demonstrate the percentage of each component in the entire pipelines' population, in the first pipeline branch (based on the branching process described in Section 5.2 and equivalent to "Branch 1" shown in Figure 4- 3), and the other branches extending from the first pipeline branch respectively (equivalent to "Branch 2" and "Branch 3" shown in Figure 4- 3). As shown in Figure 4- 4, the dominant components of the population are tube, elbow, flange, tee, and valve, which account for 85.9% of all components in the pipeline population. This percentage increases to 91.4% if the analysis is restricted to the first branch and goes down to

78.9% for other branches extending from the first one, see Figure 4- 5 and Figure 4-6.

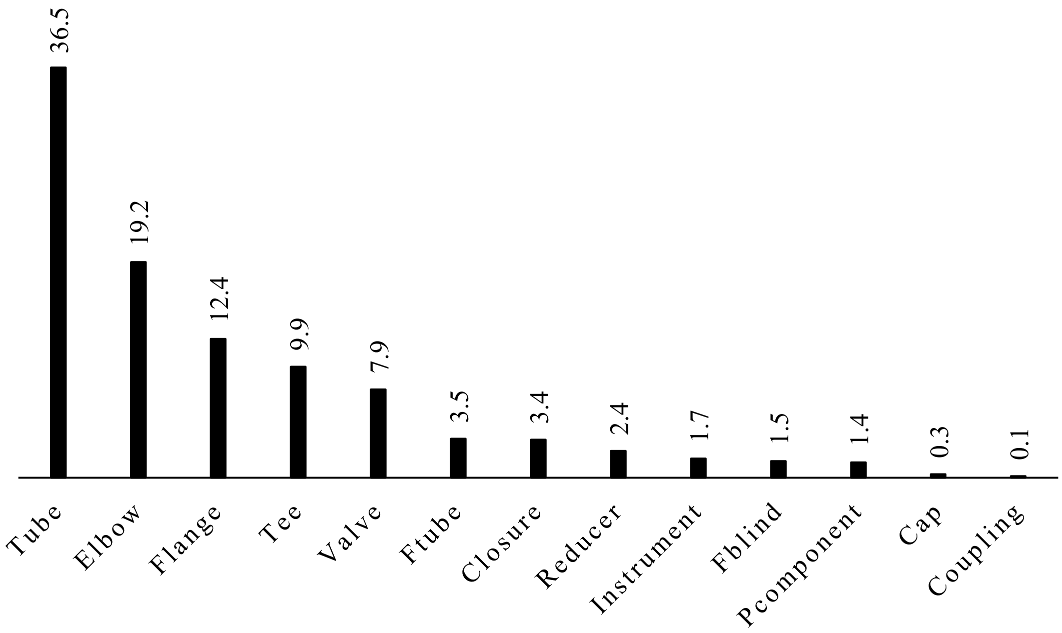


Figure 4- 4 Percentage of each component in the entire population

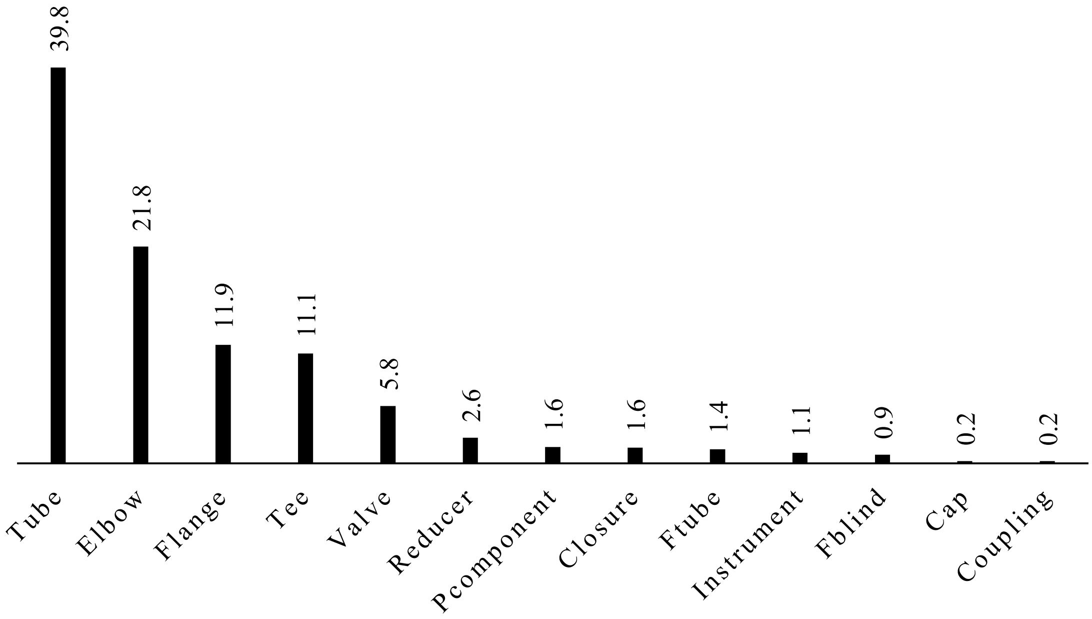


Figure 4- 5 Percentage of each component in the first pipeline branch

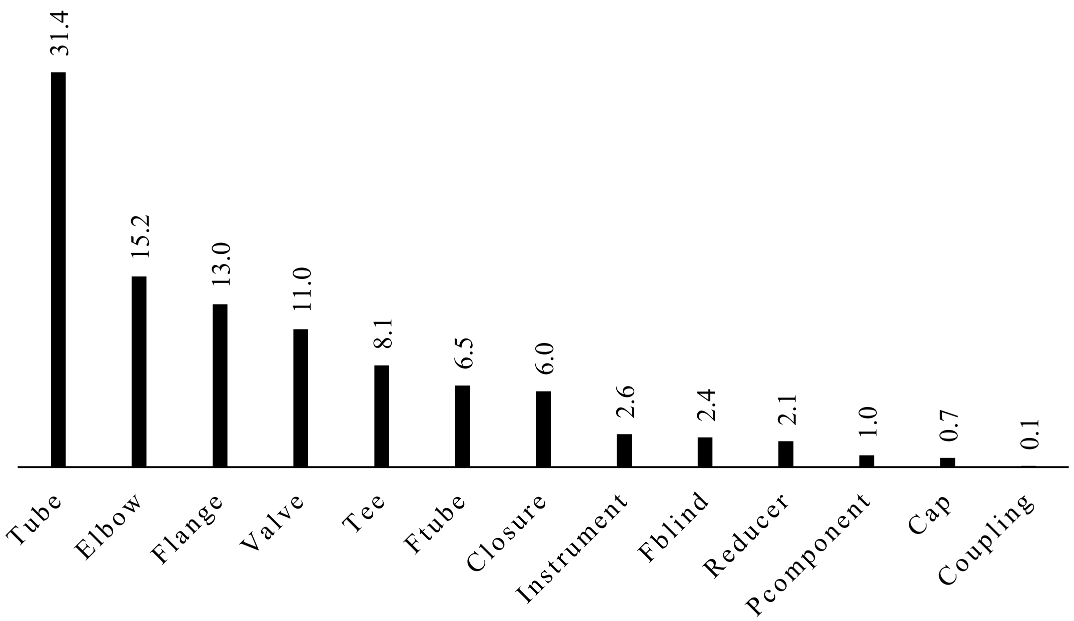


Figure 4- 6 Percentage of each component in branches extending from the first branch

The second in rank comes to components ftube and closure which account for 6.9% of the entire population. These two components are more condensed in the population representing branches extending from the first pipeline branch. This can be related to the main function of these components, which is sealing the pipeline flow. Component instrument shows the same trend as in ftube and closure, and component pcomponent demonstrates the opposite trend which shows that it is mostly condensed in the first pipeline branch population. Components coupling and cap, on the other hand, have the lowest contribution in the pipeline population, accounting for less than 0.8 %. This can be attributed to the design requirements which normally allow coupling to be installed in low-pressure pipes. Likewise, the cap is used to seal low-pressure pipes using a threaded connection. Unlike other pipeline components, the occurrence of component reducer is almost consistent in Figure 4- 4, Figure 4- 5, and Figure 4- 6, meaning that its occurrence is highly controlled by its design objective, which is to change the pipeline flow diameter.

Figure 4- 7 to Figure 4- 9 show the distribution of the number of components in the entire pipeline population, in the first pipeline branch and the other branches extending from the first pipeline branch, respectively. The majority of the number of pipeline components is less than 50 components with a high proportion devoted toward to the first pipeline branch that account for 60% (see Table 4- 1) of the entire population. The branches extending from the first pipeline branch are skewed toward a number of components less than 10 and representing 40% (see Table 4- 2) of the entire population.

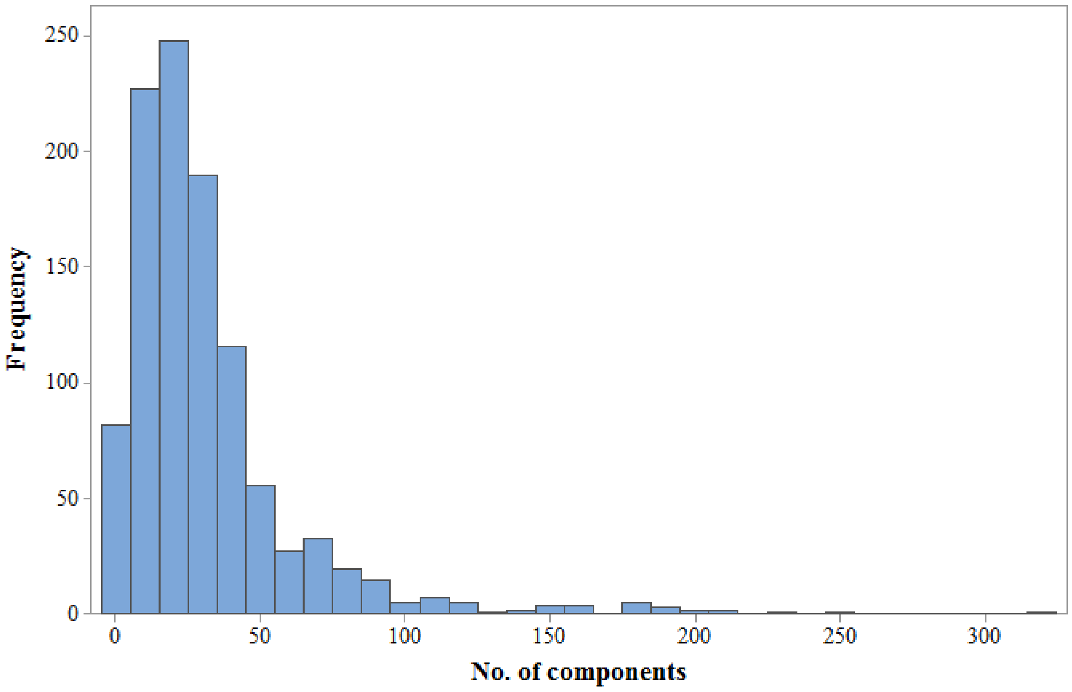


Figure 4- 7 Distribution of the number of components in each pipeline

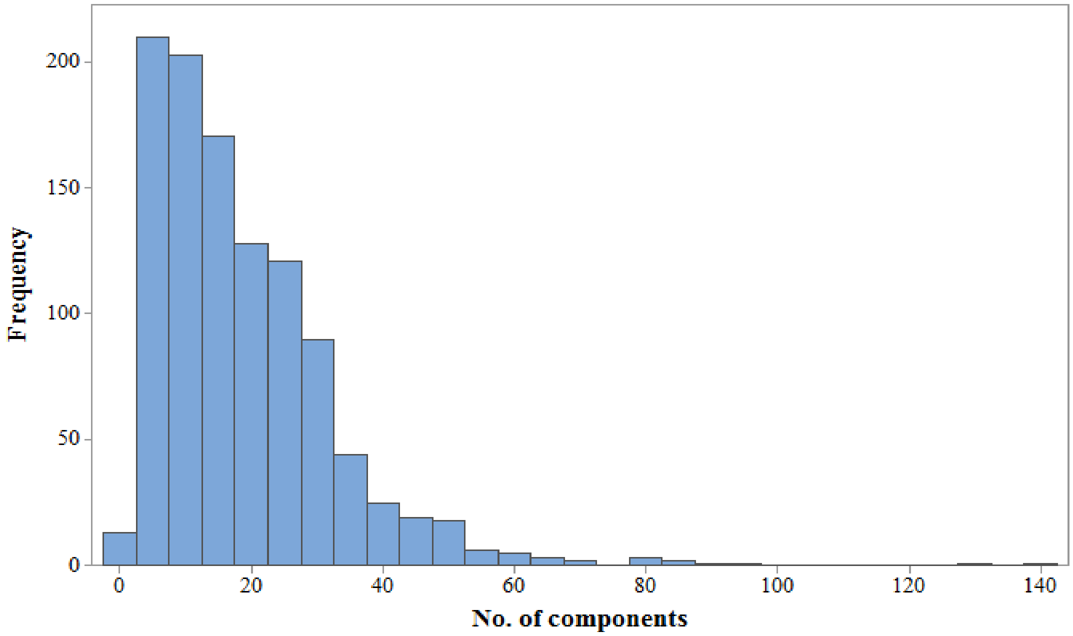


Figure 4- 8 Distribution of the number of components in the first pipeline branch

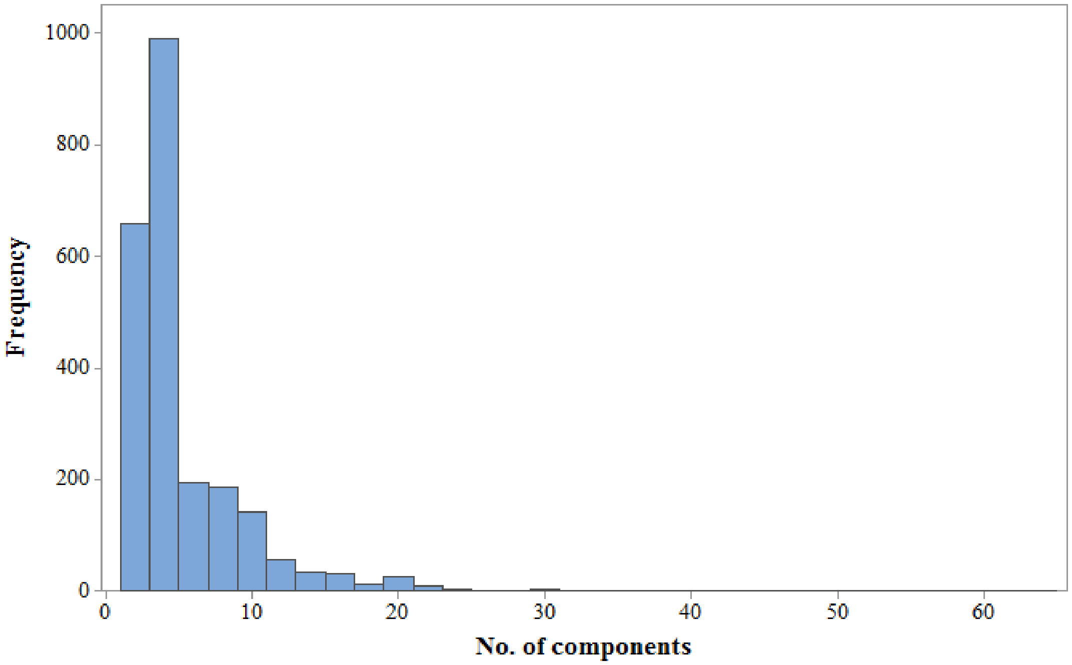


Figure 4- 9 Distribution of the number of components in branches extending from the first pipeline branch

The results in Table 4- 2 show that the number of pipelines which have branches extending from the first pipeline branch is 760 pipelines. This result does not necessarily mean that the remaining pipelines have no component of type tee; however, when investigating the pipeline database, it was found that some of the pipelines have a tee connection for the purpose of connecting to different pipelines, which means that two pipelines with different properties are designed to flow in parallel and connect to a branch that extend from one of them.

Table 4- 2 Descriptive statistical measures for the number of components

Statistic	**Total no. of components**	**No. of components in the first pipeline branch**	**No. of components in branches extending from the first pipeline branch**
Sample Size/ Components	33324	19943	13423
Mean	32	19	18
Variance	1070	213.2	817.75
Std. Deviation	32.711	14.601	28.596
No. of occurrences in pipelines	1052	1052	760

From the above analysis it is concluded that *different branches located in the same pipeline may have different characteristics.* This adds more complexity to the generation process since the degree of correlations and dependencies between pipeline components may differ according to the location of components in the pipeline structure (i.e., whether it is located in the first pipeline branch or in other branches). This conclusion can be verified by calculating the symmetric correlation coefficients' matrices on both types of branches. These matrices are illustrated in (4-1) and (4-2).

The first matrix illustrates the correlation coefficients of pipeline components in the first pipeline branch, and the second matrix illustrates the correlation coefficients of pipeline components in branches extending from the first branch. In the first matrix, all components show positive relationships except to components closure, cap, ftube, and fblind. However, positive relationships among all pipeline components are shown in the second matrix. Also, a higher degree of correlation coefficients between components is generated in the second matrix.

$$\begin{array}{c}\text{Pcomponent}\\ \text{Instrument}\\ \text{Valve}\\ \text{Flange}\\ \text{Tube}\\ \text{Elbow}\\ \text{Tee}\\ \text{Reducer}\\ \text{Coupling}\\ \text{Closure}\\ \text{Cap}\\ \text{Ftube}\\ \text{Fblind}\end{array}\left[\begin{array}{ccccccccccccc}
1 & 0.12 & 0.18 & 0.38 & 0.11 & 0.11 & 0.09 & 0.31 & 0.14 & -0.31 & -0.08 & -0.17 & -0.17\\
 & 1 & 0.38 & 0.37 & 0.24 & 0.12 & 0.22 & 0.5 & 0.28 & -0.13 & -0.07 & 0.05 & -0.09\\
 & & 1 & 0.57 & 0.22 & 0.13 & 0.31 & 0.29 & 0.18 & 0.08 & -0.08 & 0.41 & 0.06\\
 & & & 1 & 0.37 & 0.29 & 0.44 & 0.35 & 0.03 & -0.18 & -0.09 & 0 & -0.01\\
 & & & & 1 & 0.82 & 0.67 & 0.32 & 0.15 & -0.03 & 0.11 & -0.07 & -0.03\\
 & & & & & 1 & 0.36 & 0.27 & 0.13 & -0.08 & 0.03 & -0.09 & -0.14\\
 & & & & & & 1 & 0.24 & -0.01 & 0.08 & 0.24 & 0.06 & 0.12\\
 & & & & & & & 1 & 0.29 & -0.12 & -0.05 & -0.03 & -0.15\\
 & & & & & & & & 1 & -0.05 & 0 & 0 & -0.05\\
 & & & & & & & & & 1 & -0.04 & 0.45 & -0.17\\
 & & & & & & & & & & 1 & 0.11 & 0.02\\
 & & & & & & & & & & & 1 & -0.12\\
 & & & & & & & & & & & & 1
\end{array}\right] \quad 4-1$$

$$\begin{array}{c}\text{Pcomponent}\\ \text{Instrument}\\ \text{Valve}\\ \text{Flange}\\ \text{Tube}\\ \text{Elbow}\\ \text{Tee}\\ \text{Reducer}\\ \text{Coupling}\\ \text{Closure}\\ \text{Cap}\\ \text{Ftube}\\ \text{Fblind}\end{array}\left[\begin{array}{ccccccccccccc}
1 & 0.37 & 0.41 & 0.5 & 0.32 & 0.28 & 0.41 & 0.31 & 0.03 & 0.17 & 0.21 & 0.21 & 0.33\\
 & 1 & 0.62 & 0.64 & 0.38 & 0.32 & 0.46 & 0.33 & 0.09 & 0.38 & 0.27 & 0.46 & 0.46\\
 & & 1 & 0.7 & 0.58 & 0.55 & 0.64 & 0.46 & 0.07 & 0.69 & 0.32 & 0.81 & 0.59\\
 & & & 1 & 0.57 & 0.53 & 0.64 & 0.41 & 0.05 & 0.4 & 0.47 & 0.46 & 0.53\\
 & & & & 1 & 0.95 & 0.9 & 0.5 & 0.09 & 0.39 & 0.3 & 0.36 & 0.39\\
 & & & & & 1 & 0.81 & 0.46 & 0.09 & 0.38 & 0.26 & 0.34 & 0.34\\
 & & & & & & 1 & 0.48 & 0.05 & 0.44 & 0.4 & 0.44 & 0.48\\
 & & & & & & & 1 & 0.11 & 0.17 & 0.2 & 0.19 & 0.18\\
 & & & & & & & & 1 & 0.05 & 0.12 & 0.05 & 0.02\\
 & & & & & & & & & 1 & 0.12 & 0.05 & 0.02\\
 & & & & & & & & & & 1 & 0.35 & 0.22\\
 & & & & & & & & & & & 1 & 0.27\\
 & & & & & & & & & & & & 1
\end{array}\right] \quad 4-2$$

4.4 Markov chain pipeline generation model

Referring to Figure 4- 3 in section 4.2, the pipeline data structure can be presented in terms of a tree structure containing nodes that represent the type of component (e.g., tube, elbow, flange, tee, etc.), and edges that represent the connectivity between two nodes. The branching of a pipeline tree structure is controlled by the component of type tee. The number of occurrences of component type tee and its location in pipelines differs from one pipeline to another, which results in a unique pipeline topological structure. The occurrence of type tee components also depends on other pipeline components such as a component of type tube. This case is applied to all pipeline components. Incorporating such dependencies (correlation between pipeline components and the connectivity relationships among them) in the pipeline generation model is important to create a realistic sequential pattern for pipeline components. Thus, a Markov chain model is proposed to generate a realistic random sequence of pipeline components.

A Markov chain is described as a sequence of random variables or events constructed using Markov property that defines what happens next according to the current state of the system [87]. It describes a system that follows a linked chain of different states. Its mathematical statement is described as follows:

$$P(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = P(X_{n+1} = x \mid X_n = x_n) \qquad 4\text{-}3$$

The mathematical statement of the Markov chain model can be described as giving a set of random variables or states X_i. The Markov chain's sequence of states is dependent on the present state X_n at step n and the selection of the next step $n+1$ is conditioned by the probability value of the current state $P(X_n = x_n)$.

To apply a Markov chain model, first the system and its states should be defined. The pipeline is considered as a system represented by a chain of 13 dependent states which are pcomponent, instrument, valve, tube, elbow, tee, coupling, cap, reducer, flange, ftube, fblind, and closure. The transition from one state to another in the pipeline system is conditioned by the probabilities of a collection of states which may follow the present state. These probabilities are called transition probabilities. They regulate the movement between pipeline states. These probabilities are combined in the form of a matrix called a transition matrix shown in (4-4).

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{bmatrix} \quad 4\text{-}4$$

The Markov chain transition matrix is a $n \times n$ matrix containing transition probabilities such as p_{ii} that represent the probability of state i returning to itself and p_{ij} representing the probability of state i moving to state j. Since the database of the pipelines previously described is prepared and organized to reflect the connectivity between different components, it is possible to generate the transition matrix of pipeline states. However, based on the statistical analysis performed in section 4.3**,** which showed that the degree of dependencies between pipeline components is controlled by their locations in the pipelines' structure (whether in the first branch or in branches extending from the first branch), the pipeline system is split into two sub-systems, each with its own transition matrix. The transition matrix (4-5) below refers to the first pipeline branch and the transition matrix (4-6) refers to branches extending from the first pipeline branch. The order of the pipeline states shown in each row in the transition matrix is the same in the matrix columns. The values within each matrix are in the form of a percentage.

$$\begin{array}{c} \text{Pcomponent} \\ \text{Instrument} \\ \text{Valve} \\ \text{Flange} \\ \text{Tube} \\ \text{Elbow} \\ \text{Tee} \\ \text{Reducer} \\ \text{Coupling} \\ \text{Closure} \\ \text{Cap} \\ \text{Ftube} \\ \text{Fblind} \end{array} \begin{bmatrix} 0 & 0 & 22.8 & 53 & 3 & 0 & 0.7 & 18.8 & 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 12.3 & 65.9 & 5.5 & 0 & 7.7 & 4.1 & 0 & 0 & 4.0 & 0.5 & 0 \\ 1.1 & 1 & 8 & 29.3 & 14.3 & 0.1 & 6.8 & 8.3 & 2.7 & 3.5 & 6.7 & 13 & 5.1 \\ 4.2 & 4.9 & 19 & 4.2 & 23.9 & 5.4 & 12.8 & 13.4 & 0.2 & 0.2 & 11.1 & 0.4 & 0.4 \\ 14.8 & 14.9 & 3.1 & 17.5 & 5.3 & 29.7 & 8.7 & 3.3 & 0.1 & 0 & 2.7 & 0.01 & 0 \\ 0.3 & 0.3 & 16.3 & 0.9 & 32.4 & 0.05 & 16.3 & 16.4 & 0.1 & 0.1 & 16.2 & 0.2 & 0.1 \\ 0.2 & 0.3 & 14.5 & 0.8 & 32.0 & 0.4 & 15.3 & 14.8 & 1.7 & 1.8 & 14.6 & 1.9 & 1.7 \\ 2.8 & 2.7 & 14 & 13.5 & 25.8 & 5.6 & 12.6 & 11.7 & 0.3 & 0 & 11.1 & 0.1 & 0 \\ 0 & 0 & 16.3 & 0 & 19.9 & 0 & 16.3 & 30.8 & 0 & 0 & 16.3 & 0.5 & 0 \\ 0 & 0 & 56.4 & 0 & 0 & 0 & 43.6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 11.4 & 61.2 & 7.2 & 0 & 7.2 & 3.8 & 1.3 & 1.3 & 3.8 & 1.7 & 1.3 \\ 5.7 & 5.7 & 14.7 & 8.2 & 0.8 & 12.3 & 49.9 & 0.7 & 0 & 0 & 1.5 & 0 & 0.5 \\ 2.8 & 0 & 54.1 & 33.9 & 0 & 0 & 0 & 0 & 1.8 & 1.9 & 0 & 3.7 & 1.8 \end{bmatrix} \tag{4-5}$$

$$\begin{array}{c} \text{Pcomponent} \\ \text{Instrument} \\ \text{Valve} \\ \text{Flange} \\ \text{Tube} \\ \text{Elbow} \\ \text{Tee} \\ \text{Reducer} \\ \text{Coupling} \\ \text{Closure} \\ \text{Cap} \\ \text{Ftube} \\ \text{Fblind} \end{array} \begin{bmatrix} 0 & 1.6 & 20.6 & 58.7 & 6.3 & 0 & 3.2 & 3.2 & 0 & 0 & 3.2 & 0 & 3.2 \\ 1.2 & 0 & 2.4 & 77.4 & 4.8 & 0 & 9.5 & 2.4 & 0 & 0 & 2..4 & 0 & 0 \\ 2.4 & 8.5 & 2 & 15.4 & 3.4 & 0.2 & 1..7 & 2.1 & 3 & 35.8 & 1.7 & 5.9 & 17.9 \\ 4 & 6.7 & 26.3 & 5.4 & 17.1 & 3.8 & 9.4 & 9.8 & 1.4 & 1.4 & 8.6 & 2.8 & 3.4 \\ 13.7 & 13.8 & 3.3 & 18.3 & 5.9 & 27.4 & 10.4 & 4 & 0 & 0 & 3 & 0 & 0 \\ 0.3 & 0.2 & 15.9 & 1.7 & 31.9 & 0.5 & 16 & 16.1 & 0.3 & 0.3 & 15.9 & 0.6 & 0.3 \\ 0.2 & 0.2 & 15.4 & 1.1 & 30.7 & 0.4 & 15.8 & 15.7 & 0.9 & 1.2 & 15.6 & 1.9 & 0.9 \\ 2.8 & 1.7 & 17.3 & 15.2 & 22.7 & 3.3 & 12.3 & 12.7 & 0.8 & 0 & 11.3 & 0 & 0 \\ 0 & 3.4 & 10.2 & 0 & 20.3 & 0 & 10.2 & 11.9 & 6.8 & 6.8 & 10.2 & 13.6 & 6.8 \\ 0 & 0 & 16.7 & 0 & 0 & 0 & 0 & 0 & 16.7 & 16.7 & 0 & 33.3 & 16.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 18.8 & 0 & 37.5 & 18.8 \\ 3.1 & 3.2 & 76.2 & 3.5 & 0.2 & 6 & 0.4 & 0.1 & 0 & 0.4 & 6.8 & 0 & 0 \\ 0 & 0 & 0 & 3.8 & 0 & 0 & 0 & 0 & 19.2 & 19.2 & 0 & 38.5 & 19.2 \end{bmatrix} \tag{4-6}$$

Most states in pipeline properties may remain the same since $p_{i,i} \neq 0$ and the majority of states in different pipelines return to the original state in multiple steps except for state *"Tube,"* which is considered a free-floating component with no restriction on the periodicity. Therefore, to regulate the occurrence of pipeline states in the pipeline structure, a Markov chain property called state periodicity is integrated into the generation model. A state is called periodic if it has the ability to be repeated in multiple steps larger than one. Moreover, since the number of pipeline components varies according to the type of pipeline branch, it is expected that state periods may show different behavior, which adds complexity to the generation process. Therefore, similar to the Markov chain transition matrices, periods of pipeline states are calculated both for states representing the first pipeline branch and states of branches extending from the first pipeline branch. These periods are fitted to theoretical probability distribution functions that best approximate their behaviors.

Based on the definitions of types of pipeline components in section 5.2**,** periods of pipelines state can be reduced to seven instead of 13. Three factors contribute to this: first, components of the *"Closure," "Cap," "Ftube,"* and *"Fblind"* main functions block off the pipeline product flow. These components have a single connectivity which means that they are located last in the pipeline structure, i.e., they are the absorbing states. Second, the component "*Coupling*" has small probability values; at a certain state in the Markov transition matrix can move toward it. Finally, as mentioned previously, component *"Tube"* is assumed to be a free-floating pipeline component because it is the dominant component in the entire pipeline population. These factors help to reduce the number of state period distribution functions to seven and are illustrated with their associated pipeline state in Table 4- 3. These state period distribution functions are expected when working in conjunction with Markov chain transition matrices to generate pipeline structures that reflect realistic pipeline component sequences.

Table 4- 3 Probability distribution functions for components' state periods

Pipeline state	**Probability distribution function**	
	Located in the first pipeline branch	**Located in branches extending from the first pipeline branch**
"Pcomponent'	Exponential (0.19245)	Exponential (0.31376)
'Instrument"	Gamma (2.086, 7.042)	Gamma (2.4545, 9.1897)
"Valve'	Lognormal (0.795,1.899)	Gamma (1.437,5.584)
'Flange"	Pareto (0.82625,1)	Pareto (0.94902,1)
"Elbow'	Normal (2.7367,3.1097)	Laplace (0.51675,3.1097)
'Tee"	Gamma (1.454, 5.307)	Gamma (0.95639, 5.3605)
"Reducer"	Lognormal (0.809,2.073)	Lognormal (0.8612,2.159)

The pipeline generation flow chart using the Markov chain model is shown in Figure 4- 10 and is split into two phases. The first phase (Phase1) generates the sequences of components in the first pipeline branch. The output from Phase 1 is then used to generate a state sequence that represents the second pipeline (Phase 2), which is a sequences of components in branches extending from the first pipeline branch.

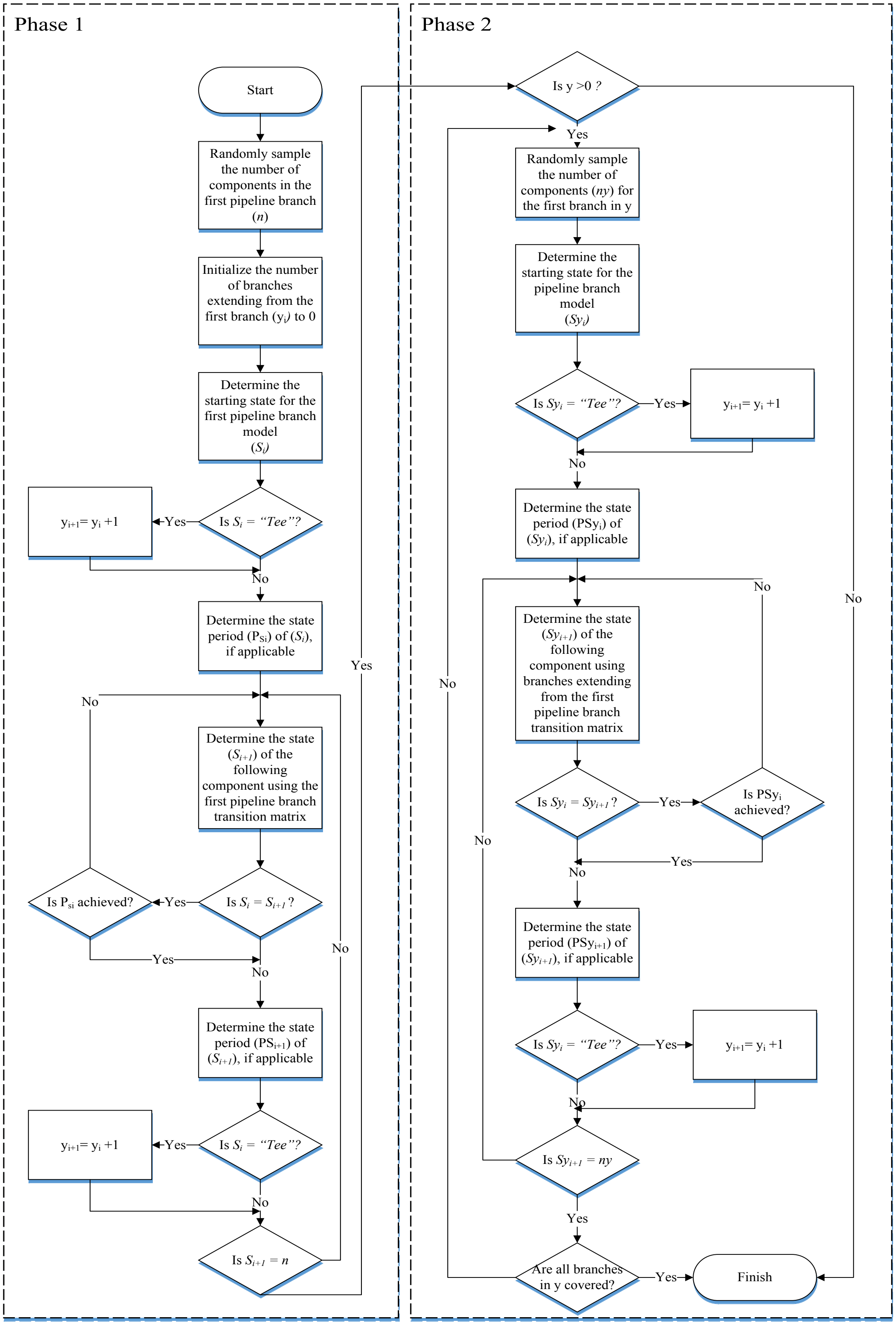

Phase 1
Start
Randomly sample the number of components in the first pipeline branch (n)
Initialize the number of branches extending from the first branch (y_i) to 0
Determine the starting state for the first pipeline branch model (S_i)
Is S_i = "Tee"?
Yes
y_{i+1}= y_i +1
No
Determine the state period (P_{Si}) of (S_i), if applicable
No
Determine the state (S_{i+1}) of the following component using the first pipeline branch transition matrix
Is S_i = S_{i+1} ?
Yes
Is P_{si} achieved?
Yes
No
Determine the state period (PS_{i+1}) of (S_{i+1}), if applicable
Is S_i = "Tee"?
Yes
y_{i+1}= y_i +1
No
Is S_{i+1} = n
No
Yes
Phase 2
Is y >0 ?
Yes
Randomly sample the number of components (ny) for the first branch in y
Determine the starting state for the pipeline branch model (Sy_i)
Is Sy_i = "Tee"?
Yes
y_{i+1}= y_i +1
No
Determine the state period (PSy_i) of (Sy_i), if applicable
No
Determine the state (Sy_{i+1}) of the following component using branches extending from the first pipeline branch transition matrix
Is Sy_i = Sy_{i+1} ?
Yes
Is PSy_i achieved?
Yes
No
Determine the state period (PSy_{i+1}) of (Sy_{i+1}), if applicable
Is Sy_i = "Tee"?
Yes
y_{i+1}= y_i +1
No
Is Sy_{i+1} = ny
No
Yes
Are all branches in y covered?
Yes
Finish
No
No

Figure 4- 10 Flow chart of Markov chain pipeline generation model

It starts by randomly generating the number of components n expected in the first pipeline branch. It is sampled from a gamma distribution function (α = 1.7039, β = 11.023) that is driven by the histogram plot of the number of components shown in Figure 4- 8. The model then initializes the number of branches y expected from the first sub-system to 0 and later changed according to the occurrence of state tee in the first pipeline branch. Thereafter, the starting state S_i of the pipeline component sequence is initialized. The initialization of the starting states is performed by first calculating the probability of states at which a pipeline may starts with. However, an assumption is made in the first and the second pipeline sub-systems which limits the starting states to 7 and 9 respectively. The assumption defines the starting state, shown in Table 4- 4 and Table 4- 5, as a state whose main objective is to host the flow of the material in the pipeline. In the case of a starting state selected as tee, a branch will be added into y, and its state period PS_i will be randomly sampled from its associated probability distribution function shown in Table 4-3. The remaining states of the pipeline components are then generated using the Markov chain transition matrix. In cases in which a Markov chain transition matrix generates two similar states consecutively or apart in multiple steps, a check is conducted. The check is based on whether or not the state period condition is satisfied (i.e., whether the distance between two similar states is equal to their state period). If the condition is satisfied, another state period, PS_{i+1}, is sampled for the current state and the following state is generated using the Markov chain transition matrix shown in (5). On the other hand, if the condition is not satisfied, another state is generated for the current step, and the same process is repeated until a sequence of pipeline components is achieved that is equivalent to the number of components generated by the first pipeline sub-system. Once the first pipeline subsystem (Phase 1) has completed generating the first branch and if the number of branches in y is greater than 0, then the pipeline's second sub-system (Phase 2) will start generating pipeline state sequences for each additional branch; otherwise, the model will end the pipeline generation process. The generation of pipeline branches in Pahse 2 is equivalent to that in Phase 1 with differences related to the used Markov chain transition matrix, state period probability distribution functions, the number and type of states in the initializing step, and the number of components in each branch that is sampled from a Lognormal distribution function (σ = .88575, μ = 1.2818).

Table 4- 4 Starting states in the first pipeline branch (Phase 1)

State	Pcomponent	Instrument	Valve	Flange	Tube	Elbow	Reducer
%	0.96	2.1	10.8	13	40.63	15.37	14.54

Table 4- 5 Starting states in branches extending from the first pipeline branch (Phase 2)

State	Instrument	Valve	Flange	Tube	Elbow	Tee	Reducer	Coupling	Ftube
%	0.2	0.66	16.18	40.16	5.65	1.12	1.66	0.15	34.22

4.5 Validation of Markov chain pipeline generation model

In this section, the performance of the Markov chain pipeline generation model is measured against a population of real pipelines. It is conducted to examine how the integrated Markov chain transition matrix and state period distribution perform when generating a pipeline sequential pattern similar to sequential patterns in the actual population. The validation is performed on components which host the flow of material in the pipeline: pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, and coupling. Two pipeline properties are used in the validation process: (1) the number of each type of component in each pipeline, and (2) the component's location in the pipeline structure. The validation process (Figure 4-11**)** starts by randomly generating pipelines using the Markov-chain pipeline generation model. The number of pipelines generated is equivalent to the original number of pipelines, which is 1052. Then each pipeline is converted to a feature vector capable of capturing pipeline properties. Finally, a three-stage validation process is performed starting with the evaluation of the number of each type of component and correlation analysis, clustering-based validation, and validation by measuring and comparing similarity distances between all feature vectors in the pipeline space.

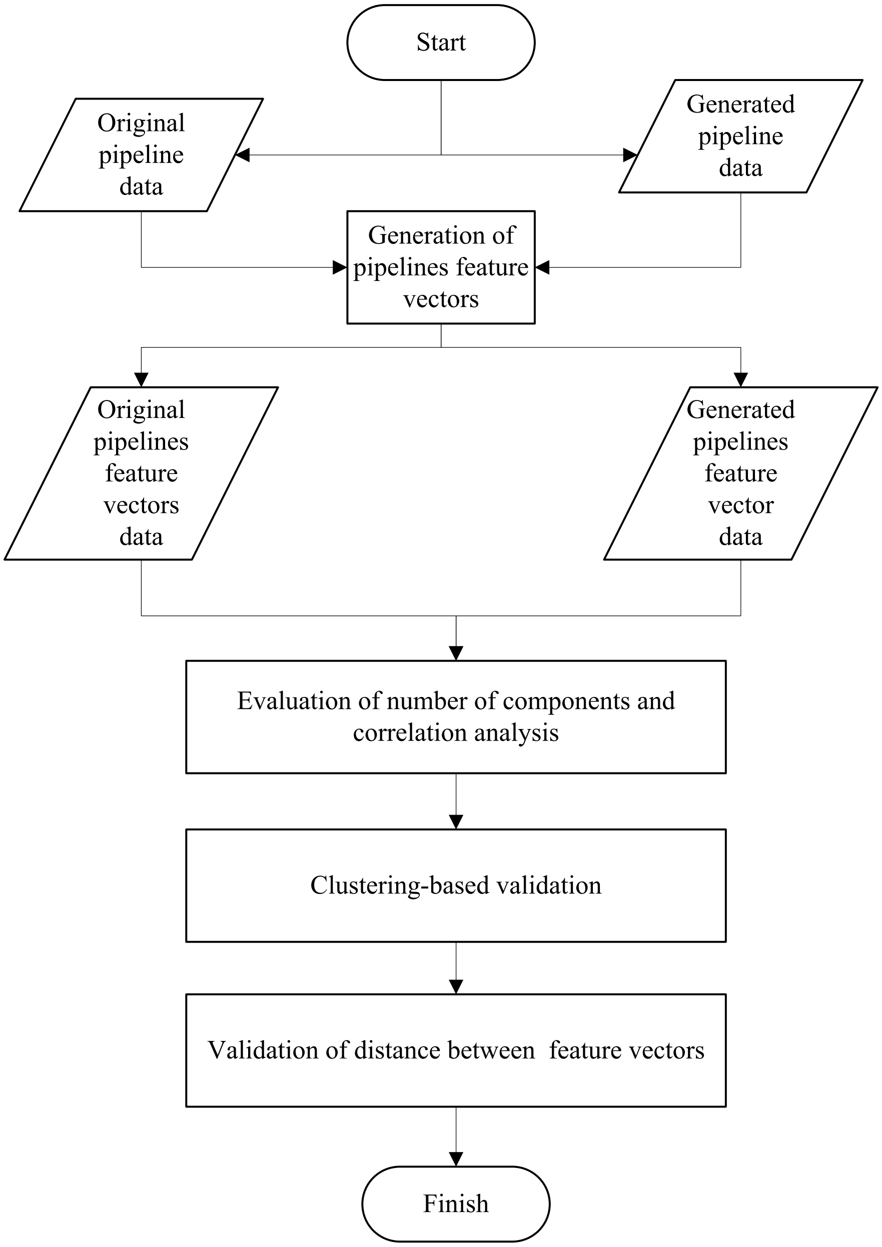

Start
Original pipeline data
Generated pipeline data
Generation of pipelines feature vectors
Original pipelines feature vectors data
Generated pipelines feature vector data
Evaluation of number of components and correlation analysis
Clustering-based validation
Validation of distance between feature vectors
Finish

Figure 4- 11 Validation process flow chart

4.5.1 Pipelines feature vectors

The conversion of the pipeline into a feature vector in this section is performed using a methodology described by Liu et al. [88]. The methodology's main objective was to represent deoxyribonucleic acid (DNA) sequences as a point in the DNA *n*-dimensional space, which represents the number of nucleotides in a DNA sequence, their distance from the origin, and their distribution along the sequence. Similarly, all pipelines are converted to a feature vector representing a point in the pipeline *n*-dimensional space. To achieve consistency in applying this methodology to both the original and generated data, the pipeline generation model was built to represent its outputs in the same way the original data is presented; after applying the pipeline branching process (see Figure 4- 3). Both original and generated pipelines will have the same linearly structured data representation. Therefore, three numerical attributes will be calculated for each pipeline component.

The first numerical attribute is the total number of each type of component in the pipeline and is denoted as n_i; where i = pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, and coupling. It is used to define the pipeline formation components and also dictate the total number of components in the pipeline. Using Figure 4- 12 as an illustration, n_{Tube}, n_{Elbow}, and n_{Tee} in pipelines ***A*** and ***B*** are equal to 5, 2, and 2 respectively. Since nine types of pipeline components are considered in the validation, the other component types which did not occur in pipelines ***A*** and ***B*** will have a value of 0. Both pipelines have the same number of branches and components. However, the distribution of components is not the same. For example, pipeline ***A*** has component tee located in node 2 and 5. In pipeline **B**, they are located in node 2 and 4. The same holds true for component tube: it is located in nodes 1, 3, 4, 7 and 9 in pipeline ***A***, and in nodes 1, 3, 5, 7, and 9 in pipeline **B**. Therefore, a second numerical attribute is proposed to distinguish the sequences of pipeline components. The second attribute represents the total distance of each type of pipeline component from the starting point. It is denoted as T_i and is calculated as per equation (4-7).

$$T_i = \sum_{j=1}^{n_i} t_j \qquad 4\text{-}7$$

where;

i = pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, and coupling, and

t_j = the distance of each type of pipeline component from the starting point.

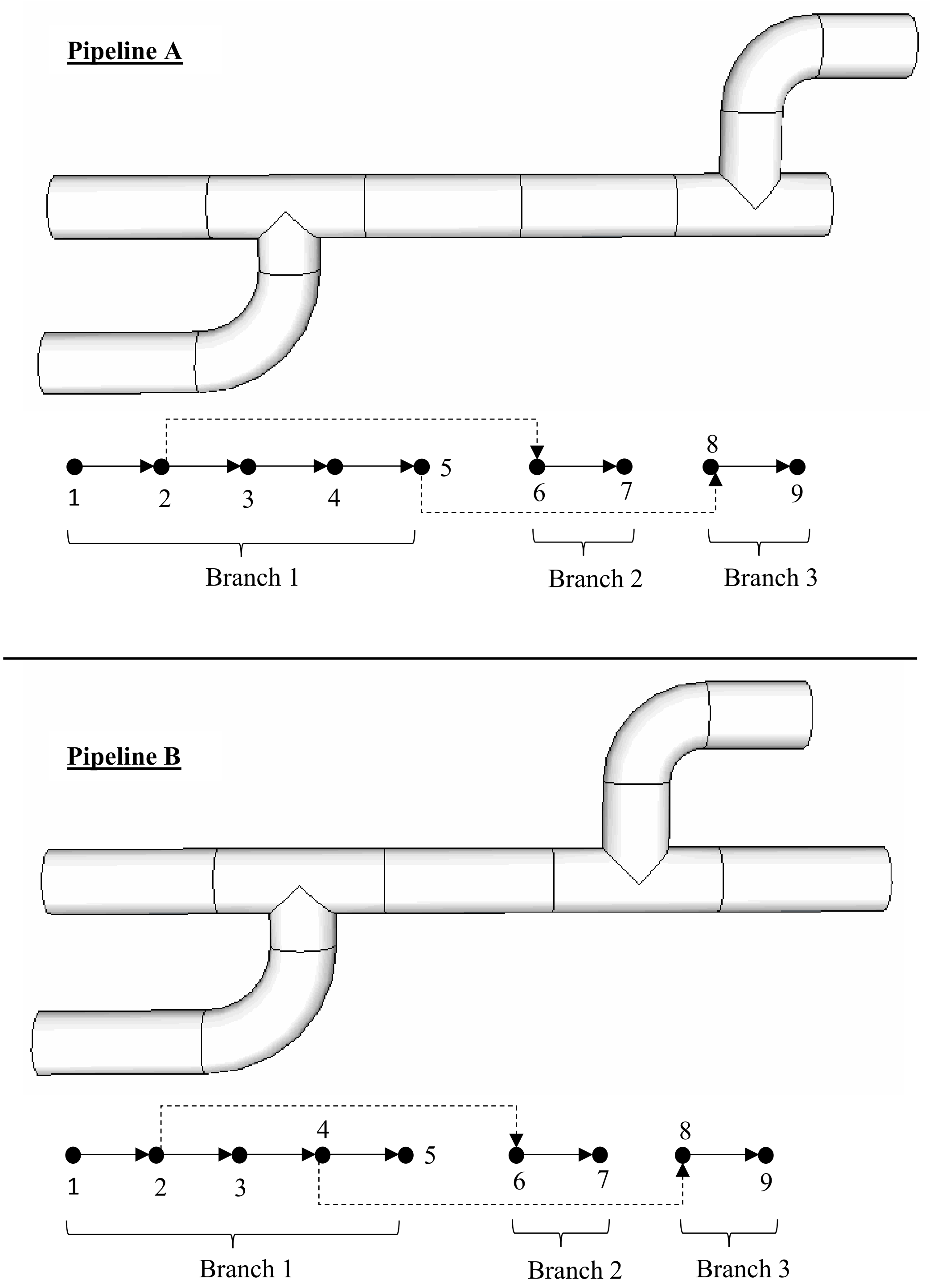


Figure 4- 12 Example of generation of pipeline features vector

It is assumed that the starting point for both pipelines is equal to 0 and the node numbers shown in Figure 4- 12 are the distances of each component from the starting point. Therefore, T_{Tube}, T_{Elbow}, and T_{Tee} for pipeline ***A*** are equal to 24 (the sum of the tubes' node location in the pipeline ***A*** strucutre, 1+3+4+7+9), 14 (the sum of the elbows' node location in the pipeline ***A*** strucutre, 6+8), and 7 (the sum of the elbows' node location in the pipeline ***A*** strucutre, 2+5) respectively; and they are equal to 25, 14, and 6 in pipeline ***B.*** The components' total distance from the starting point T_i provides a good presentation of the components' location in the pipeline sequence, making it possible to identify the differences between the two pipelines (as in tube and tee) as well as the similarities between them (as in elbow) . However, there might be some cases where T_i alone is not sufficient to represent the location differences between components from different pipelines. As an example of such case can be shown by relocating components tee in pipeline ***A*** from node 2 to node 1. The modified pipeline ***A*** structure is named ***A**** and is shown in Figure 4- 13.

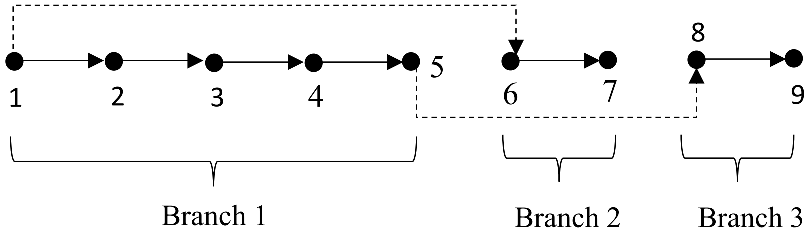


Figure 4- 13 Pipeline A* structure

This case changes the T_{Tee} of pipeline ***A*** to 6 instead of 7 which is equivalent to the T_{Tee} of pipeline ***B.*** This result means that both pipelines have the same total distance of components tee from the starting point and, more specifically, it means that the components' location is the same in both pipelines. However, in reality tee components in both pipelines are in different locations: they are located in nodes 1 and 5 in pipeline ***A***, and in nodes 2 and 4 in pipeline ***B***. Therefore, a third numerical attribute is proposed to work concurrently with the component total distance attribute T_i to further distinguish the location of each component in the pipeline sequence.

The third numerical attribute is the distribution of each component in the pipeline sequence and is denoted as D_i. It uses the variance of the distance of each component in the pipeline sequence to describe the distribution, and is defined as follows:

$$D_i = \sum_{j=1}^{n_i} \frac{(t_j - \mu_i)^2}{n_i} \qquad \text{4-8}$$

$$\mu_i = \frac{T_i}{n_i} \qquad \text{4-9}$$

where,

i = pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, and coupling,

t_j = the distance of each type of pipeline component from the starting point,

μ_i = the average total distance of each pipeline component,

T_i = the total distance of each component from the starting point, and

n_i = the total number of each type of pipelines' components.

As an illustration of how D_i may create unique component location characteristics, Table 4- 6 shows all three numerical attributes for component tee in pipelines ***A, B,*** and ***A****. The tee component in all pipelines has a unique location. Furthermore, when comparing pipeline ***B*** and pipeline ***A****, adding attribute D_i created a good differentiation base between pipeline component with the same n_i and T_i attributes.

Table 4- 6 Attributes n_{Tee}, T_{Tee}, D_{Tee} for pipelines **A, B**, **A***

Pipeline	n_{Tee}	T_{Tee}	D_{Tee}
Pipeline A	2	7	2.25
Pipeline B	2	6	1
Pipeline A*	2	6	4

As described above, each pipeline from the two sources, the original and the generated populations, is converted to a feature vector so it can be used to validate the Markov-chain pipeline generation model. Since nine components have been selected in the validation process, the pipeline feature vector is going to have 27 dimensions and the order of each attribute in pipeline feature vector should be consistent in both the original and generated data. The selected order of pipeline components is:

- 1st: Pcomponent $[n_{Pcomponent}, T_{Pcomponent}, D_{Pcomponent}]$
- 2nd: Instrument $[n_{Instrument}, T_{Instrument}, D_{Instrument}]$
- 3rd: Valve $[n_{Valve}, T_{Valve}, D_{Valve}]$

- 4th: Flange [n_{Flange}, T_{Flange} , D_{Flange}]
- 5th: Tube [n_{Tube}, T_{Tube}, D_{Tube}]
- 6th: Elbow [n_{Elbow} , T_{Elbow}, D_{Elbow}]
- 7th: Tee [n_{Tee} , T_{Tee} , D_{Tee}]
- 8th: Reducer [$n_{\mathrm{Re}ducer}$, $T_{\mathrm{Re}ducer}$, $D_{\mathrm{Re}ducer}$]
- 9th: Coupling [$n_{Coupling}$, $T_{Coupling}$, $D_{Coupling}$]

and the final representation of each pipeline feature vector is:

$$[n_{Pcomponent}, T_{Pcomponent} , D_{Pcomponent}, \ldots\ldots\ldots\ldots\ldots\ldots, n_{Coupling}, T_{Coupling}, D_{Coupling}]$$

4.5.2 Number of components evaluation and correlation analysis

In this section, the number of components (n) generated by the Markov-chain pipeline generation model is compared to that in the original pipeline population. **Table 4- 7** shows summary results of the number of each type of pipelines' components. The Markov-chain pipeline generation model generated approximately the same total number of components with 2.5% higher what than the original pipeline populations. However, the breakdown of each component in the two populations differs in some ways. Significant differences can be seen between the original and the generated pipelines at components pcomponent, instrument, and tee; meanwhile, the rest of the components showed small differences. This indicates that the assumptions regarding the number of components when constructing the Markov-chain model worked well for six out of nine components. Additionally, assessing the number of occurrences of each component in each pipeline in the two populations can provide good insight into how the original and generated pipeline populations are similar. For this work, since the distributions describing the pipeline components are not normally distributed (as indicated by the Anderson-Darling procedure, see Table D-1 in Appendix D), the two-sample t-test cannot be applied. As a result, non-parametric tests in the form of Kruskal-Wallis, Mood-Median, and Mann-Whitney tests were applied to assess the ways in which the two populations are similar. Table 4- 8 shows *p*-values of all three tests when applied to each pipeline component. *P*-values of Kruskal-Wallis and Mood-Median tests are greater than the significance level (α=0.05), which means that in terms of the number of each component, there are no significant differences between the original and the generated pipeline populations. However, the Mann-Whitney test showed different results: components tube*,* elbow*,* reducer*,* and coupling did not differ significantly from the two populations. Meanwhile, the rest of the components showed the opposite result. These tests are one-dimensional and applied to each component independently from others. Furthermore, the combination of all components including their attributes, and the number of components and their location in the pipeline sequence actually represent the overall pipeline characteristics. For this reason, the other two validation methods proposed in this study will complement the results driven by this section.

Table 4- 7 Number of components in original and generated pipelines

Component	**Original**		**Generated**	
	n	**%**	*n*	**%**
Pcomponent	451	1.5	935	3.0
Instrument	557	1.8	199	0.6
Valve	2611	8.6	1947	6.3
Flange	4101	13.6	3036	9.8
Tube	12036	39.9	13888	44.9
Elbow	6332	21.0	8699	28.1
Tee	3268	10.8	1505	4.9
Reducer	788	2.6	670	2.2
Coupling	55	0.2	59	0.2
Total No. of pipeline components	30199		30938	

Table 4- 8 P-values of Kruskal-Wallis, Mood-Median, and Mann-Whitney tests

Component	***p*-value**		
	Kruskal-Wallis	**Mood-Median**	**Mann-Whitney**
Pcomponent	0.276	0.092	0
Instrument	0.241	0.267	0
Valve	0.185	0.123	0
Flange	0.7	0.809	0
Tube	0.314	0.835	0.2452
Elbow	0.797	0.809	0.2358
Tee	0.415	0.345	0
Reducer	0.624	0.585	0.767
Coupling	0.929	1	0.748

In building the Markov chain pipeline generation model, it was assumed that the integration of both the Markov chain transition matrix and the state periods of each component can preserve the correlation between pipeline components. Therefore, correlation coefficient matrices for pipeline components in both the original and generated data were calculated (see Appendix E), and to measure the similarity between both matrices, an eigenvalue-based similarity measure was employed. Figure 4- 14 illustrates the comparison between eigenvalues calculated from the correlation matrices associated with the original and simulated data. Since in both cases, the eigenvalues are comparable except for the component coupling which exerted a noticeable difference, and since this component accounts only for 0.1-0.15 % in both populations, it is possible to conclude that the Markov-chain pipeline generation model was capable of maintaining the correlation between the components.

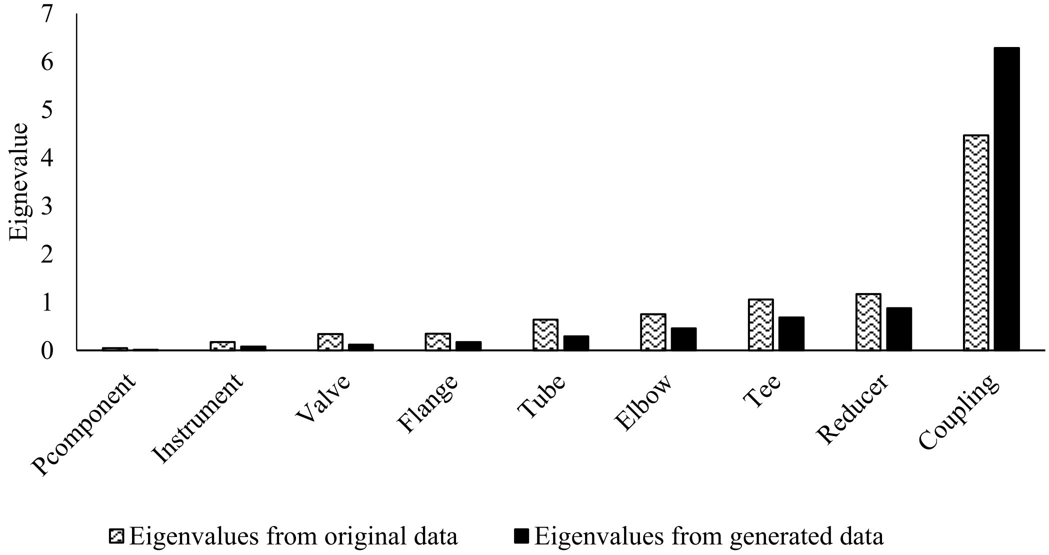


Figure 4- 14 Eigenvalue distributions of component correlation matrices from original and generated pipeline population

4.5.3 Clustering-based model validation

In this section, a clustering-based validation method is proposed to measure similarities between the original and generated pipeline spaces. A clustering technique was employed in this study because of its ability to find a true topology of certain data [89]. The main concept of this validation method was to assess the performance of a clustering model built using two different sets of data. A similar methodology was implemented by Sikonja [90] to compare the similarity of a generated data set against an original data set. Many clustering algorithms were proposed to carry the task of clustering analysis. A density-based clustering algorithm is selected in this study. It was selected because it does not require the pre-determination of the number of clusters to be generated; it has the ability to define clusters of higher density, and provides the optimum number of clusters accordingly (refer to [91] for more details on density based clustering). Table 4- 9 shows a summary of the clustering of the original and generated pipeline feature vectors (refer to Table F- 1 and Table F- 2 in Appendix F for more details on the clustering results). It has two parts: the clustering model and the test results. The clustering model is built using 80% of the total population and the rest of the data is used to test the model. The density-based clustering resulted in generating two clusters for each pipeline space: one that was high density cluster (cluster 0 from the original data source shown in Table 4- 9) and one that was low density cluster (cluster 1 from the original data source shown in Table 4- 9). In the original data clusters, the distribution of the number of instances in the high and low density clusters was 75.4% to 24.6% respectively. In the generated data clusters, the distribution of the number of attributes was 73.65 to 26.4% respectively; 1.8%

different from the original data. The same trend was found in test results using the remaining 20% of total data, which means that the spread of pipeline feature vectors from different sources in the pipeline space was approximately the same. However, using these results alone is not sufficient to verify that the original and generated pipeline spaces share the same characteristics. There might be a case where two datasets share the same number of clusters, but the location of these clusters may differ significantly. Analyzing cluster centroids can provide a good comparison basis. Figure 4- 15 and Figure 4- 16 illustrate the centroids of the high and low density clusters generated from both the original and generated pipeline feature vectors. The centroids of clusters calculated from the original and generated pipeline feature vectors have a similar trend. The degree of similarity between the original and the generated centroids in both the high and low density clusters was equal to 64%. These results confirm that in terms of the number and location of components in the pipeline, the Markov-chain pipeline generation model is producing reasonable outputs.

Table 4- 9 Results summary of clustering models

Model	Data source	Total no. of attributes	Clusters	Attribute	SSE
Clustering model	Original	841	0	793	
			1	48	
	Generated		0	67	
			1	774	
Test results	Original	221	0	179	165.6
			1	32	
	Generated		0	39	115.6
			1	172	

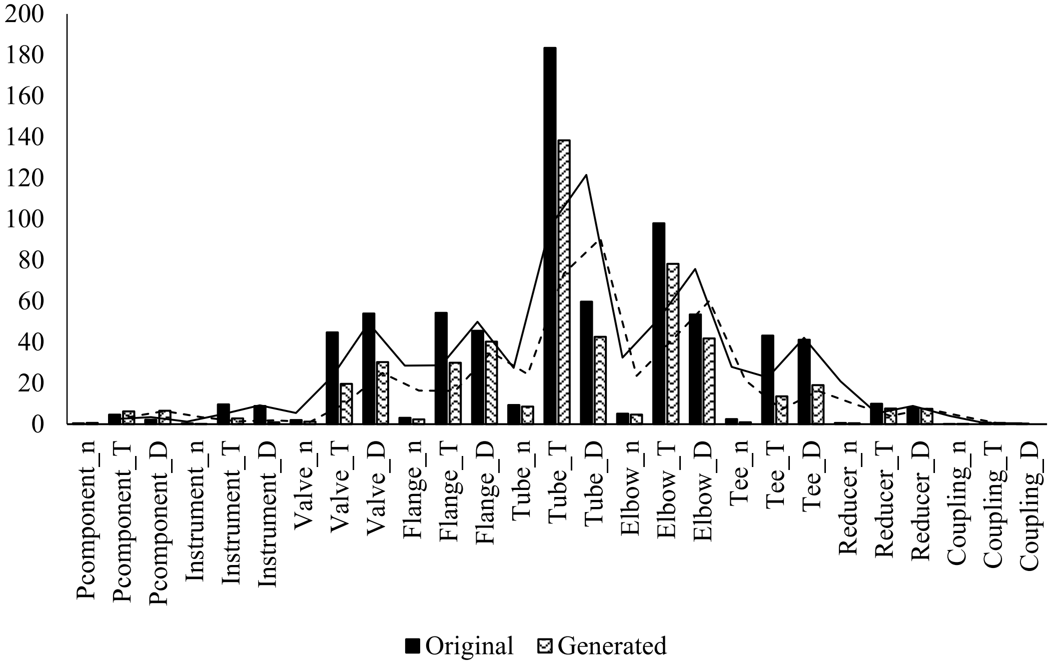


Figure 4- 15 Attributes centroids in the high density cluster

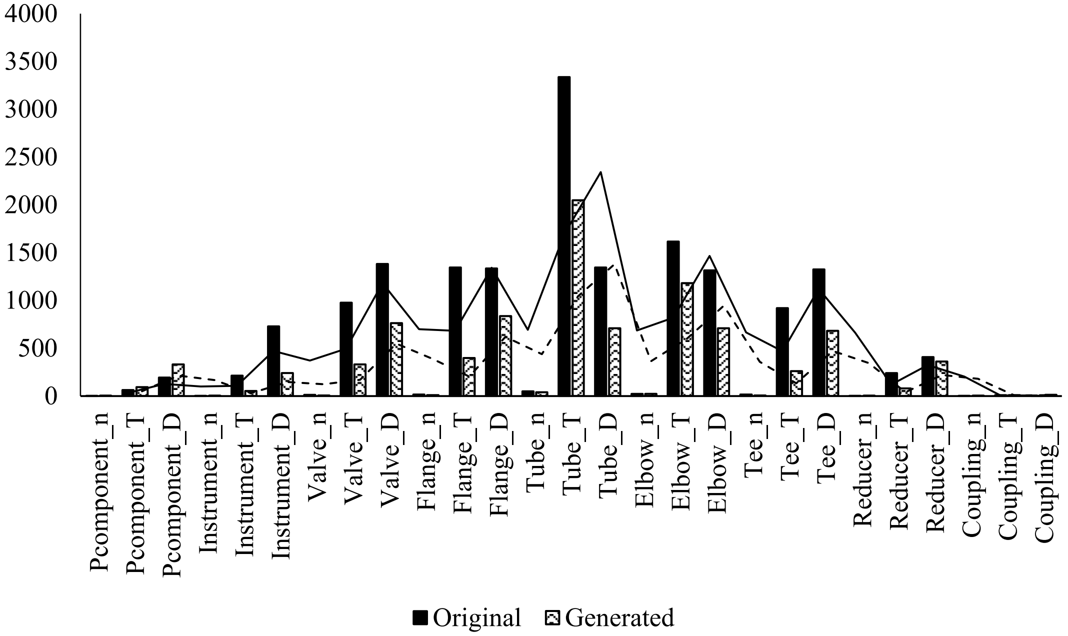


Figure 4- 16 Attributes centroids in the low density cluster

4.5.4 Model validation using distances between all feature vectors

In this section, the distance between all feature vectors in both the original and generated pipeline data sets are calculated using the Euclidian distance. The distance between feature vectors indicates the degree of similarity between pipelines; if two pipelines are similar, the distance between their vectors should be small, and if not, the distance should be large. This approach is well used for data clustering and classification [92]; however, it is used herein to assess the similarity of distances between pipelines feature vectors calculated from the original and the generated pipeline populations. The Euclidian distance formulation is defined as:

$$E = \sqrt{\sum_{j}\sum_{i}(j_i - j_i^{'})^2} \qquad 4\text{-}10$$

where,

E = the Euclidian distance between pipelines vectors,

i = pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, and coupling, and

$j, j' \in \{ n, T, D\}$ and satisfying $j \neq j'$.

Since the total number of pipelines in each population is equal to 1052, by using the equation $\frac{\text{no. of pipelines} \times (\text{no. of pipelines} - 1)}{2}$, it is expected to have 552826 distance data points for each population. The distance data points' distribution is shown in the histogram plots in Figure 4- 17. The data shown in both histograms are highly skewed to the right, which requires a careful attention to the tests being used to assess the similarity between the two populations. According to Figure 4- 18, it turns out that both populations are not normally distributed as confirmed by the Anderson-Darling normality test results for which the p-value < 0.05. As a result, parametric methods such as the “t” or “f” test, which rely on the normality of the data, cannot be applied. However, the distance between pipeline vectors represents the degree of similarity which lies between $[0, \infty)$. Grouping these distances leads to the identification of the similarity of topological structures of both the original and the generated pipeline populations. Furthermore, the histogram plots shown in Figure 4-17 represent the similarity between topological structures of both populations. Thereafter, using histogram similarity measures in the form of histogram intersections can provide a good basis for assessing degrees of similarity.

Another result which can be derived from Figure 4- 18 is that there is a gap separating the generated pipeline data from the points in the probability plots of the distance between pipeline vectors. Although both populations have approximately the same mean values, as shown in Figure 4- 18, this gap created significant differences in the standard deviations. This result justifies the clustering-based validation method in 4.5.3, which concluded that both populations were split into two clusters. As a result, the population of the distances between pipelines is split into two: the distances that represent the 95% of the whole population and the distances that represent the higher 5% of the population. Each population is then analyzed individually.

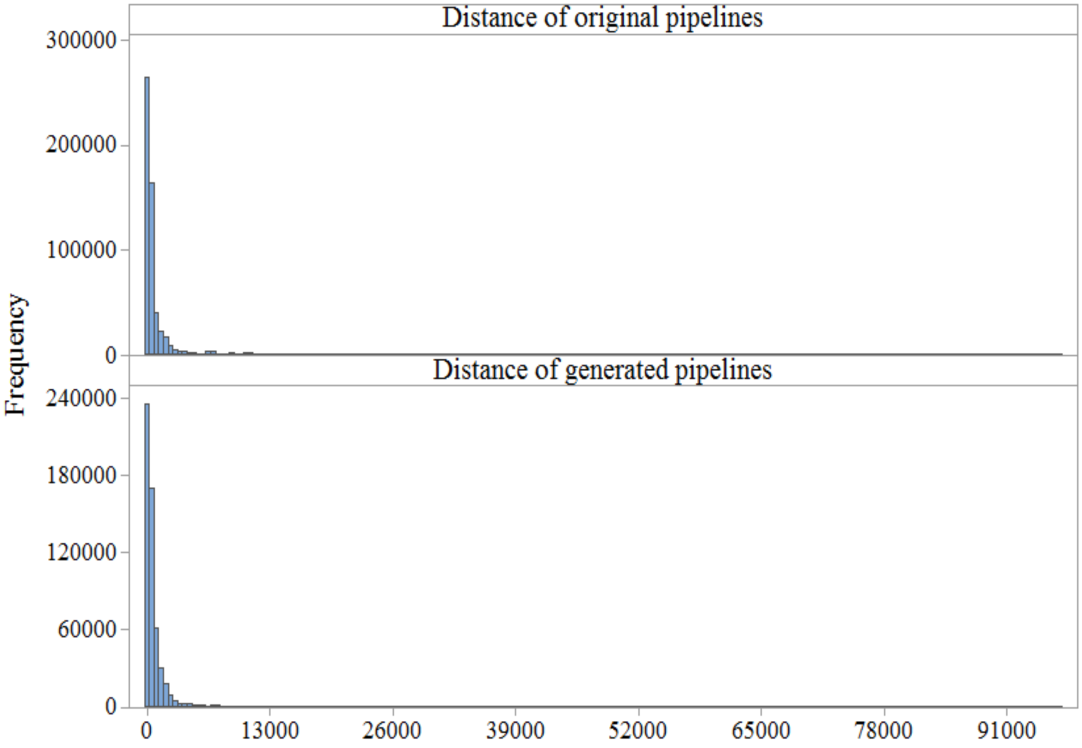


Figure 4- 17 Histograms of distances between pipeline vectors for the original and the generated pipeline populations

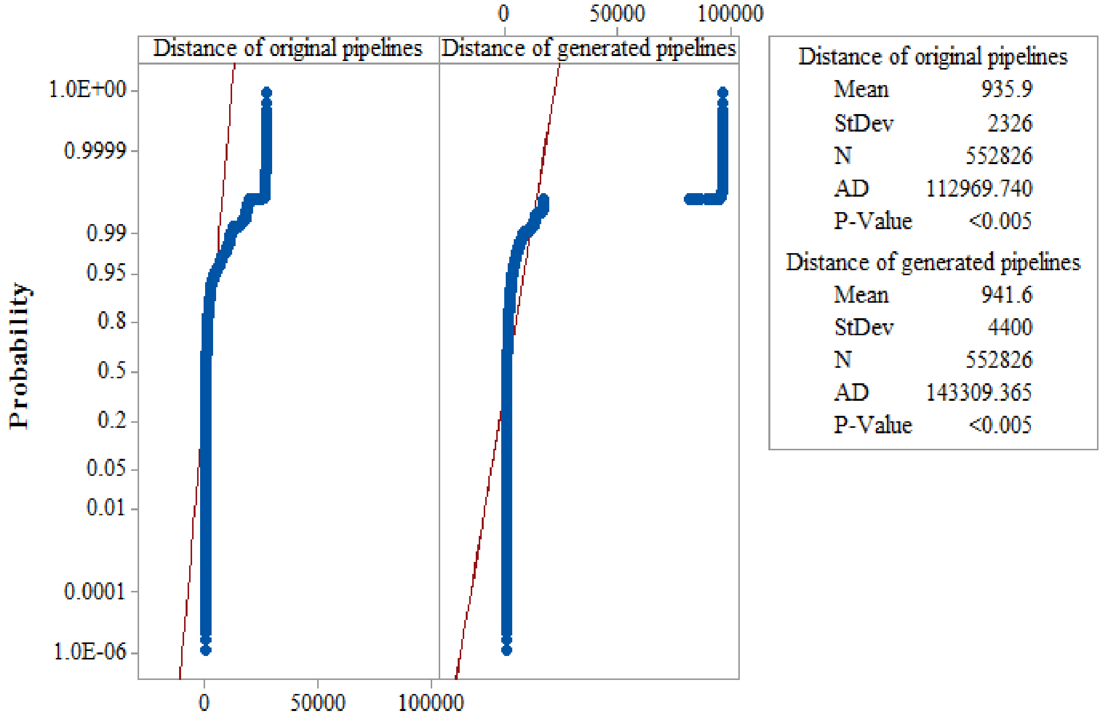


Figure 4- 18 Probability plots of distances between pipelines vectors from the original and generated pipeline data

Figure 4- 19 shows the distribution of the distances between pipelines vectors that underlies 95% of the total populations. The distribution of the distances calculated from the original pipeline population represents 899 pipelines (equivalent to 85.5 % of the total number of pipelines) and in the generated pipeline population, it represents 936 pipelines (equivalent to 89% of the total number of pipelines). Both distributions are highly skewed with no trace of normality. Figure 4- 20 shows the probability plot of the distances that underlie the 95% of the total population. Both data have approximately the same mean ($\mu_{original}$=658.4, $\mu_{generated}$=668.1), and they are slightly different in terms of the standard deviations ($\sigma_{original}$=1176, $\sigma_{generated}$=985). These results indicate that in terms of mean and standard deviations, 89% of the pipelines generated from the Markov chain pipeline generation model have characteristics similar to 85.5 % of pipelines in the original populations.

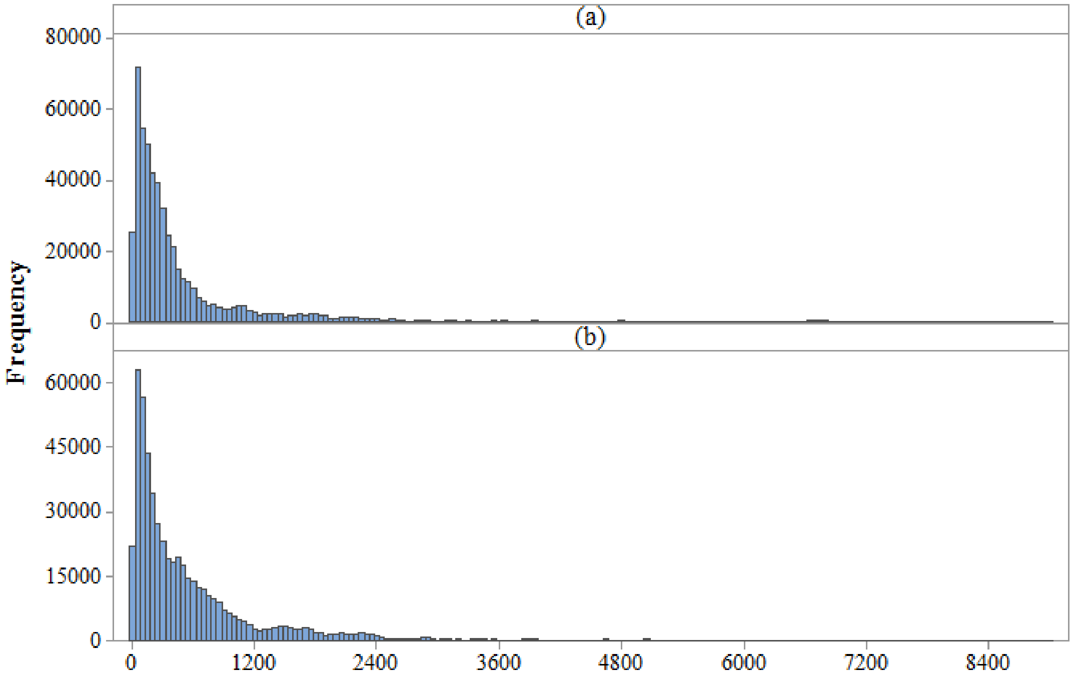


Figure 4- 19 Histograms of distances between pipelines vectors that represent the 95% in (a) the original and (b) the generated pipeline populations

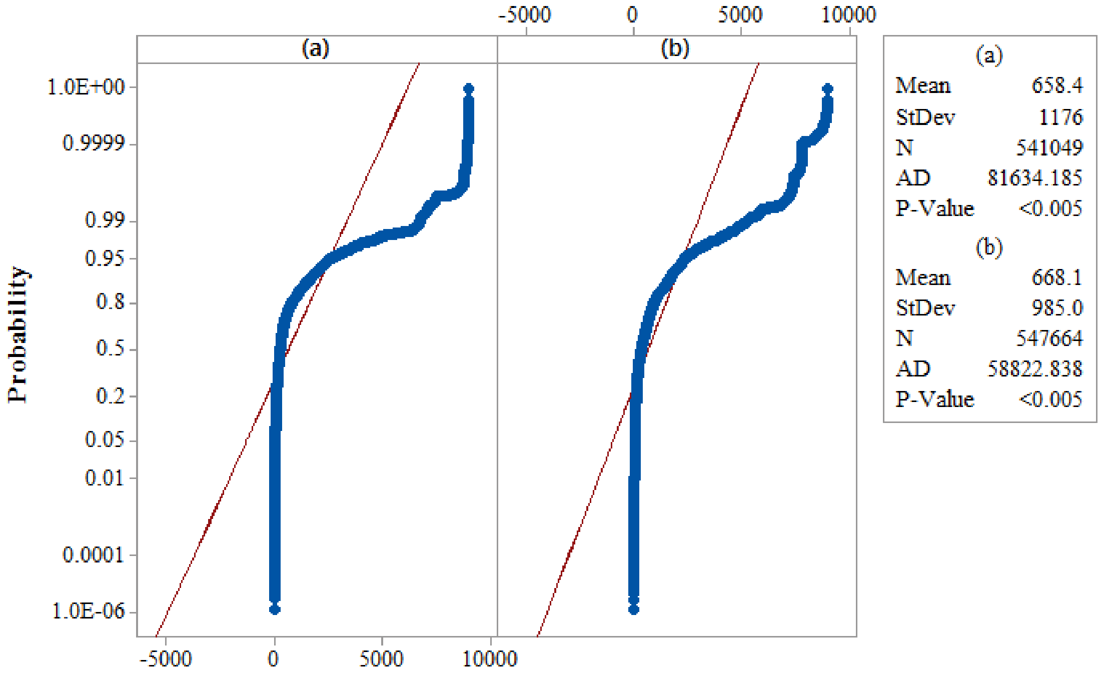


Figure 4- 20 Probability plots of distances between pipelines vectors that represent the 95% in (a) the original and (b) the generated pipeline populations

The other test to apply in this section is the histograms intersection. It is proposed by Swain and Ballard [93] and described as follows: given a pair of histograms x (histogram of distances between generated pipelines) and y (histogram of distances between original pipelines), with each having the same number of bins n, the intersection between histograms is defined as:

$$\sum_{i=1}^{n} \min(x_i, y_i) \qquad 4\text{-}11$$

where,

x_i= bin i in histogram x,

y_i= bin i in histogram y.

Equation (4-11) is then normalized, as in (4-12), to calculate a fractional match value between the two histograms $H(x, y)$. The fractional match value H lies between [0, 1] and the degree of similarity is interpreted as if H is equal to 0, which means the histograms are not identical. If H is equal to 1, the histograms are identical.

$$H(x, y) = \frac{\sum_{i=1}^{n} \min(x_i, y_i)}{\sum_{i=1}^{n} y_i} \qquad 4\text{-}12$$

Figure 4- 21 illustrates the intersection between the histograms of the normalized distance of the original and generated pipeline populations. Visually, the two histograms are almost identical. This conclusion is supported by calculating the match value. Table 4- 10 shows a sample calculation of the match value. The match value H is found to equal to 0.979 which means that the two histograms are almost identical. This result is based on the number of bins equal to seven, which is driven by Sturges' formula ($\log_2(n)+1$). However, the match value, based on Equation 4-13, is sensitive to the number of bins. The higher the number of bins used, the more accurate the measure is. Therefore, the calculation shown in Table 4- 10 is repeated 10 times with a 10-bin increment added in each run. The changes in H with respect to the bin number are shown in Figure 4- 22. The H value converges to 0.88 which represents the actual similarity measure between the two histograms.

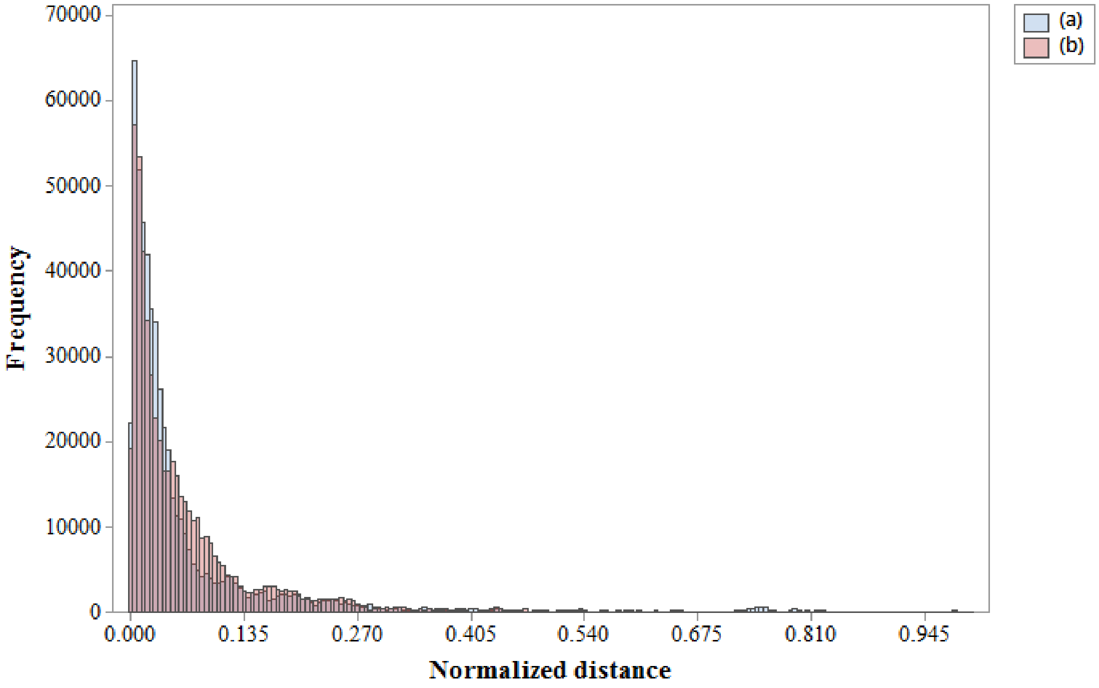


Figure 4- 21 Histograms intersection between: (a) histograms of normalized distances between original pipeline vectors that represent the 95% of the population, and (b) histogram of normalized distances between generated pipeline vectors that represent the 95% of the population

Table 4- 10 Calculation of histograms intersection: (a) histograms of normalized distances between original pipeline vectors that represent the 95% of the population, and (b) histogram of normalized distances between generated pipeline vectors that represent the 95% of the population

Bin #	**Bin density**			***H***
	x-Histogram (a)	y -Histogram (b)	$\min(x_i, y_i)$	
1	6.065	6.000	6.000	0.979
2	0.572	0.696	0.572	
3	0.146	0.136	0.136	
4	0.078	0.089	0.078	
5	0.030	0.043	0.030	
6	0.088	0.032	0.032	
7	0.020	0.004	0.004	
$\sum$	7.000	7.000	6.852	

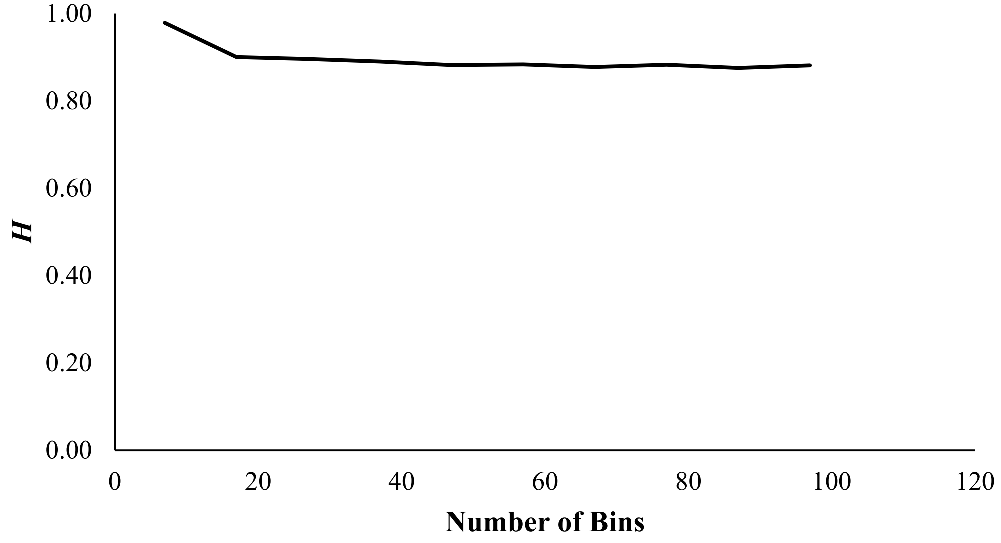


Figure 4- 22 Changes in match value H with respect to the increase of histograms' bin number in the distances that underlie the 95% of the total population

Figure 4- 23 shows the distribution of distances between pipelines that represents the higher 5% of the total population. Distances calculated from the original pipeline population represent 156 pipelines. Distances calculated from the generated pipeline population represent 116 pipelines. A significant difference in terms of mean and standard deviation can be seen in Figure 4- 24, which means that these two populations are significantly different. Moreover, when a histogram intersection measure was applied, as shown in Figure 4- 25 and Table 4- 11, the match value H was equal to 0.504. The match value is calculated based on a bin number equal to 14, driven by Sturges' formula, but when increasing the number of bins, the H value converges towards 0.29. Figure 4- 26 shows 10 runs with an increment of 10 bins added in each run. The H values change dramatically. When extending the increase in the number of bins to 1000, the H value becomes steady at 0.21. This confirms that the distances representing the higher 5% of the total original and generated populations share a small trace of similarity equivalent to 0.21 in the range of 0 to 1.

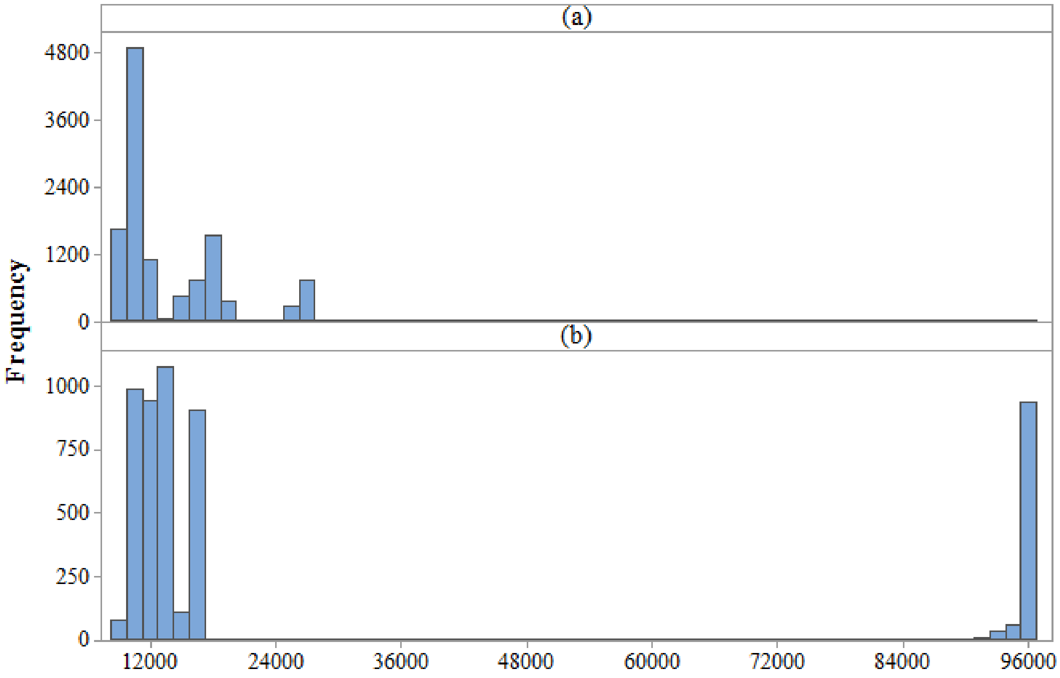


Figure 4- 23 Histograms of distances between pipelines vectors that represent the higher 5% in (a) the original and (b) the generated pipeline populations

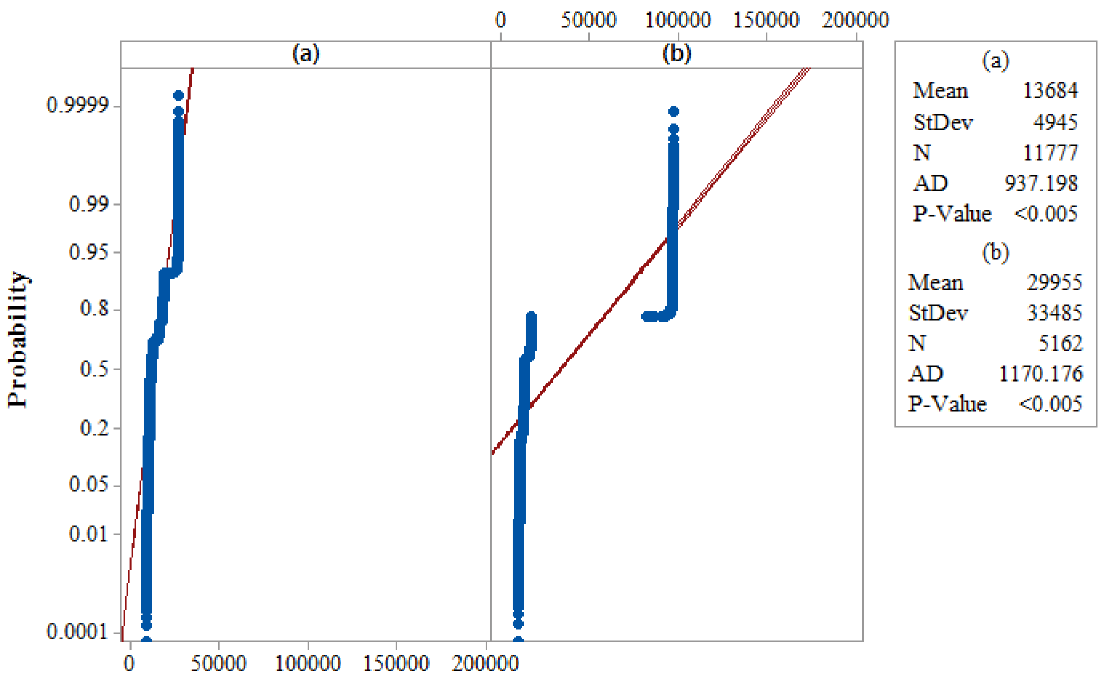


Figure 4- 24 Probability plots of distances between pipelines vectors that represent the higher 5% in (a) the original and (b) the generated pipeline populations

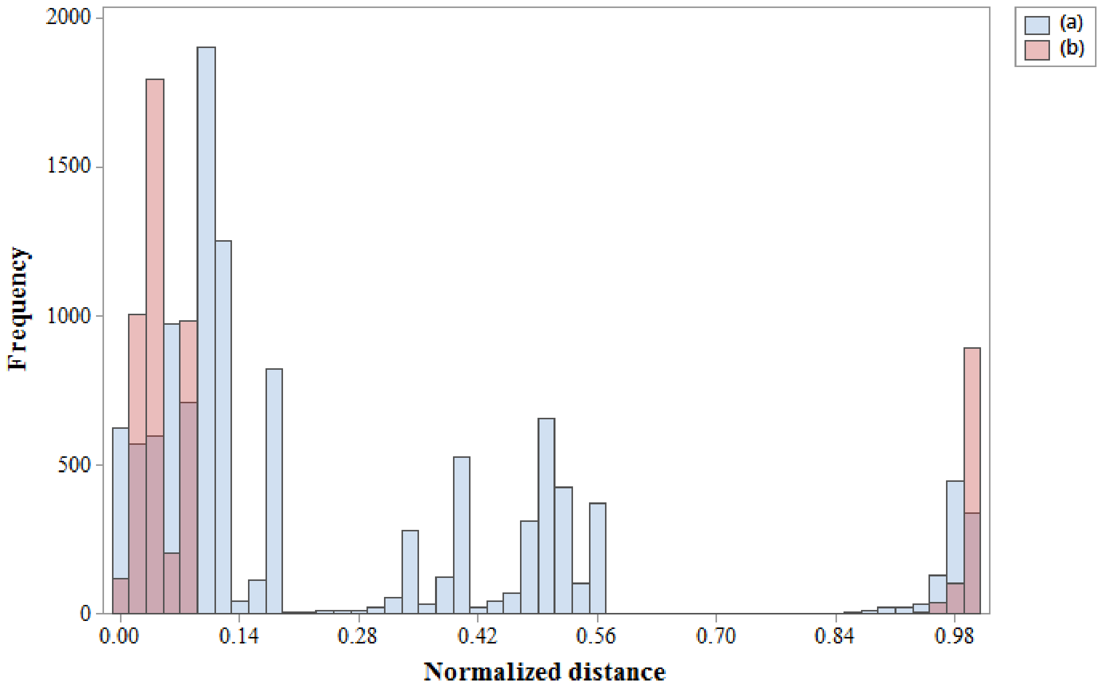


Figure 4- 25 Histogram intersection between; (a) histograms of normalized distances between original pipeline vectors that represent the higher 5% of the population, and (b) histogram of normalized distances between generated pipeline vectors that represent the higher 5% of the population

Table 4- 11 Calculation of histograms' intersection; (a) histograms of normalized distances between original pipeline vectors that represent the higher 5% of the population, and (b) histogram of normalized distances between generated pipeline vectors that represent the higher 5% of the population

Bin #	**Bin density**			***H***
	x-Histogram (a)	y -Histogram (b)	$\min(x_i, y_i)$	
1	3.315	8.511	3.315	0.508
2	4.608	2.639	2.639	
3	1.142	0.000	0.000	
4	0.048	0.000	0.000	
5	0.442	0.000	0.000	
6	0.836	0.000	0.000	
7	1.251	0.000	0.000	
8	1.123	0.000	0.000	
9	0.002	0.000	0.000	
10	0.010	0.000	0.000	
11	0.005	0.000	0.000	
12	0.017	0.005	0.005	
13	0.068	0.014	0.014	
14	1.134	2.831	1.134	
$\sum$	14.000	14.000	7.107	

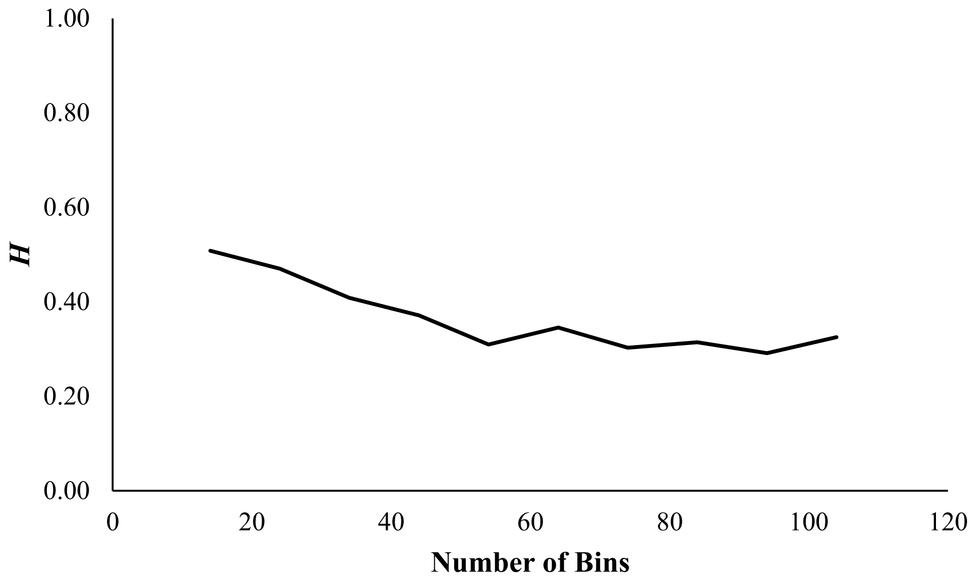


Figure 4- 26 Changes in match value H with respect to the increase of histograms' bin number in the distances that underlie the higher 5% of the population

4.6 Conclusion

In this study, a Markov-chain model is proposed to randomly generate pipeline data structures. Model development is motivated by the need to generate random input that represents the diverse and complex nature of the product to support simulation of construction fabrication processes. The construction of the Markov-chain model was preceded by original pipeline data preparation and analysis and succeeded by intensive validation processes. The study also illustrated how to convert pipeline data structure to a feature vector that can preserve pipeline characteristics prior to applying validation processes. The performance of the Markov-chain pipeline generation model was measured against a real dataset. In terms of the number of the generated pipelines' components, the model generated approximately the same collection of components as in the original pipelines. The model also maintained the correlation between the generated pipeline components that existed in the original pipelines. The results show that the model is reasonably similar to the original pipelines. Using a clustering-based model validation as a similarity measure shows that the Markov-chain model generated a reasonable performance. In the context of model validation using the distance between all feature vectors, it was found that the majority of the generated pipelines (89% of the total population) shared characteristics similar to 85.5% of the original pipeline populations with a degree of similarity of 0.88 (on a scale of 0 to 1 with 0 meaning not identical, and 1 meaning identical). Meanwhile, the rest of the generated pipelines (11% of the total population) were significantly different when compared to the original pipelines. This confirms that the Markov chain model used to randomly generate pipeline data structure is capable of generating the dominant characteristics of pipelines found in reality.

Chapter 5

Application of industrial pipeline data generator for testing the efficiency pipe modules optimization algorithms

5.1 Introduction

Three approaches have been the main pillars in analyzing and evaluating the efficiency of solution algorithms: theoretical worst-case analysis, theoretical average-case analysis and experimental analysis [94]. Experimental analysis is the most widely used method of growing interest in the operational research community [95]. It is used to test and analyze the performance of algorithms by running them on sets of instance problems. These sets of problems are either extracted from a real-world system or randomly generated using a synthetic data generator, which in both cases results in generating valuable knowledge. However, the use of real-world instances is associated with difficulties related to the size and the number of the available test instances. In addition, although it represents a problem's real behavior, the high cost of the data collection and documentation limits testing scenarios that researchers may examine [96]. Synthetic data generators provide a solution to such difficulties, as well as flexibility in generating test instances of different properties such as the size and complexity of an instance problem. Such properties are of great importance in the experimental analysis of NP-hard problems in which the computational time increases dramatically with the size of the problem [95] [97].

Data generators, in the context of experimental design and analysis, are highly recommended to be used [98]. Different data generators for different problems, in the field of operational research, have been well documented and referenced in literature for researchers to generate data sets for computational experiments. For example, Drexl et al. [99] introduced an activity network generator to generate instance problems for a resource-constrained project scheduling problem. Gau and Wascher [100] introduced a problem generator for a one-dimensional cutting stock problem, and Silva et al. [101] introduced a problem generator for a two-dimensional rectangular cutting and packing problem. These examples of instance problem generators are used in a defined class of problems. They provide flexibility in generating a large number of instance problems with different properties and define a standard data set to be used in evaluating the efficiency of newly developed algorithms [100]. Similarly, in this chapter, an industrial pipeline data generator constructed in the previous chapter is introduced as a pipeline instance problem generator. The pipeline instance problem generator is designed to generate test sets for testing solutions' algorithms applied on industrial project problems. However, it is first important to identify and define the problem at which the industrial pipelines' data generator will be targeted. Therefore, this chapter is structured as follows:

1. Section 5.2 presents an overview of industrial projects related to modularization, modules, and pipe spools. It includes the problem definition and mathematical formulation. The problem definition, which is related to the pipe-spooling process in industrial construction, was conducted in cooperation with a partner company which has significant experience in industrial construction.
2. Section 5.3 presents background about the bin-packing problem. The selection of this class of problem is based on the mathematical formulation of the pipe-spooling problem. This section includes an overview of the bin-packing problem and its heuristics, the projection of the pipe-spooling problem as a three-dimensional bin-packing problem, and the heuristic (branch-and-bound) that is proposed to approximate the pipe-spooling solution.
3. Section 5.4 presents the generation of pipeline instance problems. It includes a description of the additional pipeline attributes (pipeline component lengths, diameter and running direction) required by the pipe-spooling problem and the integration of these attributes in the industrial pipeline generator.
4. In Section 5.5, a computational experiment is conducted, and its results are reported as benchmark results for the proposed heuristic in approximating the pipe-spooling solution.

5. Section 5.6 presents the future use of the industrial pipeline data generator and Section 5.7 presents the conclusion of this chapter.

5.2 Overview of modularization, modules, and pipe spools in industrial projects: The defined problem statement

In industrial projects such as petrochemical plants, petroleum refineries, and oil and gas production facilities, piping systems are the major and most complex elements, accounting for ; 40% of the total time and budget [102]. These costs lead the project owners to increase their demands for safety, quality, productivity and performance of their projects [103]. Such demands have led to an increased interest in using time- and cost-efficient construction techniques. The modularization of construction elements has a positive impact on construction operations throughout the industrial project life cycle. Modularization has improved productivity in the construction industry, resulting in (1) improved project schedules, (2) budget and cost reductions, (3) improved site safety, (4) waste reduction, (5) reduced weather impact on the fabrication process, (6) reduced field labour requirements, and (7) improved quality [104].

Modularization is defined as “the preconstruction of a complete system away from the job site that is then transported to the site” [105]. The system may be too large to transport, so is broken into smaller units called modules. These modules are defined as “a major section of a plant resulting from a series of remote assembly operations and may include portions of many systems; usually the largest transportable unit or component of a facility” [106]. The modularization technique is part of a construction methodology called “prework” that includes prefabrication, preassembly, modularization, and off-site fabrication processes (PPMOF) [107]. The adoption of "prework" has increased in the construction industry; it was found that during the past 15 years, the implementation of prefabrication has increased by 86% and the implementation of both prefabrication and preassembly increased by 90% [108].

Modularization as a part of the "prework" construction method has demonstrated great potential in terms of increasing efficiency in industrial construction. However, it is also associated with disadvantages, identified by Dawar et al. [109], such as (1) the increase in engineering and home office costs, (2) the increase in fabricator quality surveillance, expediting and logistical costs, (3) additional cost and scheduling for transportation and logistics, (4) additional cost and scheduling for preliminary studies in the early stages of the project, and (5) an increase in installation costs due to heavy modules. In general, these disadvantages can be divided into two major groups: (1) the expensive mode of transportation and excessive logistics planning required in the area of modular construction, (2) and the excessive requirements of engineering [110]. The first group, transportation and logistics, plays a critical role in modularization. Failure to properly plan the transportation and logistics of industrial projects may cause catastrophic costs and scheduling damages. For example, the Kearl oil sands project located in northern Alberta, Canada experienced cost and schedule overruns due to transportation and logistics problems [111]. Korean-made modules were shipped to Canada, but due to their large size and the transportation regulations, Kearl had to break 200 modules into smaller pieces for shipping and then reassemble them on site. This unanticipated problem increased the project cost by $2 billion. Constructing transportation and logistics strategies is imperative to increase the efficiency of modularization in an industrial project, but building these strategies is a complex process. It involves route identification, constraints, transportation envelopes, and scheduling availability. Murtaza et al. [110] identified and defined areas of consideration in planning transportation and logistics strategies. These areas, their definition, and citations are shown in Table 5- 1.

Table 5- 1 Areas of consideration in planning transportation and logistics strategies *[110]*

No.	Area	Definition	Reference
1	Total delivered cost management	"Ability to analyze and predict the total supply chain costs from the source to the point of distribution. It includes the capability to roll up both international and domestic logistics costs by product and delivery route, plus the ability to accurately calculate all the applicable duty, tariffs and other customs-related costs while factoring in any preferential trade agreements. More advanced capabilities would include the ability to model and estimate inventory levels and total carrying costs"	SupplyChainDigest [112]
2	Supplier portals and advance ship notice (ASN) capability	"Web portals that provide some level of visibility, the ability to generate ASNs, and print barcode labels. Shippers post freight movement requests and/or detail ASN notices delivered"	SupplyChainDigest [112]
3	Total product identification and regulatory compliance	"Systemized approach to identify products and ensure conformance to regulatory and export rules"	SupplyChainDigest [112]
4	Dynamic routing	"System modeled rates/lanes give realistic view of cost/service advantages between shipping alternatives"	SupplyChainDigest [112]

No.	Area	Definition	Reference
5	Variability management	"Ability to manage in-transit exceptions more effectively"	SupplyChainDigest [112]
6	Integrated international and domestic workflow	"Reduction in total logistics costs through a more holistic approach to process and carrier/mode coordination across international and domestic moves"	SupplyChainDigest [112]
7	Integrated planning and execution platform	"End-to-end, optimized global logistics control and cost minimization"	SupplyChainDigest [112]
8	Global logistics process automation	"Transitioning from manual intensive processes and adopting such things as internet-based transaction automation technology"	AberdeenGroup [113]
9	End-to-end visibility	"Increased visibility of logistics process steps creates control"	AberdeenGroup [113]
10	Financial supply chain management	"Financial supply chain management is about looking at how to optimize working capital of a company"	Kristofik et al. [114]

Reading through Table 5- 1 and from the industrial construction perspective, the areas of consideration which have a potential effect on modularization are total product identification and regulatory compliance, integrated international and domestic workflow, and the integrated planning and execution platform. Improving these areas increase confidence in selecting modular construction in an industrial project. This fact is supported by results obtained by O'Connor et al. [115], who determined 21 influential critical success factors that led to an effective use of modularization. These factors were generated from a survey conducted on more than 170 modular projects located in 13 countries with industry representatives made up of owners, contractors, design firms, and fabricators (a detailed description of each critical success factor is presented in Appendix G.). The most influential success factor was understanding the module envelope limitations, a factor that aligned with the finding in Table 5-1**,** which is the consideration of total product identification (standardizing the module design) and regulation compliance in modular construction. It is important to accurately define and identify the product/module in accordance with the transportation envelope limit. This makes it possible to recognize modularization's cost savings in the front-end-planning stage.

Cost recognition is achieved the advanced planning for contractors' resources; it includes testing construction methods and transportation strategies under different expected scenarios. Knowing that owners/contractors may use local and international fabricators or suppliers to achieve the most cost efficient modular construction strategies [116] results in an investigation of new research directions in the field of modularizing industrial projects. A research direction that integrates product optimization with the optimization of transportation process to support decision making in the front-end planning stage.

5.2.1 Modules and pipe spools

In industrial construction projects, a module is the main construction element/product. There are different types, identified by Dawar et al. [109] and shown in Table 5- 2. Most require prefabrication, preassembly, and transportation. Most of the modules in industrial projects have common formation components which are structural steel frames that include racks of pipes, cables, and equipment [117]. More specifically, they are composed of construction materials and installed equipment. These components account for 50 to 60% of the total cost of the industrial project [118]. Construction materials are further classified as off-the-shelf (e.g., nuts, bolts, steel sheets, elbows, small pipes, and hand valves), long-lead bulks (items or material require a long time to design and fabricate) and engineered items (e.g., pipe spools) [119]. Of these categories, engineered items in the form of pipe spools are featured as high-cost unique items that require more front-end planning effort [120].

Table 5- 2 List of the common types of modules in the petrochemical industry *[109]*

No.	Module type	Definition
1	Pipe rack and pipe bridge modules	Modules loaded with pipes, electrical trays, and attachments. These modules are usually fully fabricated and loaded at the shop and then installed in the field on pile and concrete foundations.
2	Structural modules	Modules without any equipment or pipes.
3	Skidded packages	A small vendor furnished modules
4	Process structure modules	Multilevel-process open-frame or enclosed structures including single or multiple equipment, piping, electrical and instrumentation. These modules may weigh 20 tons to 4,000 tons or more.
5	Dressed towers	Vertical vessels that have been pre-assembled with all the insulation, ladders, platforms, lighting, and instrumentations.
6	Pump modules	Modules that include platform-supported pumps, weather shelters, piping, electrical and instrumentation.
7	Pre-assembled units	Units that are broken down into smaller components such as pre-assembled units due to volume and size constraints in shipping and transporting large complex structures.

Pipe spools are unique in terms of their components such as tee connections and a tube and elbow [121]. They are also unique in shape, material, and finish [122]. Pipe-spool manufacturing starts with cutting the pipes to the required size, fitting and welding (pipe spool shape formation), testing (e.g., hydro testing), and transporting the product to the module yard for assembly [123]. This process represents the first stage in modular construction, shown in Figure 5- 1. The other processes include module assembly (installing the manufactured pipe spools inside a module), and transporting and installing modules on the project site. It is obvious from Figure 5- 1 that the pipe spool is the controlling element in modular construction because although the module size may be standardized, the cost and time required to produce a module may vary significantly. This variation is driven by the fact that each module may be composed of multiple unique pipe spools (in shape, size, and materials) with each acquiring a different number of man-hours to be produced.

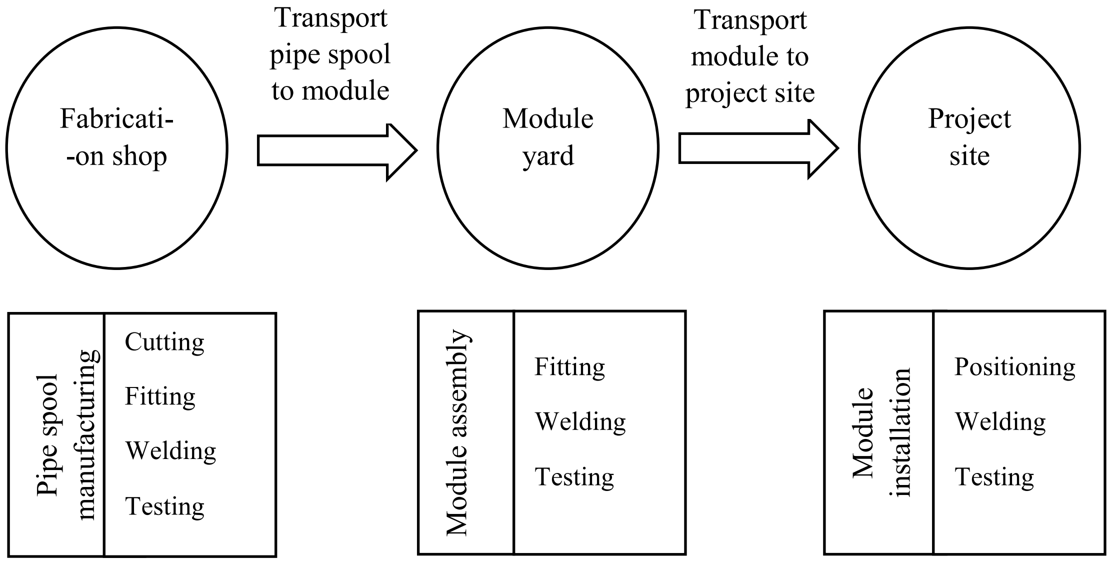


Figure 5- 1 Module production processes in industrial project construction

Also, transporting the pipe spool from the fabrication shop to the module yard creates a transition state in the production process, a transition from a controlled manufacturing environment to an uncontrolled environment. This transition more specifically affects the welding and testing processes. Contractors prefer to perform pipe-spool welding and testing in a controlled environment because the cost associated with these two processes is higher in an un-controlled environment (e.g., project site).

However, welding in a module yard or project site is unavoidable, which results in cost variations from one module to another. Furthermore, producing different sizesmakes it more expensive to transport the pipe stools to the module yard or project site. In the case of contractors employing different fabricators, the contractors prefer to optimize the size of the pipe spools to minimize additional cost due to transportation, off-site welding, and testing; thereafter, the total manufacturing cost of the module is also optimized. However, the question to be answered before proceeding with this discussion is "How pipe-spools are designed, and how is their configuration optimized?

5.2.2 The generation of pipe-spool cut-sheets

In industrial construction, the typical production process starts with the delivery of International Organization for Standardization (ISO) drawings, drafting, material delivery, followed by the rest of processes shown in Figure 5- 1 [124]. These ISO drawings are provided by the client and represent a piping system that includes the pipe section, dimensional properties, transition pieces, in-line instrumentation, and support. These drawings can either represent a pipeline or a pipeline partition. Upon receiving the drawings, the contractors break the drawings into smaller drawings called cut-sheets [125]. These cut-sheets break the pipeline partition illustrated in the ISO drawing into smaller elements called pipe spools. Since clients provide ISO drawings, contractors have little-to-no control over the pipelines partitioning process. However, they have full control over the optimization of the pipe-spooling process (optimizing the generation of pipe-spool cut-sheets). The generation of pipe spool cut-sheets is performed based on different rules and heuristics. The common rules are (1) limiting the spool length to the transportation envelope [125], or (2) limiting the spool volume to the fabrication shop clearance limit so that roll fitting and welding (i.e., main pipe rotating and fitter/welder position is fixed) are maximized and the position fitting and welding (i.e., pipe position is fixed and fitter/welder moving around the main pipe to perform fitting and welding) are minimized [126]. Both rules are performed by fabrication shop personnel, and it is difficult for each rule to reach a degree of consistency while applied in the generation of pipe-spool cut-sheets. This is due to the human factor employed in the process; generally speaking, two fabrication shop personnel may generate different spool-pipe cut-sheets for the same pipeline ISO drawing. Moreover, none of the generated cut-sheets may represent the optimum configuration that serves the described objectives: (1) maximize the pipe-spool size to optimize the transportation cost and generate efficient transportation and logistics strategy, (2) maximize the welding and testing process in a controlled environment (fabrication shop), and (3) minimize welding and testing in an uncontrolled environment (project site). Therefore, it is important to identify the dominant rules used in the generation of the pipe-spools' cut-sheet. The selection of such rules is dependent on the area of study, and since the decision support system in front-end planning in industrial construction is the area of consideration in this study, the selected rule to be investigated and modeled is "limiting the spool length to transportation envelope." This rule identifies an optimization problem in the field of industrial construction which is the "pipe-spooling optimization problem." It is defined in the following section.

5.2.3 The problem definition of pipe spooling

The defined pipe-spooling problem is given a transportation envelope (EV) with a dimensional limit (L, W, H) and a pipeline data structure. This raises the question, what is the optimum configuration of pipe spools that can be generated from the given pipeline data structure while considering the dimensional limitation of the transportation envelope? Figure 5- 2 shows the graphical representation of the proposed pipe-spooling optimization process and includes three major stages: (1) input (the determination of the instance problem), (2) optimization model (the identification of the optimization algorithm suitable to solve the pipe-spooling problem), and (3) the expected optimization output.

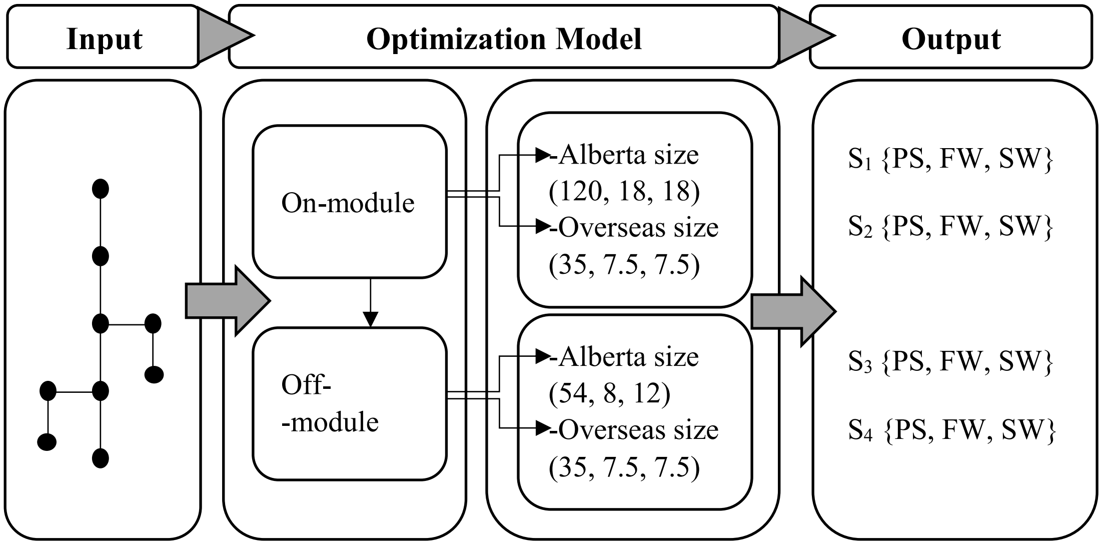


Figure 5- 2 Pipe-spooling optimization process

Starting from the last stage (output), the main contractor requirements from the pipe-spooling optimization problem are: (1) to generate the minimum number of pipe spools for each pipeline(PS), (2) to maximize welding and testing requirements in the fabrication shop (FW), and (3) to minimize welding and testing requirements on the project site (SW). These requirements are considered the optimization variables. The contractor also highlighted the importance of adding a decision weight factor c_k (k= PS, FW, SW) for each variable so that the effect of different cost-time trade-off scenarios can be studied and analysed. Part of these scenarios includes testing the optimum configuration of pipe spools under a different selection of transportation envelopes such as Alberta and overseas sizes, as shown in the optimization model stage in Figure 5- 2. This will make it possible to investigate the transportation and logistics strategies to be applied in case different fabrication companies are involved in the industrial project. These sizes are further classified to two categorizes: On-Module and Off-Module. The main difference between these two categories is that when the piping system in the industrial project is broken down to modules, some of pipeline spools/components may branch out of the module. These are called off-modules spools or components while spools or components contained within the module are called on-module spools or components. Furthermore, off-modules spools or components are not transported to the module assembly yard; rather, they are transported directly to the project site.

The first stage shown in Figure 5- 2 is the input modeling of the instance problem. Pipelines, as mentioned previously, are the problem instances and as per the contractor requirements these instances should include types of pipelines components (*tube, valve,* etc.), their connectivity information (the sequential pattern of the pipeline components), and their physical properties (such as length, diameter, running direction). These properties are important to quantify the required fabrication shop weld (FW), and project site weld (SW) for each pipe-spool generated. For instance, there is no weld required in the case of two flanges connected, but welding is required in the case of two tubes that are supposed to be connected.

5.2.4 Mathematical formulation

As per the problem definition in **5.2.3** the mathematical formulation of the pipe-spooling optimization objective function is defined as:

$$OPS_j = Min\ [S_{ij}] \qquad \text{5-1}$$

Where,

OPS_j = optimum configurations of pipe spools in pipeline j ($j = 1, \dots, n$).

S_{ij} = solution weight value of each pipe spool configuration i ($i = 1, \dots, m$) in pipeline j.

The pipe-spooling optimization objective function provides the optimum configuration of pipe spools from the i expected solutions S_{ij} of pipeline j and the solution incorporates all optimization variables that are (1) the number of generated pipe spools (N), (2) the fabrication shop weld (FW), and (3) the project site weld (SW). Thereby, the mathematical formulation of the solution weight value is defined as:

$$S_{ij} = (c_{PS} \times N) + (c_{FW} \times \sum FW_i) + (c_{SW} \times \sum SW_i) \qquad \text{5-2}$$

where,

c_{PS} = weight factor for the number of generated pipe spools (PS).

c_{FW} = weight factor for the number of fabrication shop welds (FW).

c_{SW} = weight factor for the number of project site welds (SW).

N = the number of generated pipe spools for solution i.

$\sum FW_i$ = the sum of the number of expected fabrication shop welds for solution i.

$\sum SW_i$ = the sum of the number of expected project site welds for solution i.

The number of expected solutions for each pipeline is driven by the possible number of starting points. For example, Figure 5- 3 shows a graphical representation of pipeline ISO fitted to an on-module envelope. The pipe spooling can start from three possible starting components denoted as St_{11}, St_{21}, and St_{31}.

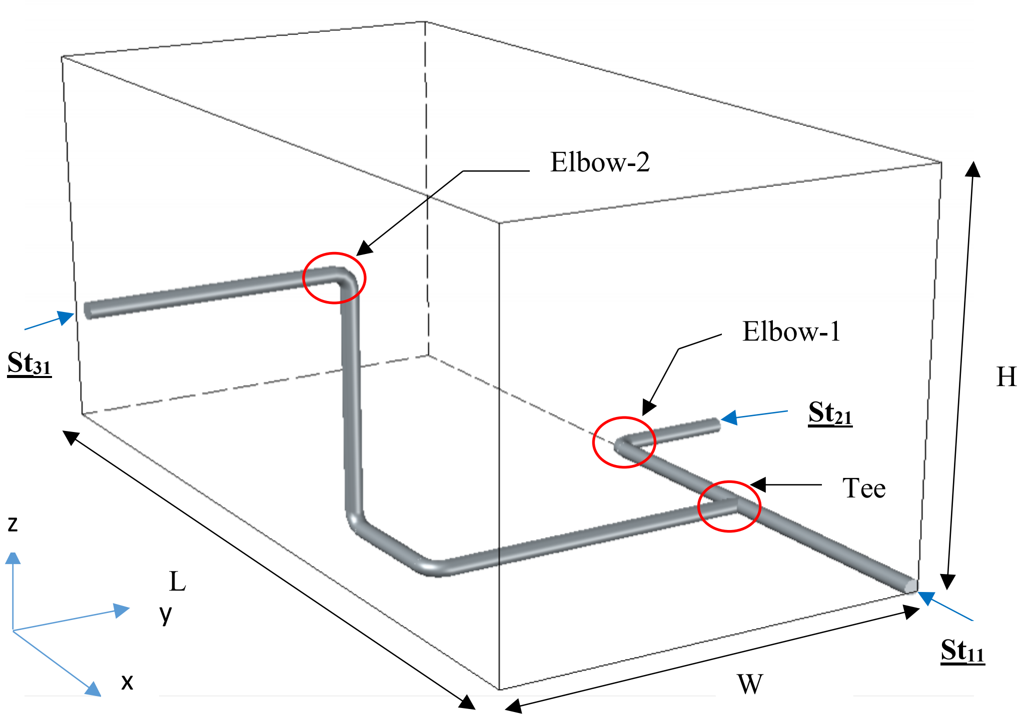


Figure 5- 3 Graphical representation of pipeline ISO and on-module envelope

Each starting component may have a different number of pipe spools because the sequential pattern of pipeline components differs from one starting component to another. Moreover, these components have unique properties such as the type of component and length. These properties, in addition to the unique sequential pattern of pipeline components, control the allowable size of the generated pipe spool. Also, the pipe-spooling process is subject to other constraints along with the modules' dimensional limits, shown in Figure 5- 3. These constraints are explained as follows:

1. If any component, except for components of type tube, extends beyond the envelope boundary $EV\ (L, W, H)$ by ≤ 25 cm, then the component can be considered an on-module spool/component. For example, if *"Èlbow-1"* branch in the y direction, shown in Figure 5- 3, is extending out from the boundary (W) by less than 25 cm, it is considered part of the on-module spool/component. If not, it is considered an off-module spool/component.
2. If a pipeline component of type tube extends beyond the envelope boundary $EV\ (L, W, H)$ by ≥ 25 cm, the component can be cut into two pieces. The component piece inside the module is considered an on-module component and the one outside is considered an off-module component.
3. Any connectivity between components tube*,* tee*,* elbow*,* reducer*,* and flange requires either a fabrication shop weld or a project site weld. However, other connectivity with other components such as valve*,* instrument*, or* closure requires a bolt connection. This constraint limits the calculation of both FW and SW to certain types of components.

In this section, the pipe-spooling problem statement and its mathematical formulation of the optimization objective function have been defined. However, the pipeline data structure is a combinatorial structure, and it is important to identify the heuristic/approximation algorithm that can provide the optimum pipe-spooling solution. Therefore, the following section will identify the class/type of optimization problem and the heuristic/approximation algorithm which can be adopted in pipe-spooling optimization modeling.

5.3 The packing problem, bin-packing, and heuristics overview

The problem definition of pipe-spooling presented in Section 5.2.3 is closely related to packing problems, specifically to container-loading problems. The container-loading problem is defined as packing a set of rectangular-shaped items into a rectangular fixed-shape container [127]. Pisinger [127] defined different types of container-loading packing problems in accordance with their objective function. These packing problems are:

- Strip packing: in this problem, the container has a fixed height and width, and infinite depth. The objective function in this problem is to pack all the items in a way that minimizes the container depth.
- Knapsack loading: in this problem each packed item is associated with profit and the objective function is to choose the set of items to be packed in the container so that each container is loaded with maximum profit.

- Bin-packing: in this problem all containers have a fixed height, width, and depth. The objective function is to pack all the items into a minimum number of containers.
- Multi-container loading: in this problem the shipping containers may have different dimensions and the objective function is to choose a set of containers that minimizes the shipping cost.

Of these packing problems, the bin-packing container-loading problem is more relevant than the pipe-spooling optimization because the optimum pipe-spool configuration is constrained by a transportation envelope (a transportation envelope is equivalent to the container size described in the container loading problem). Also, the packed items in the pipe-spooling process are pipeline components. Although they are of an irregular shape, they can all be represented by rectangular- shaped components with different dimensions.

5.3.1 The bin-packing problem overview

Given n items, each with a different weight w_i, and an infinite number of bins with capacity c, the bin-packing problem is defined as packing all items in a way that the number of bins used is minimized and their capacity usage is maximized [128]. Based on the bin-packing definition, the problem is dependent on two elements: packed items and bins used. These two elements control the dimensionality of the bin-packing problem, imposing different dimensionality features. The one-dimensional bin-packing problem is the classical version that represents both the packed item and the bin size with an integer number. It was initially used to model a range of real-world problems such as packing trucks having a weight limit and assigning station breaks on television [129]. The different application, specifically in the manufacturing industry, highlighted the need to model a higher-dimensional bin-packing problem. For, example the steel, glass, and wood industries imposed the need for an optimization model to cut these items to a specific area so that produced waste would be minimized [130]; thus, the two-dimensional bin-packing was one of the offered optimization modeling approaches [131]. Also, a three-dimensional bin-packing problem was presented to solve a complex problem such as container loadings in the field of transportation and logistics. However, all bin-packing problem, regardless of their dimensionality differences, are considered NP-hard problems [132] [133] ,which normally require heuristic methods to generate a solution in a reasonable time frame [134] [135].

5.3.2 The bin-packing heuristics

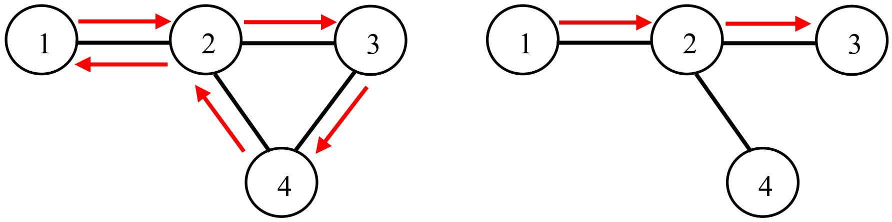
Heuristic (or approximation algorithms) is defined as “a procedure to reduce search in problem-solving activities, or a means to obtain acceptable solutions within a limited computing time” [136]. Four bin-packing heuristics were first presented and evaluated by Garey et al. [137] to generate good placement of bins: (1) first fit, (2) best fit, (3) first fit decreasing, and (4) best fit decreasing. The differences between the first and the last two heuristics are that the earlier heuristics start packing items in an increasing order (small items first), while the later ones start packing items in decreasing order (large items first). However, the common characteristics are that the previously processed bins can be considered to pack the currently processed item. Some applications prevent the consideration of any processed bin. Therefore, another bin-packing heuristic called “next fit” was proposed. The packing process of the next fit heuristic starts packing items in a sequential order, regardless of their weight or size, until the bin capacity is achieved. After that the bin is closed with no future consideration. These are the classical types of heuristics, applied on both on-line (packing the item with no knowledge about the successive items), and off-line (information about all items is available before packing) approximation algorithm classes, and applied on one- or higher-dimensional packing problems [138]. The described heuristics (or the approximation algorithms) consider packing one item at a time; in other words, the described heuristics represent a one-directional instances problem with no multidimensional connectivity relationships between the packed items. However, if modeling a pipe-spooling process as a three-dimensional bin-packing problem, two questions should be first answered:

1. How to consider the combinatorial structure of the pipeline problem instances for modeling packings?
2. How to adopt the pipe-spooling process constraints described in Section 5.2.4 in the bin-packing problem definition?

5.3.3 Pipe spooling as a three-dimensional bin-packing problem

Pipeline data structure, as mentioned in the previous chapter, is treated as a tree structure. It contains vertices/nodes that describe the pipeline component and edges that describe the connectivity relationships between pipeline components. Furthermore, at the design stage of pipeline systems, the pipeline system is assigned to either one or multiple modules (based on industrial practices) and each pipeline component assigned to a pipeline partition is tagged with a coordinate to identify its location in reality. In addition, the pipe-spooling process starts by first identifying the possible starting point in the pipeline (refer to St_{11},St_{21} and St_{31} in Figure 5- 3) and thereafter follows the running direction of the second component. Therefore, the pipeline instance problem can be treated as a directed graph $G = (V, E)$; where V is the graph vertices and E is the graph edges. This type of data structure creates a challenge in constructing optimal solutions for packing problem; however, Fekete and Schepers [139] presented an approach to construct feasible packing using a graph-theoretic characterization of the relative position of the packed items. The same approach is to be applied to reflect the pipe-spooling optimization problem into a three-dimensional bin-packing problem.

The approach starts by first defining the input data for the three-dimensional bin-packing problem; the input is a set of components V that form the pipeline partition (more specifically, components that form the pipeline ISO shown in Figure 5- 3) and a size vector w. The size vector w represents the minimum and maximum point coordinates of the box that contains the component. The other input is the size of the container W which is equivalent to the module sizes illustrated in Figure 5- 2 and to EV envelope shown in Figure 5- 3. Fekete and Schepers [139] used a graph characteristic called an induced subgraph to construct a feasible solution. The induced graph represents the set of vertices associated with their edges that can be enclosed within the container. However, in the case of pipe spooling, using only an induced graph may not produce the feasible solution because spools generated from pipeline ISO may take the form of either an induced graph or a subgraph. The main difference between these two graphs is that the induced graph has a set of edges, E, that connect all vertices. In other words, any vertex can return to itself by passing through other vertices. This condition does not necessary apply in the subgraph. As an illustration example, Figure 5- 4 shows both the induced graph and subgraph. It can be seen that vertex 1 in the induced graph can return to itself by passing through vertices 2, 3, and 4. However, it is not possible for the same vertex to return to itself in the subgraph because of the missing edge between vertices 3 and 4.



(a) Induced Graph

(b) Subgraph

Figure 5- 4 Graphical example of (a) induced graph, and (b) subgraph

Therefore, we consider that the feasible solution is a set of pipe spools, PS_{i},, each presented in the form of an induced graph or subgraph and denoted as $ps_j = (C_{nj}, E[C_{nj}])$, where C_{nj} is a set of components, n, forming pipe spool j ($C_{nj} \in V$) and $E[C_{nj}]$ is the edges connecting the pipe spool j components. The pipe spool ps_j is called feasible only if $\sum_{C_{nj}\in ps_j} w(C_{nj}) \leq \ W$, and the arrangement of pipe-spool components in the three-dimensional packing should generally satisfy the following constraints [140]:

1. Orthogonality: the pipe spool component should be parallel to the container face.

2. Closedness: no pipe spool component should exceed the container boundaries.
3. Disjointness: no two pipe-spool components may overlap.
4. Fixed orientations: no component should be rotated once the packing starts.

The above description of the use of graph-theoretic characterization answers the question of how to consider the combinatorial structure of pipeline problem instances for modeling packings. However, the arrangement constraint "closedness" in three-dimensional packing contradicts the first and the second pipe-spooling process constraints described in section 5.2.4. The first pipe-spooling process constraint is overcome by adding a marginal level of 25 cm to the container size W while running a heuristic or approximation algorithm,. The second pipe-spooling constraint, where a pipe-spool component of type tube can be cut into two pieces to maximize the component's inclusion in the container, can be maintained by employing the item fragmentation feature of the bin-packing problem. The bin-packing problem that applies item fragmentation is called the Bin-Packing Problem with Item Fragmentation (BPPIF). It allows the packed item (a_i) to be fragmented to two pieces $(a_{i1} + a_{i2})$ to minimize the number of bins in the packing process [141]. Based on the above discussion, this section concludes with the following: (1) a graph-theoretic characterization is used to reflect the pipe-spooling problem as a bin-packing problem, and (2) the item fragmentation feature of the bin-packing problem is employed to adopt the pipe-spooling constraints.

5.3.4 The branch-and-bound heuristic

As explained in the previous section, the pipeline-instance problem is a combinatorial type of structure. The instance problem is treated as a directed graph $G = (V, E)$ and the bin-packing process starts from pre-determined vertices in graph G. The feasible pip- spool solution ps_j is either an induced or subgraph from G that is constrained by a dimensional limit or size. Therefore, the obvious heuristic or approximation algorithm that can be applied in this case is the classical branch-and-bound heuristic. The branch-and-bound heuristic is "an intelligently structured search of the space of all feasible solutions. Most commonly, the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets" [142]. The branch-and-bound heuristic was first introduced by Land and Doig [143] and is most often used in constrained optimization problems [144]. The heuristic first identifies the number of possible solutions. Figure 5- 5 provides an example in the form of an illustration; the figure shows a tree representation of the pipeline data structure. The number of possible solutions is determined by the number of possible components that the packing process starts with. In this example, four possible solutions can be identified by starting with components 1, 3, 7, and 8.

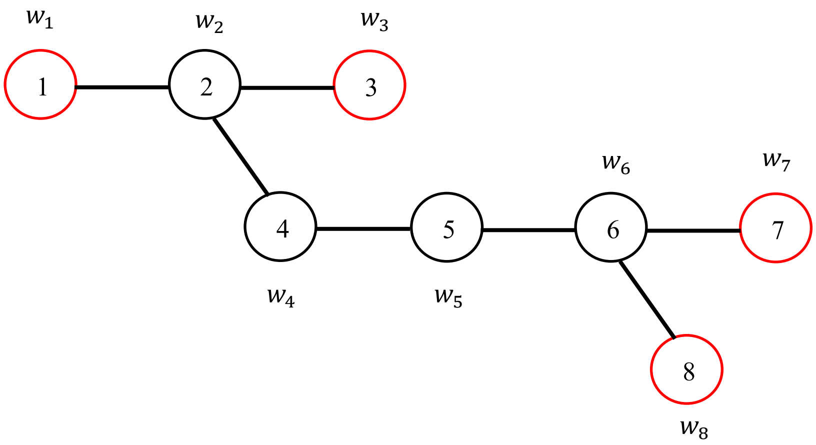


Figure 5- 5 Tree representation of pipeline- problem instance

As mentioned previously, the solution is defined as a set of pipe spools, PS_i . Each pipe spool ps_j is called feasible only if it satisfies the following condition:

$$\sum_{C_{nj} \in ps_j} w(C_{nj}) \leq \ W \text{ 5-3}$$

where,

$w(C_{nj})$ = the size vector of component C_n that belongs to pipe spool ps_j. The size vector contains the location of the minimum $(x_{min}, y_{min}, z_{min})$ and maximum $(x_{max}, y_{max}, z_{max})$ points of the component. In short, it is a cuboidal envelope that contains the component.

W = the size vector of the container/module envelope. It also contains the location of the minimum (0, 0, 0) and maximum $(x_{max}, y_{max}, z_{max})$ points of the container.

Starting from Component 1, shown in Figure 5- 5, the branch-and-bound heuristic first tests the component to determine whether it satisfies condition (3) or not. If $w_1(C_1)$ $\leq W$, then Component $C_1$is packed in the container/module envelope and considered as the first component of pipe spool ps_1. The packing arrangement, on the other hand, follows the “bottom-left” procedure, such that the first component is to be located at the lower bound of the container. The branch-and-bound then moves to component (C_2) and applies the same test as in component (C_1). Assuming that the second component passes the test, it will be added to ps_1. Component (C_2) branches to components (C_3) and (C_4); therefore the branch-and-bound will test the condition under two scenarios. The first scenario, ranked first, is to pack both components if they satisfy $w_3(C_3) + w_4(C_4) \leq W$. Otherwise, the second scenario is applied which packs the component that has the longest span (e.g. component C_4). If it is assumed that the first scenario can be applied, then both components 3 and 4 are packed in the container and added to ps_1. The same packing procedure is applied until no possible component can be packed. For example, if component $w_5(C_5) \geq W$, then the applicability of packing C_5 is checked in accordance with the first and the second constraints in the pipe-spooling process described in Section 5.2.4. This step is performed to check whether C_5 can be fragmented or not. If C_5 cannot be fragmented, then the packing process is ended for the first pipe spool ps_1 and the second packing process starts from C_5 to generate the second pipe spool ps_2. Once the branch-and bound heuristic that started from C_1 covers all components, the set of generated pipe spools is called PS_1.

The generated set of pipe spools, PS_1, works as the foundation in calculating the solution weight value S_{ij} described in equation (5-2) in Section 5.2.4 Refer to Figure 5- 6 for a graphical illustration on how the pipe spooling solution weight value S_{ij} is calculated.

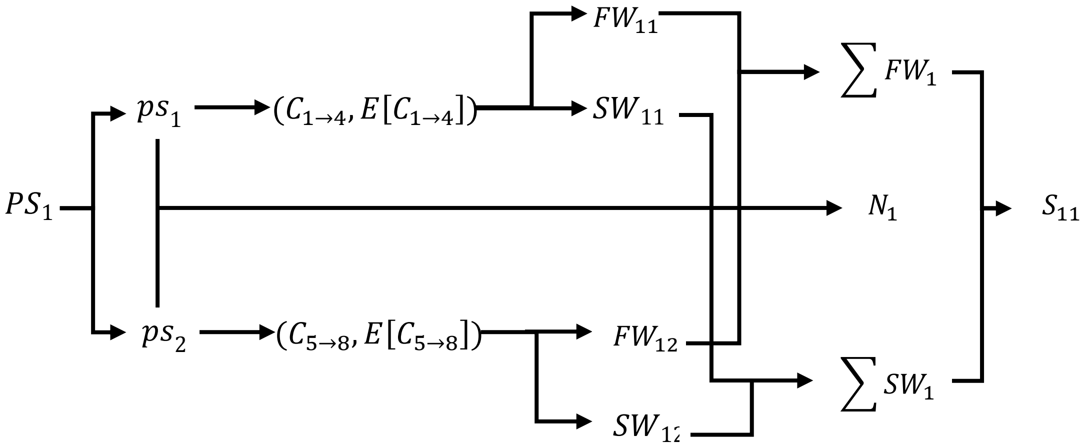


Figure 5- 6 Calculation flow of the solution weight value S_{ij}

The generated set of pipe spools, PS_1, from the branch-and-bound heuristic search started from C_1 resulted in providing two optimum pipe spools, ps_1 and ps_2, as shown in Figure 5- 6. The weight value of this solution, denoted as S_{11}, is calculated by first determining the three variables of equation (2); $N_1, \sum FW_1$ and $\sum SW_1$. The number of pipe spools, N_1, is equivalent to the number of pipe spools in set PS_1 , which is 2. However, the other variables, $\sum FW_1$ and $\sum SW_1$, are determined by relating to the generated induced graph or subgraph of both pipe spools $(ps_{1=}(C_{1\to4}, E[C_{1\to4}]), ps_{2=}(C_{5\to8}, E[C_{5\to8}]))$. Both graphs have the components' properties and their connectivity relationship. Therefore, using this information makes it possible to determine the required fabrication shop (FW_{11}, and FW_{12}) and project site weld (SW_{11}, and SW_{12}), after which S_{11} can be calculated.

5.4 The generation of pipeline problem instances using the industrial pipelines' data generator

In the previous section, a pipe-spooling process was described and modeled as a three-dimensional bin-packing problem. A branch-and-bound heuristic is proposed and described in detail to construct a feasible packing solution, and the remaining is to test the computational efficiency of the proposed heuristic using a pipeline problem instances test set. The pipeline problem instances can be extracted from either the real industrial project or can be generated using a data generator. The first option is normally challenging because of the high cost of data collection, cleaning, and preparation. Randomly generating pipeline problem instances from a data generator provides more flexibility in testing and improving the studied approximation algorithms. Therefore, the industrial pipelines' data generator described in the previous chapter will be used to randomly generate a data test set of different pipeline problem instances.

The validation processes applied on the generated pipeline data structure proved the ability of the pipeline data generator to provide characteristic behavior of pipeline components that was approximately similar to the behavior in reality. The generated pipelines are composed of a set of components, each defined by its type (pcomponent, instrument, valve, flange, tube, elbow, tee, reducer, coupling, closure, cap, ftube, and fblind) and its connectivity relationships with the other components. In addition to these properties, the packing process of the pipe spool components requires the size $w_i\ ([x_{min}, y_{min}, z_{min}], [(x_{max}, y_{max}, z_{max}])$ of each component. The size w_i of each component represents the boundary box that contains the component. In order to allocate the minimum and maximum points of the component boundary box, three additional properties should be added to the pipeline components:(1) component's length, (2) component's diameter, and (3) component's running direction. To accommodate these properties two additional layers of component properties will be added to the pipeline generator. The first layer will generate the length and diameter for each component, and the second layer will generate a running direction for each component. These two layers will make it possible to generate a data test set of pipeline problem instances that can be used to test the computational efficiency of the proposed heuristic. A detailed description of each layer is given in the following subsections.

5.4.1 Layer I: The generation of component lengths and diameters

In this layer, the same industrial pipeline data used in the previous chapter to construct the pipeline data generator is used to extract the length and diameter properties.

5.4.1.1 The generation of component lengths

Each pipeline may be formed from a different combination of pipeline components (refer to Table 5-3), and each component may exert different length properties. Therefore, each component length from the real pipeline data is fitted to a theoretical probability distribution function that best approximates its behavior. Table 5-3 shows the theoretical probability distribution functions for each pipeline component. Most of the components either have a different distributional behavior when compared with others, or they have a similar distributional behavior with different approximation parameters. Unlike other components, instrument*,* valve*,* flange*,* and closure*,* were given a value of 0 because the real pipelines data showed that these components do not have any length properties.

Table 5- 3 Probability distribution functions for components' lengths (mm)

Components Type	Probability distribution function
Pcomponent	Wald (726.79, 1072.6)
Instrument	0
Valve	0
Flang	0
Tube	Wald (2323.2, 489.58)
Elbow	Laplace (339.860,0.00111)
Tee	Uniform (57,432)
Reducer	Gamma (3.4645, 51.749)
Coupling	Beta (0.02229, 0.42713)
Closure	0
Cap	Lognormal (0.8399,4.4939)
Ftube	Uniform (75.394,76.985)
Fblind	Uniform (122, 777.19)

The industrial pipeline data generator was updated so that each generated pipeline component would be tagged with its expected length. For the purpose of validating the overall pipeline length characteristics, 1000 pipelines were randomly generated using the industrial pipeline data generator and compared with 1000 pipelines from the real pipeline data set. The applied validation process does not evaluate each type of pipeline component individually; rather, the set of all components' lengths in the entire pipeline population is considered in the validation process.

Table 5-4 shows the comparison between the generated and original components' lengths using different statistical measures. Both data sets have a mean of component length approximately equal to one meter. The generated components' lengths have a higher standard deviation compared to the real components' lengths. In general, the generated components' lengths provided higher values (first-quartile, median, third-quartile, and maximum values) than those in the real data set. The generated median has a higher value when compared to the median of the original data (almost twice the size of the real median). The same applies to the third-quartile and maximum values since they are 20-25% higher than the normal values.

The minimum length in both data sets is 0. It is not realistic for a component to have a length of 0. An investigation of this issue showed that the length of some components, such as closure and the flange, was given a value of 0 in the original pipeline data set. This case shows that the original pipeline data missed some of components' properties. Therefore, it is assumed in the generation process of components' lengths, that any component with a length value of 0 was substituted with a length value of 350 mm. This value was used to produce realistic industrial pipeline data.

Table 5- 4 Statistical measures' results for the real and generated components' lengths

Statistical measure	Real components' length	Generated components' length
Mean (mm)	967.8	1125
SE mean	19.6	20.4
Standard deviation	3309.3	3454
Minimum (mm)	0	0
Q1 (mm)	0	134
Median (mm)	158.5	339
Q3 (mm)	508	612
Maximum (mm)	86299.1	107396

5.4.1.2 The generation of components' diameters

The design of pipeline systems is based on achieving functionality of the pipeline facilities. As previously described, the pipeline is a collection of different types of components, and when analyzing the 1000 real pipelines, it was found that components of type tube are dominant in the whole real dataset. Moreover, any component connected to component of type tube share the same diameter, except for *"Reduced"* where the reduction of flow diameter exists at this particular component. The industrial pipeline data generator should include a pipeline flow diameter so that a pipeline problem instance can be generated. It is assumed that all components will share the same flow diameter that is controlled by component tube, and the value of the flow diameter is to be sampled from a theoretical probability distribution function that best approximates the distributional behavior of the tube's diameter. Since the reduction in flow diameter is controlled by the reducer, a theoretical probability distribution function for the reducer's diameter will also be determined. Table 5- 5 shows the probability distribution functions for both the tube and reducer. Both components have the same distributional behavior.

Table 5- 5 Probability distribution functions for tube's and reducer's diameters

Component	Probability distribution function
Tube	Burr (0.7444, 1.9, 135.11)
Reducer	Burr (0.76059, 2.858, 124.3)

The industrial pipelines' data generator was updated to include the flow diameter in the pipeline data structure. When generating a pipeline component, the generator first samples a flow diameter value from the tube's probability distribution function shown in Table 5- 5. The value is then assigned to all following components. In the case of generating a reducer, the flow diameter will be reduced or increased based on a value generated from the probability distribution function of component reducer. The validation process in this sub-section will be based on the same approach described in Section 5.4.1.1, "The generation of components' lengths.". The collection of generated flow diameters from 1000 generated pipelines will be compared to a collection of flow diameters from 1000 real pipelines. Different statistical measures applied in the comparison and their results are shown in Table 5- 6.

Table 5- 6 Statistical measures' results for the real and generated components' diameters

Statistical measure	Real Tubes' Diameter	Generated Tubes' Diameter
Mean (mm)	193.56	170.87
SE mean	7.65	7.68
Standard deviation	198.2	198.85
Minimum (mm)	1.39	2.84
Q1 (mm)	72	63.28
Median (mm)	114.45	114.84
Q3 (mm)	220	192.61
Maximum (mm)	1453.94	1793.32

Table 5- 6 shows that both the real and generated pipeline datasets have similar statistics. However, it is worth mentioning that the average minimum flow diameter is found to be 2.115 mm ($\frac{1.39+2.84}{2}$), which is not an acceptable pipe diameter. For that reason, the generation of the flow diameter is conditioned to have a minimum value equivalent to the first-quartile value of 72 mm.

5.4.2 Layer II: The generation of components' running direction

In Layer I, two components' properties, length and diameter, were added in the generation process of the pipeline data structure. These two properties are used to define the minimum and maximum point locations of each component envelope. These points are located in a three-coordinates' space. To identify the required coordinates, each generated component should have an expected running direction. The industrial pipeline data generator is updated so that each component is associated with its running direction. The running direction of any pipeline component is driven by the running direction of the previous component. For example, if the first component is found to be a tube and is flowing from direction x, then the coming component will the follow the same direction except for components of types tee or elbow. These two components alter the component running direction to a different axis. Component elbow alters the running direction by moving it to a different single direction (refer to component number 4006 in Table 5- 7), and component tee branches the pipeline into either one or two different directions. Table 5- 7 shows a sample of a pipeline data set generated using the industrial pipelines' data generator. Only a positive x, y, z running direction (RD) is considered in the pipeline generation process. Meanwhile, in the real pipelines' data structure, components' running directions may be altered to either a positive or negative directional axis. This case is not integrated into the current pipeline data generator. However it should be considered in future research.

Table 5- 7 Pipeline dataset generated using the updated industrial pipeline generator

Pipeline	Branch	C#*	Connected to	Type	D*	L*	RD*
1	Main_Line	4001	4002	Instrument	314	0	x
1	Main_Line	4002	4001	Flange	314	0	x
1	Main_Line	4003	4002	Tube	314	2651	x
1	Main_Line	4004	4003	Elbow	314	339	y
1	Main_Line	4005	4004	Tube	314	572	y
1	Main_Line	4006	4005	Elbow	314	339	z
1	Main_Line	4007	4006	Tube	314	2614	z
1	Main_Line	4008	4007	Elbow	314	339	x
1	Main_Line	4009	4008	Tube	314	221	x
1	Main_Line	4010	4009	Pcomponent	314	898	x
1	Main_Line	4011	4010	Reducer	54	266	x
1	Main_Line	4012	4011	Tube	54	121	x
1	Main_Line	4013	4012	Elbow	54	339	y
1	Main_Line	4014	4013	Tube	54	1519	y

***(C#: Component number, D: Diameter (mm), L: Length(mm), RD: Running direction)**

5.5 Computational experiment-Testing the computational performance of the bin-packing algorithm

The branch-and-bound heuristic was proposed to approximate the pipe-spooling solution in the three-dimensional bin-packing modeling of the pipe-spooling process. It was implemented in C# and tested on a DELL Xeon 3.5 GHz processor. The pipelines' data test sets were randomly generated using the industrial pipelines' data generator. One-thousand pipelines with a total number of components equal to 28602 were randomly generated to test the performance of the applied heuristic. The popular performance indicators used to evaluate the efficiency of algorithms are (1) CPU processing/run time, (2) operations' count, (3) number of iterations, (4) storage requirements, (5) robustness, (6) accuracy, and (7) reliability [145]. These indicators are normally used in the context of comparing two different algorithms solving a certain problem. However, a processing-time indicator will be used in this computational experiment to report the differences in the performances of the proposed heuristic when applied on two different container sizes: Alberta and overseas sizes. The results are to be used as benchmarks when the defined pipe-spooling problem is solved using a different solution's algorithm. The CPU run times to solve 1000 pipelines in two different container configurations is shown in Figure 5- 7. A detailed result for each pipeline instance problem is given in Appendix H.

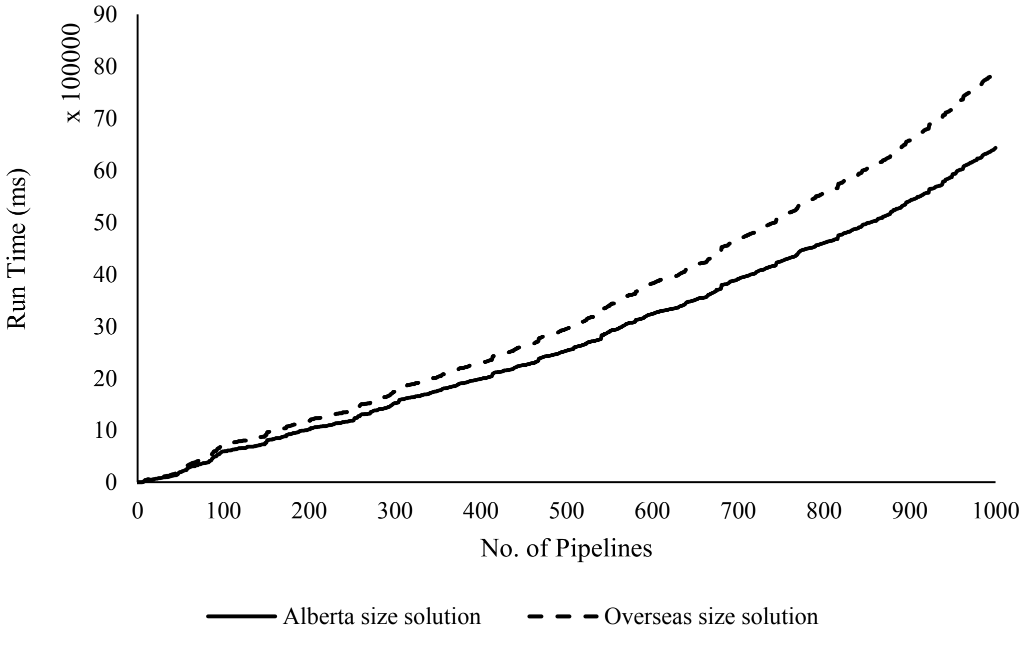


Figure 5- 7 Pipe-spooling solution's run time with respect to the number of pipeline instances problems

The solution run-time results are summarized in Table 5- 8. Table 5- 8 illustrates the average solution's CPU run time that is expected to solve the instance problems. The average solution's CPU run time is associated with different instance problem sizes. The problem instance size is measured in terms of the number of components in a pipeline. Also, the average solution's CPU run time is associated with the average number of pipe spools generated.

Table 5- 8 Pipeline solutions results

No. of pipelines components	Alberta Size		Overseas Size	
	Average No. of Spools	Average solution's CPU run time (ms)	Average no. of spools	Average solution's CPU run time (ms)
<=10	2	1186.13	2	1466.29
11-20	2	2804.44	4	3575.53
21-30	3	5008.36	6	6349.72
31-40	4	7249.30	9	9247.07
41-50	6	10527.44	11	13199.69
51-60	6	13644.07	12	16035.77
61-70	7	14139.92	14	16802.92
71-80	9	17113.25	17	19520.13
81-90	9	24240.39	19	28523.33
91-100	9	23424.17	24	29984.17
101-110	11	33520.00	23	38008.40
111-120	14	41015.14	27	48196.00
121-130	14	38085.20	28	44562.80
131-140	19	59687.00	36	74076.00
141-150	16	56819.33	36	66339.33
>150	26	60093.00	46	69625.75

5.6 Conclusion

An application of an industrial pipeline data generator was presented in the context of experimental analysis of optimization of an algorithm's efficiency. The pipe-spooling optimization problem identification, definition, and mathematical model were described in detail. The instance problem attributes of the pipe-spooling problem were identified, and the industrial pipelines' data generator was updated accordingly to integrate the additional attributes. A test set of 1000 randomly generated pipelines' instance problems was generated using the pipeline data generator to test the efficiency of the proposed branch-and-bound heuristic in approximating the pipe-spooling solution. The solution approximation was tested in configurations of two different envelope sizes. The heuristic efficiency in terms of CPU run-time performance was reported as performance benchmark results.

Chapter 6

Conclusion

6.1 Conclusion

In construction engineering and management research, modeling data is a vital process to capture the variation of the construction systems' behavior. Construction-related data varies in terms of structure and dependencies between variables within and across each sample. This research was designed to investigate mathematical techniques that are capable of randomly generating data sets while preserving the relationships and the dependencies embedded within the generated data sets variables. It was also intended to develop data generators and illustrate the application of the developed generators in the field of construction research.

Two different types of data with different degrees of complexity were selected in this research; the first is weather variables and the second is pipeline data structure. Two different modeling approaches were used to model these data types: a bootstrapping technique was used to generate weather variables and a Markov chain model was used to randomly generate the industrial pipeline data structure.

In Chapter 2, a non-parametric weather generation approach in the form of a bootstrapping technique was proposed to randomly generate weather variables and develop a weather data generator. The generator's performance was evaluated against a parametric weather generator constructed in the field of modeling construction operations. It was found that the non-parametric approach performed in a way similar to that of the parametric approach. The parametric and non-parametric weather generators were applied in two different weather-sensitive construction models: the first estimated the temperature and wind speed effect on construction labors and the second estimated temperature effects on tower crane operation. Two testing scenarios were applied in both models: the first considered the expected weather variables every day and the second considered the weather variables expected within the eight working hours. The results showed that the non-parametric weather generator outperformed the parametric weather generator when a specified construction period (e.g., an eight-hour window in a day) was considered. The parametric weather generator provided a better result when no consideration of a working period was applied.

In Chapter 3, the non-parametric weather generator was applied to model earthmoving operations in oil sand mining. The weather generator was used to provide different weather scenarios for testing the effect of temperature on truck and excavator breakdowns and repair durations. It was found that 7% to 13.3% of truck operational time contributed to breakdown repair duration, and 8.5% to 12% of excavator operational time contributed to the same event.

In Chapter 4, a Markov chain model was presented to generate industrial pipeline tree structures. The Markov chain model used a transition matrix to generate a sequence of pipeline components. The matrix used state periodicity (a Markov chain property) to regulate the reproduction of the pipeline components. In the validation section, the generated pipelines were converted to feature vectors and a three-stage validation process was applied. The process included (1) evaluation of the number of components and correlation analysis, (2) a clustering-based model validation, and (3) model validation using distances between feature vectors. The Markov chain model was able to generate a collection of pipeline components similar to those found in real pipelines. It was also found that the Markov chain model preserved the correlation between pipelines' components. When comparing the topological structure of the generated pipelines to the original industrial pipelines using a density-based clustering and histogram intersection of the similarity distance between pipelines feature vectors, some discrepancies were found. The majority of generated pipelines (89%) shared characteristics with 85.5% of the original pipelines. No similarities were found between the remaining generated pipelines (11%) and the remaining 14.5% of the original pipelines.

In Chapter 5, the application of the industrial pipelines' data generator was demonstrated. Pipeline instance problems were generated to test the computational efficiency of an optimization solution for the pipe-spooling process. The pipe-spooling problem was intended to identify the optimum configurations of pipe spools that can be generated from a certain pipeline. The industrial pipelines' data generator was updated to integrate the pipelines' components' properties required by the optimization problem. The length, diameter and running direction of each component were added in the generation process of the industrial pipelines' data structure. Including such components' properties provided a good foundation for generating a realistic pipelines instance problem. A dataset containing 1000 industrial pipelines was generated, structured, and used to test the efficiency of a branch-and-bound heuristic applied to approximate the pipe-spooling solution. The efficiency in terms of CPU run-time performance was reported as a benchmark performance result.

6.2 Research contributions

6.2.1 The academic research contributions:

The main academic research contributions are:

1. The study presented a simplified weather generation approach in the form of a bootstrapping technique to generate correlated weather variables.
2. The study showed that a Markov chain model can adopt the heterogeneity associated with pipeline components' properties and can generate industrial pipelines' tree structures.
3. The study presented a three-stage validation methodology to generate a set of weather variables and an industrial pipelines' data structure. It also demonstrated the use of different statistical measures to validate the assumptions used in each model and the generated outputs from each model.
4. The study presented the advantage of using a feature vector concept in converting an overall system into a manageable vector while at the same time

preserving the uniqueness associated with the component properties and their topological structures.

6.2.2 The industrial research contributions:

This research also presented indirect industrial contributions. These contributions are:

1. The study presented three weather-sensitive construction models: the first estimates the temperature and wind speed effects on construction labour, the second estimates the temperature effect on tower cranes, and the third estimates the temperature effect on breakdown and repair durations. These models represent applications in simulation that utilize the developed weather generator and demonstrate its potential benefits to the industry.
2. The study demonstrated the potential use of the industrial pipelines data generator for testing module optimization algorithm. It was used to generate a set of industrial pipelines instance problems with characteristics similar to those found in reality. The generator can also be very beneficial for modeling and simulation fabrication operations. It can generate a vast range of unique industrial pipelines to assess the performance of pipelines fabrication processes.

6.3 Limitations and future research

This limitations of and areas of improvement in this research are:

1. In the non-parametric weather generator, the bootstrapping technique relies on the size of data available. Hence the size of the available data controls the variation in the generated samples. For example, if the performance of a

construction operation is studied with respect to changes in weather conditions in a certain month and the size of historical records is 40 years, then the bootstrapping technique will sample from 40 different months only. This raises an issue related to the limited range of values the non-parametric weather generator is generating for the weather variables. Furthermore, the bootstrapping technique generates a historical weather data with no forecasting abilities.

2. In the application of the non-parametric weather generator in modelling construction operation, the weather generator was used to analyze the temperature effects on trucks' and excavators' breakdown and repair durations. The simulation model was built using industrial practitioners' feedback to identify its inputs and outputs. The limitation in this part is related to the validation of the model's output. A comparison with actual operation data was not conducted. Although this part was motivated to illustrate the application of the non-parametric weather generator in modelling construction operation, a future research can be performed to upgrade the simulation model and validated using actual data.
3. The generation process of the industrial pipelines' data assumes that the pipelines are branching in the positive directions of x, y, and z axes only; however, in reality, the branching process may also take place in negative directions. Therefore, the future update of the industrial pipelines' data generator should be formulated by integrating all possible running directions which pipelines' components may experience in reality. This future update can

be performed by studying two major pipelines' components: *"Elbow"* and *"Tee."* These components are responsible for altering the pipelines' running directions. Successfully integrating all possible running directions can provide more realistic physical properties of industrial pipelines.

4. The industrial pipelines' data generator was developed to provide researchers with a realistic industrial pipelines' data structure. It uses the number of pipelines as an input and provides a detailed pipeline data structure including, as an output, components' connectivity and components' physical properties. The limitation to this part is that the industrial pipelines' generator relies on a probability distribution function derived from original pipeline data to generate the expected number of components for the pipeline's first branch. In the future, the industrial pipelines' data generator can be expanded by adding flexibility to generate pipelines based on a certain number of components. This update will support the area of pipeline optimization problems. It will make it possible to test the computational efficiency of optimization algorithms under a different number of components' scenarios. Furthermore, it will make it possible to create different data test sets of benchmark pipelines' instance problems. Researchers can use these test sets to test new methods to solve the pipe-spooling optimization problem.
5. The applied validation process in both the generation of weather and industrial pipelines data used the same data sets for both modelling and validation. This practice limits the verification whether both generators can provide a realistic

data. Therefore, it is recommended to use different data set for validation purposes.

6. The application of the industrial pipelines' data generator has been illustrated in the field of pipe-spooling in industrial projects. The pipe-spooling optimization problem was clearly defined, as were its instance problem data structure and attributes. Another application of the industrial pipelines' data generator that can be pursued in the future is modules' optimization in industrial projects. As described in Chapter 5, the industrial construction project is broken down into small entities of different sizes, called modules. Future investigations can answer the following questions related to these entities:

    - How are industrial construction projects broken into modules?
    - Are the number of modules and their configurations optimized?
    - If modules are not optimized, is it possible to formulate and solve the modules' optimization problem?
    - What is the instance problem to be used in the modules' optimization problem?

    The industrial pipelines' data generator can be used in this area of study to generate a set of pipeline case studies. These pipeline case studies can be used to conduct in-house experiments, analysis, and testing before applying any proposed solution to the modules' optimization problem in large-scale real projects. Furthermore, the time and cost associated with data collection and preparation can be minimized, which will allow more time for the modelling stage of industrial construction research.

References

[1] D. J. Papageorgiou, G. L. Nemhauser, J. Sokol, M.-S. Cheon and A. B. Keha, "MIRPLib – A library of maritime inventory routing problem instances: Survey, core model, and benchmark results," *European Journal of Operational Research,* vol. 235, no. 2, pp. 350-366, 2014.

[2] A. Otto, C. Otto and A. Scholl, "Systamatic data generation and test design for solution algorithms on the example of SALBPGen for Assembly line balancing," *European Journal of Operational Research,* vol. 228, no. 1, pp. 33-45, 2013.

[3] D. R. Jeske, B. Samadi, P. J. Lin, L. Ye, S. Cox, R. Xiao, T. Younglove, M. Ly, Holt Douglas and R. Rich, "Generation of synthetic data sets for evaluating the accuracy of knowledge discovery systems," in *International conference on Knowledge discovery in data mining*, New York, 2005.

[4] W. J. Trypula, "Building simulation models without data," in *International Conference of Systems, Man and Cybernetics*, 1994.

[5] T. Perera and K. Liyanage, "Methodology for rapid identication and collection of input data in the simulation of manufacturing systems," *Simulation Practice and Theory,* pp. 645-656, 2000.

[6] B. L. Nelson and M. Yamnitsky, "Input modeling tools for complex problems," in *Winter Simulation Conference*, 1998.

[7] S. M. AbouRizk, D. W. Halpin and J. R. Wilson, "Fitting beta distributions based on sample data," *Journal of Construction Engineering and Management,* pp. 288-305, 1994.

[8] R. J. Wales, Incorporating weather effects in project simulation, Edmonton: University of Alberta, 1994.

[9] C. W. Richardson, "Stochastic simulation of daily precipitation, temperature, and solar radiation," *Water Resources Research,* vol. 17, no. 1, pp. 182-190, 1981.

[10] D. Hu and Y. Mohamed, "Pipe spool fabrication sequencing by automated planning," in *Construction Research Congress*, 2012.

[11] P. Wang, Y. Mohamed, S. M. AbouRizk and T. A. Rawa, "Flow production of pipe spool fabrication: Simulation to support implementation of lean technique," *Journal of Construction Engineering and Management,* vol. 135, no. 10, pp. 1027-1038, 2009.

[12] D. Hu and Y. Mohamed, "A Dynamic programming solution to automate fabrication sequencing of industrial construction components," *Automation in Construction,* vol. 40, pp. 9-20, 2014.

[13] "python," Python Software Foundation, 2016. [Online]. Available: https://www.python.org.

[14] H. N. Ahuja and V. Nandakumar, "Simulation model to forecast completion time," *Journal of Construction Engineering and Management,* vol. 111, no. 4, pp. 325-342, 1985.

[15] T. H. Randolph and I. Yikamoumis, "Factor model of construction productivity," *Journal of Construction Engineering and Management,* vol. 113, no. 4, pp. 623-639, 1987.

[16] T. H. Randolph, R. R. David and E. S. Victor, "Loss of labour productivity due to delivery methods and weather," *Journal of Construction Engineering and Management,* vol. 125, no. 1, pp. 39-46, 1999.

[17] E. Koehn and G. Brown, "Climatic effects on construction," *Journal of Construction Engineering and Management,* vol. 111, no. 2, pp. 129-137, 1985.

[18] ACGIH, "Threshold Limit Values (TLV) and Biological Exposure Indices (BEI)," ACGIH, Cincinnati, 2008.

[19] K. El-Rayes and O. Moselhi, "Impact of rainfall on the productivity of highway construction," *Journal of Construction Engineering and Management,* vol. 127, no. 2, pp. 125-131, 2001.

[20] D. P. Kavanaga, "SIREN: A epetitive construction simulation model," *Journal of Construction Engineering and Management,* vol. 111, no. 3, pp. 308-323, 1985.

[21] O. Moselhi, G. Daji and K. El-Rayes, "Estimating weather impact on the duration of construction activities," *Canadian Journal of Civil Engineering ,* pp. 359-366, 1997.

[22] R. Wales and S. AbouRizk, "An integrated simulation model for construction," *Simulation Practice and Theory,* pp. 401-420, 1996.

[23] A. Shahin, S. M. AbouRizk and Y. Mohamed, "Modeling weather-sensitive construction activity using simulation," *Journal of Construction Engineering and Management,* vol. 137, no. 3, pp. 238-246, 2011.

[24] S. Apipattanavis, K. Sabol, K. Molenaar, B. Rajagopalan, Y. Xi, B. Blakard and S. Patil, "Integrated framework for quantifying and predicting weather-related highway construction delays," *Journal of Construction Engineering and Management,* vol. 136, no. 11, pp. 1160-1168, 2010.

[25] S. Fatichi, V. Ivanov and E. Caporali, "Simulation of of future climate scenarioswith a weather generator," *Advances in Water Resources,* pp. 448-467, 2011.

[26] A. Shahin, A framework for cold weather construction simulation, Edmonton: University of Alberta, 2007.

[27] E. H. Chin, "Modeling daily precipitation occurrence process with Markov chain," *Water Resource Research,* vol. 13, no. 6, pp. 949-956, 1977.

[28] B. Rajagopalan, J. D. Salas and U. Lall, "Stochastic methods for modeling precipitation and stream flow," in *Advances in Data-based Approaches for Hydrologic Modeling and Forecasting*, Singapore, World Scientific Publishing Co. Pte. Ltd, 2010, pp. 17-52.

[29] M. Dubrovsky, "Creating daily weather series with use of the weather generator," *Environmetrics,* vol. 8, pp. 409-424, 1997.

[30] B. Rajagopalan, U. Lall, D. Tarboton and D. Bowles, "Multivariate nonparametric resampling scheme for generation of daily weather variables," *Stochastic Hydrology and Hydraulics,* vol. 11, no. 1, pp. 65-93, 1997.

[31] S. Apipattanavis, G. Podesta and B. Rajagopalan, "A semiparameric multivariate and multisite weather," *Water Resources Research,* Vols. VOL. 43, W11401, 2007.

[32] A. Shahin, S. M. AbouRizk and Y. Mohamed, "A weather generator for use in construction simulation models," *International Journal of Architecture, Engineering and Construction,* vol. 2, no. 2, pp. 73-87, 2013.

[33] "National Centers for Environmental Information," NOAA, [Online]. Available: http://www.ncdc.noaa.gov/.

[34] M. Dubrovsky, J. Buchtele and Z. Zalud, "High-frequency and low-frequency variability in stochastic daily weather qenerator and its effect on agricultural

and hydrologic modelling," *Climatic Change,* vol. 63, no. 1, pp. 145-179, 2004.

[35] V. Yevjevich, "Structural analysis of hydrological time series," *Colorado State University Hydrology Paper No.56,* 1972.

[36] N. C. Matalas, "Mathematical assessment of synthetic hydrology," *Water Resources Research,* vol. 3, no. 4, pp. 937-945, 1967.

[37] B. Efron, "Bootstrap methods: Another look at the jacknife," *The Annals of Statistics,* vol. 7, no. 1, pp. 1-26, 1979.

[38] H. Kunsch, "The jackknife and the bootstrap for general stationary observations," *Ann Stat. 17,* pp. 1217-1241, 1989.

[39] B. Efron and R. Tibshirani, An introduction to the bootstrap, London: Chapman and Hall, 1993.

[40] S. Robinson, Simulation : The practice of model development and use, John Wiley & Sons., 2004.

[41] N. Razali and Y. B. Wah, "Power comparisons of Shapiro–Wilk, Kolmogorov–Smirnov, Lilliefors and Anderson–Darling tests," *Journal of Statistical Modeling and Analytics,* pp. 21-33, 2011.

[42] A. Shapira and B. Lyachin, "Identification and analysis of factors affecting safety on construction sites with tower cranes," *Journal of Construction Engineering and Management,* pp. 24-33, 2009.

[43] Bclaws.ca, "Occupational Health and Safety Regulation," 29 Jul. 2015. [Online]. Available: http://www.bclaws.ca/Recon/document/ID/freeside/296_97_11.

[44] P. W. Richmond, S. A. Shoop and G. L. Blaisdell, "Cold regions mobility models," Cold Regions Research & Engineering Laboratory- US Army Corps of Engineers, 1995.

[45] S. D. Smith, J. R. Osborne and M. C. Forde, "Analysis of earth-moving systems using discrete-events simulation," *Journal of Construction Engineering and Management,* vol. 121, no. 4, pp. 388-396, 1995.

[46] T. J. LaClair and R. Truemner, "Modeling of fuel consumption for heavy-duty trucks and the impact of tire rolling resistance," SAE international, Warrendale, 2005.

[47] Environment Canada, "Weather and Meteorology - Glossary," 08 05 2014. [Online]. Available: https://ec.gc.ca/meteo-weather/default.asp?lang=En&n=B8CD636F-1&def=allShow.

[48] R. L. Peurifoy and W. B. Ledbetter, Construction planning equipment and methods, N.Y.: McGraw-Hill, 2006.

[49] M. Kyte, Z. Khatib, P. Shannon and F. Kitchener, "Effect of weather on free-flow speed," *Transportation Research Record,* pp. 60-68, 2014.

[50] D. P. Lindroth, W. R. Berglund and C. F. Wingquist, "Microwave thawing of frozen soils and gravels," *Journal of Cold Regions Engineering,* vol. 9, no. 2, pp. 53-63, 1995.

[51] A. Orlando and L. Branko, Frozen ground engineering, Hobokin, New Jersey: John Wiley & Sons, 2004.

[52] K. A. Czurde, "Freezing effects on soils: comprehensive summary of the ISGF 82," *Cold Regions Science and Technology,* vol. 8, pp. 93-107, 1983.

[53] Y. Li, W. Y. Liu and S. Frimpong, "Effect of ambient temperature on stress, deformation and temperature on dump truck tire," *Engineering Failure Analysis,* vol. 23, pp. 55-62, 2012.

[54] J. Meech and J. Parreira, "Predicting wear and temperature of autonomous haulage truck tires," in *16th IFAC Symposium on Automation in Mining, Mineral and Metal Processing*, San Diego, 2013.

[55] G. Year, "Tire maintenance manual-Goodyear Off-The-Road (OTR)," Goodyear, 2008.

[56] D. Diemand and J. H. Lever, "Cold regions issues for off-road autonomous vehicles," Engineer Research and Development Center-US Army Corps of Engineers, 2004.

[57] D. R. Freitag and T. McFadden, Introduction to Cold Regions Engineering, New York: ASCE PRESS, 1997.

[58] R. M. Fujimoto, Parallel and Distributed Simulation Systems, New York: John Wiley & Sons, 2000.

[59] R. M. Fujimoto, "Parallel and distributed simulation," in *Winter Simulation Conference*, 1995.

[60] S. Cheung and M. Loper, "Synchronizing simulations in distributed interactive simulation," in *Winter Simulation Conference*, 1994.

[61] A. Hakiri, P. Berthou and T. Gayraud, "Addressing the challenge of distributed interactive simulation with data distribution service," arXiv.org, 2010.

[62] N. Cen and E. Irene, "Adopting HLA standard for interdependency study," *Reliability Engineering and System Safety,* pp. 149-159, 2011.

[63] Department of Defence, "High level architecture run-time programmers guide (ver 3.2)," 2000.

[64] J. S. Dahmann, R. M. Fujimoto and R. M. Weatherly, "The department of defence high level architecture," in *Winter Simulation Conference*, 1997.

[65] R. Paes and M. Throckmorton, "An overview of canadaian oil sand mega projects," in *Petrolume and Chemical Industry Conferene*, Weimar, 2008.

[66] SUNCOR, "Getting oil from the oil sand," SUNCOR, 15 08 2016. [Online]. Available: http://www.suncor.com/about-us/oil-sands/process. [Accessed 16 08 2016].

[67] G. o. Canada, "Natural Resources Canada," Givernment of Canada, 19 02 2016. [Online]. Available: http://www.nrcan.gc.ca/energy/oil-sands/18094. [Accessed 15 08 2016].

[68] L. D. Nguyen, J. Kneppers, B. G. de Soto and W. Ibbs, "Analysis of adverse weather for excusable delays," *Journal of Construction Engineering and Management,* vol. 136, no. 12, pp. 1258-1267, 2010.

[69] P. &. R. Tubb, "Worldwide Construction Report," Pipeline & Gas Journal, 2016.

[70] S. AbouRizk, "Role of simulation in construction engineering and management," *Journal of Construction Engineering and Management,* vol. 136, no. (10), pp. 1140-1153, 2010.

[71] S. AbouRizk, D. Halpin, Y. Mohamed and U. Hermann, "Research in modeling and simulation for improving construction engineering operations," *Journal of Construction Engineering and Management,* vol. 137, no. (10), pp. 843-582, 2011.

[72] W. J. Trybula, "Building simulation models without data," in *Systems, Man, and Cybernetics*, San Antonio, TX, 1994.

[73] B. Biller and C. Gunes, "Introduction to simulation input modeling," in *Winter Simulation Conference*, 2010.

[74] I. D. Tommelein, "Pull-driven scheduling for pipe-spool installation: Simulation of lean construction technique," *Journal of Construction Engineering and Management,* vol. 124, no. (4), pp. 279-288, 1998.

[75] J. Shi and S. AbouRizk, "Continuous and combined event-process models for simulating pipeline construction," *Construction Management and Economics,* vol. 16, pp. 489-498, 1998.

[76] B. Biller and S. Gosh, "Mutlivariate input processes," in *Handbooks in Operations and Management Science*, vol. 13, 2006, pp. 123-153.

[77] B. Biller and B. L. Nelson, "Modeling and generating multivariate time-series input processes using a vector autoregressive technique," *ACM Transactions on Modeling and Computer Simulation,* vol. 13, no. 3, pp. 211-237, 2003.

[78] A. Touran and E. P. Wiser, "Monte carlo techniques with correlated random variables," *Journal of Construction Engineering and Management,* vol. 118, no. 2, pp. 258-272, 1992.

[79] A. Touran, "Probabilistic cost estimating with subjective correlations," *Journal of Construction Engineering and Management,* vol. 119, no. 1, pp. 58-71, 1993.

[80] Y. Moret and H. H. Einstein, "Modeling correlations in rail line construction," *Journal of Construction Engineering and Mangement,* vol. 138, no. 9, pp. 1075-1084, 2012.

[81] A. Firouzi, W. Yang and C.-Q. Li, "Prediction of total cost of construction project with dependent cost items," *Journal of Construction Engineerng and Management,* 2016.

[82] L. Song, P. Wang and S. AbouRizk, "A virtual shop modeling system for industrial fabrication shops," *Simulation Modelling Practice and Theory,* vol. 14, pp. 649-662, 2006.

[83] D. Hu and Y. Mohamed, "Simulation-model-structuring methodology for industrial construction fabrication shops," *Journal of Construction Engineering and Management,* vol. 140, no. 5, 2014.

[84] I. D. Tommelein, "Process benefits from use of standard products — simulation experiments using the pipe spool model," in *Conference of the International Group for Lean Construction*, Santiago, 2006.

[85] M. Drmota, Random trees, An interplay between combinatorics and probability, NewYork: SpringerWien, 2009.

[86] L. Alonso and R. Schott, Random Gneration of Trees, Boston: Kluwer Acamdemic Publishers, 1995.

[87] D. Revuz, Markov chains, Amesterda: New York: North Holland, 1984.

[88] L. Liu, Y.-K. Ho and S. Yau, "Clustering DNA sequences by feature vectors," *Molecular Phylogenetics and Evolution,* vol. 41, pp. 64-69, 2006.

[89] B. Everitt, Cluster Analysis, New York: Wiley & Sons, 1980.

[90] M. Robnik-Sikonja, "Data generators for learning systems based on RBF networks," *IEEE Transactions on Neural Networks and Learning Systems,* 2014.

[91] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[92] C. C. Aggarwal, Data classification: algorithms and applications, CRC Press, 2015.

[93] M. . J. Swain and D. H. Ballard, "Color indexing," *International journal of computer vision,* vol. 7.1, pp. 11-32, 1991.

[94] N. G. Hall and M. E. Posner, "Generating experimental data for computational testing with machine scheduling applications," *Institute for Operations Research and the Management Sciences,* vol. 49, no. 6, pp. 854-865, 2001.

[95] A. Otto, C. Otto and A. Scholl, "Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line

balancing," *European Journal of Operational Research,* vol. 339, pp. 33-45, 2013.

[96] H. P. Crowder, R. S. Dembo and J. M. Mulvey, "Reporting computational experiments in mathematical programming," *Mathematical Programming,* vol. 15, pp. 316-329, 1978.

[97] M. Cadoli, A. Giovanardi and M. Schaerf , "Experimental analysis of the computational cost of evaluating quantified boolean formulae," *Advances in Artificial Intelligence,* vol. 1321, pp. 207-218, 1997.

[98] C.-Y. LEE, J. Bard, M. Pinedo and W. E. Wilhelm, "Guidlines for reporting computational results in IIE transactions," *IIE Transactions,* vol. 25, no. 6, pp. 121-123, 1993.

[99] A. Drexl, . R. Nissen, J. H. Patterson and F. Salewski, "ProGen/px ± An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions," *European Journal of Operational Research,* vol. 125, pp. 59-72, 2000.

[100] T. Gau and G. Wascher, "CUTGENI" A problem generator for the standard one-dimensional cutting stock problem," *European Journal of Operational Research,* vol. 84, pp. 572-579, 1995.

[101] E. Silva , J. F. Oliveira and G. Wascher , "2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems," *European Journal of Operational Research ,* vol. 237, pp. 846-856, 2014.

[102] J.-J. Kim and W. C. Ibbs, "Work-Package-Process model for piping construction," *Journal of Construction Engineering and Management,* vol. 121, no. 4, pp. 381-387, 1995.

[103] J. Song, W. R. Fagerlund, C. T. Hass, C. B. Tatum and J. A. Vanegas, "Considering prework on industrial projects," *Journal of Construction Engineering and Management,* vol. 131, no. 6, pp. 723-733, 2005.

[104] McGraw-Hill, "Prefabrication and modularization: Increasing productivity in construction industry," McGraw-Hill, Bedford, MA, 2011.

[105] C. T. Haas, J. T. O'Connor, R. L. Tucker, J. A. Eickmann and W. R. Fagerlund, "Prefabrication and preassembly trends and effects on the construction workforce," Center for Construction Industry Studies, 2000.

[106] C. B. Tatum, J. A. Vanegas and J. M. Williams, "Constructability improvement using prefabrication, preassembly, and modularization," Construction Industry Institute, Austin, TX, 1987.

[107] J. O. Choi, J. T. O'Connor and T. W. Kim, "Recipes for cost and schedule successes in industrial modular projects: qualitative comparative analysis," *Journal of Construction Engineering and Management,* vol. 142, no. 10, 2016.

[108] C. T. Hass and W. R. Fagerlund, "Preliminary research of fabrication, pre-assembly, modularization and off-site fabrication in construction," The Construction Industry Institute, The University of Texas at Austin, Austin, Texas, 2002.

[109] P. E. Dawar Naqvi, P. E. Eric Wey, P. Jayesh and S. Glenn, "Modularization in the petrochemical industry," in *Structure Congress*, 2014.

[110] M. B. Murtaza, D. J. Fisher and M. J. Skibniewski, "Knowledge-based approach to modular construction decision support," *Journal of Construction Engineering and Management,* vol. 119, no. 1, pp. 115-130, 1993.

[111] C. News, "Kearl oilsands project price tag increases by $2B," CBC News, 1 2 2013. [Online]. Available: http://www.cbc.ca/news/business/kearl-oilsands-project-price-tag-increases-by-2b-1.1380005. [Accessed 19 7 2016].

[112] SupplyChainDigest, "The 10 keys to global logistics excellence," SupplyChainDigest, 2007.

[113] AberdeenGroup, "Best practices in international logistics," AberdeenGroup, 2006.

[114] P. Kristofic, J. Kok, S. de Vries and J. v. S.-v. Hoff, "Financial supply chain management – Challenges and obstacles.," in *Proceedings in Finance and Risk Percpectives*, 2012.

[115] J. T. O'Connor, W. J. O'Brien and J. O. Choi, "Critical success factors and enablers for optimum and maximum industrial modularization," *Journal of Construction Engineering and Management,* vol. 140, no. 6, 2014.

[116] C. B. Tatum, "Management challenges of integrating construction methods and design approaches," *Journal of Construction Engineering and Management,* vol. 5, no. 2, pp. 139-154, 1989.

[117] H. Taghaddos, U. Hermann, S. AbouRizk and Y. Mohamed, "Simulation-based scheduling of modular construction using multi-agent resource allocation," in *International Conference on Advances in System Simulation*, 2010.

[118] K. U. Damodaru, "Materials management: The key to successful project management," *Journal of Construction Engineering and Management,* vol. 15, no. 1, pp. 30-34, 1999.

[119] F. F. Alireza Ahmadian, A. Akbarnezhad, T. H. Rashidi and S. T. Waller, "Accounting for transport times in planning off-site shipment of construction materials," *Journal of Construction Engineering and Management,* vol. 142, no. 1, p. 04015050, 2016.

[120] J. Song, C. T. Haas, C. Caldas, E. Ergen and B. Akinci, "Automating the task of tracking the delivery and receipt of fabricated pipe spools in industrial projects," *Automation in Construction,* vol. 15, pp. 166-177, 2006.

[121] M. Al-Alawi, A. Bouferguene and Y. Mohamed, "Random generation of complex data structures for the simulation of construction operations," in *Construction Research Congress*, Puerto Rico, 2016.

[122] J. Song, C. Haas, C. Caldas, E. Ergen, B. Akini, C. R. Wood and J. WadephulL, "Field trials of RFID technology for tracking fabricated pipe-phase II," FIATECH, Austin,TX, 2004.

[123] P. Wang , Y. Mohamed, S. M. Abourizk and A. R. Tony Rawa, "Flow production of pipe spool fabrication: Simulation to support implementation of lean technique," *Journal of Construction Engineering Management,* vol. 135, no. 10, pp. 1027-1038, 2009.

[124] P. Wang and S. M. AbouRizk, "Large-scale simulation modeling system for industrial construction," *Candian Journal of Civil Engineering ,* vol. 36, pp. 1517-1529, 2009.

[125] I. D. Tommelein, "Process benefits from use of standard products- Simulation experiments using the pipe spool model," in *Conference of International Group of Lean Construction*, Santiego, 2006.

[126] D. HU and Y. MOHAMED, "Pipe spool fabrication sequencing by automated planning," in *Construction Research Congress*, West Lafayette, Indiana, 2012.

[127] D. Pisinger, "Heuristics for the container loading problem," *European Jpurnal of Operational Research,* vol. 141, pp. 382-392, 2002.

[128] E. G. Coffman, J. Y.-T. Leung and D. W. Ting, "Bin packing: maximizing the number of pieces packed," *Acta Informatics,* vol. 9, pp. 263-271, 1978.

[129] M. R. Garey and D. S. Johnson, "Approximation algorithms for bin packing problems: A survey," in *Analysis and Design of Algorithms in Combinatorial Optimization*, Springer-Verlag Wien, 1981, pp. 147-172.

[130] S. P. Fekete and J. Schepers, "On more-dimensional packing I: Modeling," Center for Applied Computer Science, Universitat¨ zu Koln, 2000.

[131] J. M. Valerio de Carvalho, "LP models for bin packing and cutting stock problems," *European Journal of Operational Research,* vol. 141, pp. 253-273, 2002.

[132] J. O. Berkey and P. Y. Wang, "Two-dimensional finite bin-packing algorithms," *The Journal of the Operational Research Society,* vol. 38, no. 5, pp. 423-429, 1987.

[133] K. Fleszar and K. S. Hindi, "New heuristics for one-dimensional bin-packing," *Computers & Operations Research,* vol. 29, pp. 821-839, 20002.

[134] D. Mack and A. Bortfeldt, "A heuristic for solving large bin packing problems in two and three dimensions," *Central European Journal of Operations Research,* vol. 20, pp. 337-354, 2012.

[135] M. Delorme, M. Iori and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research,* vol. 255, pp. 1-20, 2016.

[136] S. H. Zanakis and J. R. Evans, "Heuristic "Optimization": Why, When, and How to Use It," *The Instituete of Management Sciences,* vol. 11, no. 5, pp. 84-91, 1981.

[137] M. R. Garey, R. L. Graham and J. D. Ullman, "Worst-case analysis of memory allocation algorithms," in *Proceedings of the fourth annual ACM symposium on Theory of computing*, New York, 1972.

[138] E. G. Coffman J., J. Csirik, G. Galambos, S. Martello and D. Vigo, "Bin packing approximation algorithms: survey and classification," in *Handbook of Combinatorial Optimization*, New York, Springer Science+Business Media, 2013, pp. 455-531.

[139] S. P. Fekete and J. Schepers, "A combinatorial characterization of higher-dimensional orthogonal packing," *Mathematics of Operations Research,* vol. 29, no. 2, pp. 353-368, 2004.

[140] S. P. Fekete and J. Schepers, "On more-dimensional packing II: Bounds," Applied Computer Science, Universitat¨ zu Koln, 2000.

[141] H. Shachni, T. Tamir and O. Yehezkely, "Approximation schemes for packing with item fragmentation," *Theory of Computing Systems,* vol. 43, no. 1, pp. 81-89, 2008.

[142] E. L. Lawler and D. E. Wood, "Branch-and-Bound methods: A survey," *Operations Research,* vol. 14, no. 4, pp. 699-719, 1966.

[143] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," *Econometrica,* vol. 28, no. 3, pp. 497-520, 1960.

[144] R. T. Haftka and Z. Gurdal, Elements of structural optimizarion, The Netherlands: Kluwer Academics Publishers, 1992.

[145] R. H. Jackson and J. M. Mulvey, "A critical review of comparisons of mathematical programming algorithms and software (1953-1977)," *Journal of Research of the National Bureau of Standards,* vol. 83, no. 6, pp. 563-584, 1977.

Appendix A.

1. Parametric weather generator- Python code

```
import numpy
import math
import os, sys
import pyodbc
import random

def Paraweather(day, month, length,filenumber):

# (1) Connect to the database which contains all parameters needed for generating
weather series

 conect="DSN=DataParametric"

 c1=pyodbc.connect(conect)


# (2) Initialize first-day residuals, matrix A and matrix B

IRS=[random.normalvariate(0,1),random.normalvariate(0,1),random.normalvariate(0
,1),random.normalvariate(0,1)]

 A=numpy.matrix([[0.368,-0.014,0.077,-0.058],[0.221, 0.004, 0.004, 0.037],[0.085,-
0.005, 0.406, 0.246],[0.02,0.002,0.095,0.411]])

 B=numpy.matrix([[0.923,0,0,0],[0.393,0.894,0,0],[-0.016,-0.005,0.406,0],[-
0.252,0.135,0.304,0.784]])


# (3) Generate weather series using input "length"

 file = open("ParametricWeatherSeries"+str(filenumber)+".txt", "w")

file.write("Number"+","+"Day"+","+"Month"+","+"State"+","+"MAXTEMP"+","+"MINTE
P"+","+"MAXRH"+","+"MINRH"+","+"PRECIPITATION"+","+"WINDSPEED"+"\n")

 for y in range (1,length+1):

  x= day


# (4) Define SQL queries that will be used to extract information from the database

  SQL1= ''' SELECT MON,DY, P_w, P_w_w, P_w_d FROM wet_dry WHERE
MON='''+str(month)+''';'''
```

```
  SQL2= ''' SELECT MON,DY , a, b FROM Precipitation WHERE
MON='''+str(month)+''';'''

  SQL4= ''' SELECT MON,DY,  a, b FROM WindSpeed WHERE
MON='''+str(month)+''';'''


# (5) The “if statement” below will make sure that each month will use different
parameters as per the database

  if month <=12:


# (6) Generate average wind speed for each day in a month


    for row in c1.execute(SQL4):

        wdsp= random.gammavariate(row.a,row.b)

# (7) Define the state of the day (wet/dry) and other weather variables


    for row in c1.execute(SQL1):

       if x <= row.DY: # moves from one month to another

# The probability that a day in month m will be wet Pm(w)

            P_w=row.P_w

# The probability that a wet day in month m is preceded by a wet day Pm(w/w)

            P_w_w=row.P_w_w

# The probability that a wet day in month m is preceded by a dry day Pm(w/d)

            P_w_d=row.P_w_d

######## Define the dry state of the day########

            if (random.uniform(0,1)-P_w)>0:

                State=0

                P_w= P_w_d

                preci=0

# generate the mean and standard deviation of the correlated weather variables
```

```
            SQL3= ''' SELECT MON,DY, State, TMAXM, TMAXSTD, TMINM,
TMINSTD, RHMaxM, RHMaxSTD, RHMinM, RHMinSTD FROM MSTD WHERE
MON='''+str(month)+'''AND DY='''+str(day)+''' AND State='''+str(State)+''';'''

            for row in c1.execute(SQL3):

 # calculate weather residuals

 # (we call it ed)= (nx1) matrix of random components sampled from a standard
normal distribution with a mean of 0 and a standard deviation of 1.


ed=[random.normalvariate(0,1),random.normalvariate(0,1),random.normalvariate(0,
1),random.normalvariate(0,1)]


# xd= (nx1) matrix of residual elements for day d for parameters 1 to n

               xd=numpy.dot(A,IRS)+numpy.dot(B,ed)

# Calculate values of all weather variables

               TMAXV=(xd[0,0]*row.TMAXSTD)+row.TMAXM

               TMINV=(xd[0,1]*row.TMINSTD)+row.TMINM

               RHMAXV=(xd[0,2]*row.RHMaxSTD)+row.RHMaxM

               RHMINV=(xd[0,3]*row.RHMinSTD)+row.RHMinM


######## defines the dry state of the day########

         else:

            State=1

            P_w= P_w_w

            SQL3= ''' SELECT MON,DY, State, TMAXM, TMAXSTD, TMINM,
TMINSTD, RHMaxM, RHMaxSTD, RHMinM, RHMinSTD FROM MSTD WHERE
MON='''+str(month)+'''AND DY='''+str(day)+''' AND State='''+str(State)+''';'''

            # generate precipitation

            for row in c1.execute(SQL2):

               preci= random.gammavariate(row.a,row.b)

# generate the mean and standard deviation of the correlated weather variables
```

```
            for row in c1.execute(SQL3):


 # calculate weather residuals

            # ?_d (we call it ed)= (nx1) matrix of random components sampled from a
standard normal distribution with a mean of 0 and a standard deviation of 1.


ed=[random.normalvariate(0,1),random.normalvariate(0,1),random.normalvariate(0,
1),random.normalvariate(0,1)]

# xd= (nx1) matrix of residual elements for day d for parameters 1 to n

               xd=numpy.dot(A,IRS)+numpy.dot(B,ed)

            # Calculate values of all weather variables

               TMAXV=(xd[0,0]*row.TMAXSTD)+row.TMAXM

               TMINV=(xd[0,1]*row.TMINSTD)+row.TMINM

               RHMAXV=(xd[0,2]*row.RHMaxSTD)+row.RHMaxM

               RHMINV=(xd[0,3]*row.RHMinSTD)+row.RHMinM


# Advance the calendar by one day


file.write(""+str(y)+","+str(x)+","+str(month)+","+str(State)+","+str(TMAXV)+","+str(T
MINV)+","+str(RHMAXV)+","+str(RHMINV)+","+str(preci)+","+str(wdsp)+"\n")
         day+=1
         IRS=[xd[0,0],xd[0,1],xd[0,2],xd[0,3]]
         print(y,x,month,State,TMAXV,TMINV,RHMAXV,RHMINV,preci,wdsp)

# advance the calender by one month

     else:

         day=1

         month+=1

# when full year is reached, initialize the generator to January
  else:

         month=1

 file.close()
```

```
print ('Please enter the day, month, length of weather series required and the file
registered number')
userinput= [input(),input(),input(),input()]
Paraweather(int(userinput[0]),int(userinput[1]),int(userinput[2]),userinput[3])
```

2. Non-Parametric weather generator- Python code

```
import numpy
import math
import os, sys
import pyodbc
import random

def NonParaweather(day,month,length,filenumber):


# (1) Connect to the database which contains all parameters needed to generate
weather series

 conect="DSN=FortMcmurray"

 c1=pyodbc.connect(conect)

 year=int(random.uniform(1962,2002))


# (2) Save the generated data in an external text file

 file = open("NonParaWeatherSeries"+str(filenumber)+".txt", "w")


file.write("Number"+","+"Day"+","+"Month"+","+"Year"+","+"MAXTEMP"+","+"MINTE
P"+","+"MAXRH"+","+"MINRH"+","+"PRECIPITATION"+","+"WINDSPEED"+"\n")


 for y in range (1,length+1):

# first make sure to iterate the selection of the year within the database range

  if year<=2002:

# add the feature to move from one year to another whenever the end of the month
of December is reached

   if month<=12:

    SQL1= ''' SELECT MON, DY FROM Days_in_Months WHERE
MON='''+str(month)+''';'''
```

```
   for row in c1.execute(SQL1):

    SQL2= ''' SELECT Year, Month, Day, MaxTemp, MinTemp,
MaxRel_Hum,MinRel_Hum, AvgOfWind_Spd,Total_Precip_mm FROM daily
WHERE Year='''+str(year)+'''AND Month='''+str(month)+''' AND Day='''+str(day)+''';'''

# Add the feature to move from one month to another whenever the end of the
month is reached

    if day <= row.DY:

          for row in c1.execute(SQL2):

            print(y,row.Day,row.Month,row.Year,row.MaxTemp,row.MinTemp,
row.MaxRel_Hum,row.MinRel_Hum,row.Total_Precip_mm,row.AvgOfWind_Spd)

            # write all weather variables in the text file


file.write(""+str(y)+","+str(day)+","+str(month)+","+str(year)+","+str(row.MaxTemp)+",
"+str(row.MinTemp)+","+str(row.MaxRel_Hum)+","+str(row.MinRel_Hum)+","+str(ro
w.Total_Precip_mm)+","+str(row.AvgOfWind_Spd)+"\n")

            day+=1

    else:

           day=1

           month+=1
  else:

         month=1
         year+=1

 else:

      year=int(random.uniform(1962,2002))

print('Please inter the day, month, length of weather series required and the file
registered number')
userinput= [input(),input(),input(),input()]

NonParaweather(int(userinput[0]),int(userinput[1]),int(userinput[2]),userinput[3])
```

Appendix B.

Table B- 1 Monthly averages of maximum temperature (MAXTEMP)

Month	MAXTEMP		
	Parametric	**Non-Parametric**	**Historical**
Jan	-15.139	-15.270	-14.491
Feb	-7.383	-7.932	-8.180
March	-0.017	0.657	-0.785
Apr	9.134	8.846	9.048
May	16.529	16.244	16.016
Jun	20.949	19.629	20.147
Jul	23.480	21.336	21.932
Aug	22.281	21.598	20.810
Sep	16.418	13.792	14.595
Oct	8.022	7.744	8.156
Nov	-4.836	-3.251	-4.174
Dec	-11.312	-11.115	-12.229

Table B- 2 Monthly averages of minimum temperature (MINTEMP)

Month	MINTEMP		
	Parametric	**Non-Parametric**	**Historical**
Jan	-24.649	-24.209	-23.918
Feb	-20.458	-18.797	-19.541
March	-14.301	-12.086	-13.905
Apr	-3.784	-3.622	-3.908
May	2.681	3.128	3.129
Jun	7.634	7.678	8.139
Jul	10.276	10.316	10.549
Aug	8.578	8.940	8.943
Sep	3.256	2.622	3.408
Oct	-2.167	-3.269	-2.621
Nov	-14.103	-11.553	-12.680
Dec	-21.061	-19.167	-20.764

Table B- 3 Monthly averages of maximum relative humidity (MAXRH)

Month	MAXRH		
	Parametric	Non-Parametric	Historical
Jan	83.601	83.678	83.199
Feb	83.514	83.889	82.999
March	81.963	81.623	82.395
Apr	81.542	82.527	81.259
May	82.721	81.265	81.957
Jun	87.694	85.967	87.322
Jul	90.787	90.960	90.647
Aug	92.701	90.374	92.374
Sep	91.505	91.723	91.943
Oct	89.508	88.548	89.178
Nov	87.577	88.557	88.183
Dec	83.948	84.416	84.949

Table B- 4 Monthly averages of minimum relative humidity (MINRH)

Month	MINRH		
	Parametric	Non-Parametric	Historical
Jan	65.321	67.085	65.562
Feb	59.637	61.350	58.809
March	47.819	47.829	48.578
Apr	38.598	40.510	37.834
May	34.587	33.200	33.807
Jun	39.863	38.977	39.925
Jul	44.982	45.460	44.434
Aug	46.956	42.487	46.344
Sep	48.288	50.560	50.513
Oct	56.144	54.619	54.798
Nov	67.422	70.627	68.567
Dec	66.826	69.158	68.498

Table B- 5 Monthly Averages of Precipitation (mm)

Month	Precipitation		
	Parametric	Non-Parametric	Historical
Jan	20.552	22.060	19.190
Feb	18.442	11.840	15.331
March	14.375	12.820	16.560
Apr	24.247	30.600	21.414
May	41.980	35.980	37.679
Jun	78.921	52.240	70.900
Jul	64.941	78.330	79.971
Aug	57.318	52.180	68.352
Sep	45.191	40.480	49.621
Oct	40.844	36.480	28.748
Nov	24.661	25.090	23.424
Dec	22.682	19.170	20.388

Table B- 6 Monthly averages of wind speed

Month	Wind speed		
	Parametric	Non-Parametric	Historical
Jan	8.666	7.628	8.397
Feb	9.160	8.726	9.012
March	9.820	10.154	9.840
Apr	11.064	11.298	11.015
May	10.844	10.996	10.979
Jun	9.876	9.634	9.647
Jul	8.583	8.595	8.978
Aug	9.068	8.668	8.710
Sep	9.391	9.134	9.524
Oct	10.120	10.252	10.301
Nov	8.694	8.216	8.866
Dec	8.581	8.181	8.371

Table B- 7 Standard deviation of maximum temperature (MAXTEMP)

Month	MAXTEMP		
	Parametric	Non-Parametric	Historical
Jan	10.326	10.719	10.516
Feb	10.310	9.537	10.055
March	8.206	8.165	8.338
Apr	7.305	7.123	6.907
May	5.803	6.199	5.921
Jun	4.871	5.167	5.117
Jul	4.218	4.423	4.750
Aug	5.107	6.009	5.549
Sep	6.165	6.225	5.794
Oct	6.935	6.586	6.667
Nov	7.845	7.910	8.269
Dec	9.686	9.179	9.867

Table B- 8 Standard deviation of minimum temperature (MINTEMP)

Month	MINTEMP		
	Parametric	Non-Parametric	Historical
Jan	9.392	10.583	9.819
Feb	9.979	9.882	9.816
March	9.671	8.790	8.989
Apr	5.667	6.267	6.351
May	4.122	4.923	4.911
Jun	3.584	3.724	3.625
Jul	2.871	2.844	3.029
Aug	3.810	3.896	3.783
Sep	3.949	4.415	4.675
Oct	5.032	5.695	5.588
Nov	8.123	7.927	7.731
Dec	8.256	8.862	9.351

Table B- 9 Standard deviation of maximum relative humidity (MAXRH)

Month	MAXRH		
	Parametric	Non-Parametric	Historical
Jan	5.493	8.506	9.037
Feb	5.573	9.373	9.224
March	5.703	9.258	9.661
Apr	8.778	11.432	12.099
May	8.957	12.965	13.321
Jun	7.343	10.172	10.420
Jul	4.971	7.056	7.685
Aug	3.703	6.796	6.700
Sep	5.312	7.051	7.725
Oct	5.547	8.527	8.663
Nov	4.605	7.116	7.437
Dec	5.133	8.622	8.609

Table B- 10 Standard deviation of minimum relative humidity (MINRH)

Month	MINRH		
	Parametric	Non-Parametric	Historical
Jan	10.827	9.294	9.863
Feb	13.166	12.038	12.401
March	13.256	14.937	13.978
Apr	16.427	17.511	16.140
May	15.773	14.613	15.441
Jun	16.381	16.243	16.310
Jul	14.774	14.052	14.323
Aug	14.061	14.092	14.522
Sep	17.735	17.925	17.183
Oct	17.137	16.966	17.370
Nov	12.059	12.433	12.239
Dec	9.912	10.319	10.363

Table B- 11 Standard deviation of precipitation

Month	Precipitation		
	Parametric	**Non-Parametric**	**Historical**
Jan	5.781	9.482	9.698
Feb	9.826	13.524	9.679
March	8.195	7.689	8.986
Apr	14.289	11.155	12.798
May	24.999	27.474	23.538
Jun	20.118	26.310	35.203
Jul	25.420	29.374	33.208
Aug	24.820	30.693	39.455
Sep	16.753	32.923	30.901
Oct	13.041	24.846	18.716
Nov	10.549	11.652	11.780
Dec	6.827	10.125	9.418

Table B- 12 Standard deviation of wind speed

Month	Wind speed		
	Parametric	**Non-Parametric**	**Historical**
Jan	6.015	4.586	4.684
Feb	4.627	4.585	4.459
March	4.361	4.504	4.446
Apr	4.380	4.328	4.167
May	4.399	4.207	4.278
Jun	4.029	4.194	3.930
Jul	3.810	4.087	4.024
Aug	4.109	4.074	4.002
Sep	4.356	4.568	4.281
Oct	4.930	4.903	4.599
Nov	4.035	4.007	4.482
Dec	5.536	4.901	4.807

Appendix C.

Table C- 1 Performance benchmark results for trucks and excavators

Resource Type/ No.	Total working Duration	Number of Breakdown	Total Breakdown Duration	Number of Maintenance	Total Maintenance Duration	%		
						Available	Breakdown	Maintenance
Truck1	7217.75	50	1108.70	21	433.55	82.39	12.66	4.95
Truck2	7338.21	39	858.70	21	563.09	83.77	9.80	6.43
Truck3	7508.23	35	765.40	21	486.37	85.71	8.74	5.55
Truck4	7165.69	47	1033.60	21	560.71	81.80	11.80	6.40
Truck5	7355.33	32	711.40	21	693.27	83.96	8.12	7.91
Truck6	7464.24	28	616.50	21	679.26	85.21	7.04	7.75
Truck7	7060.24	45	991.00	21	708.76	80.60	11.31	8.09
Truck8	7235.85	40	874.56	21	649.59	82.60	9.98	7.42
Truck9	7174.80	42	930.30	21	654.90	81.90	10.62	7.48
Excavator1	7396.68	35	757.66	23	605.66	84.44	8.64	6.91
Excavator2	7123.33	35	1013.40	23	623.27	81.32	11.57	7.11
Excavator3	7404.59	34	746.76	23	608.65	84.53	8.52	6.95

Table C- 2 First scenario (SC1) results with respect to different temperature limit (T)

(T)	SC1 Breakdown repair durations					
	Trucks			Excavators		
	SC1-Min	SC1-Average	SC1-Max	SC1-Min	SC1-Average	SC1-Max
-18	620.66	901.41	1161.41	751.41	858.45	1053.31
-19	616.50	900.09	1161.41	751.41	856.88	1048.80
-20	616.50	898.82	1161.41	751.41	855.41	1048.80
-21	616.50	896.43	1152.72	746.76	853.70	1048.80
-22	616.50	893.61	1148.93	746.76	851.36	1048.80
-23	616.50	892.93	1152.72	746.76	850.74	1044.23
-24	616.50	891.36	1143.92	746.76	849.58	1040.05
-25	616.50	890.20	1143.92	746.76	848.70	1035.86
-26	616.50	888.47	1143.92	746.76	847.83	1035.86
-27	616.50	887.11	1143.92	746.76	846.95	1035.86
-28	616.50	886.09	1143.92	746.76	846.48	1035.86
-29	616.50	884.32	1143.92	746.76	845.02	1035.86
-30	616.50	882.73	1135.72	746.76	844.45	1035.86

Table C- 3 Second scenario (SC2) results with respect to different temperature limit (T)

(T)	SC2 Breakdown repair durations					
	Trucks			Excavators		
	SC2-Min	SC2-Average	SC2-Max	SC2-Min	SC2-Average	SC2-Max
-18	634.79	915.27	1183.66	764.50	868.84	1066.37
-19	629.22	912.41	1170.48	760.25	866.65	1057.60
-20	625.51	910.61	1170.18	760.25	865.46	1057.42
-21	625.51	909.07	1170.18	760.25	863.43	1057.42
-22	625.51	907.83	1170.18	760.25	862.43	1057.42
-23	625.51	906.32	1165.75	760.25	861.70	1053.31
-24	625.05	904.23	1161.41	760.25	860.68	1053.31
-25	625.05	902.31	1161.41	755.63	858.88	1053.31
-26	620.88	900.59	1161.41	751.01	857.57	1053.31
-27	620.88	899.33	1157.21	751.01	856.10	1053.31
-28	620.88	897.90	1157.21	751.01	854.92	1044.93
-29	620.88	896.54	1152.72	751.01	853.75	1044.93
-30	620.88	894.99	1152.72	746.76	852.88	1044.93

Table C- 4 Third scenario (SC3) results with respect to different temperature limit (T)

(T)	SC3 Breakdown repair durations					
	Trucks			Excavators		
	SC3-Min	SC3-Average	SC3-Max	SC3-Min	SC3-Average	SC3-Max
-18	620.67	900.85	1165.69	755.66	858.23	1052.92
-19	620.67	898.16	1157.14	751.08	855.59	1048.80
-20	620.67	896.24	1157.14	751.08	854.30	1048.80
-21	616.50	894.81	1157.14	746.76	853.32	1048.80
-22	616.50	893.24	1157.14	746.76	853.01	1048.80
-23	616.50	891.54	1148.29	746.76	850.75	1044.33
-24	616.50	890.52	1148.29	746.76	849.13	1044.33
-25	616.50	889.11	1148.29	746.76	848.56	1040.14
-26	616.50	888.00	1148.29	746.76	847.84	1031.39
-27	616.50	886.48	1143.58	746.76	846.66	1031.39
-28	616.50	885.57	1143.58	746.76	845.95	1031.39
-29	616.50	884.14	1138.79	746.76	845.06	1031.39
-30	616.50	882.99	1134.11	746.76	844.63	1031.39

Appendix D.

Table D- 1 P-values of Anderson-Darling normality test on the number of components in the original and generated pipelines populations

Component	***p*-value**	
	n **(original pipelines)**	n **(generated pipelines)**
Pcomponent	<0.005	<0.005
Instrument	<0.005	<0.005
Valve	<0.005	<0.005
Flange	<0.005	<0.005
Tube	<0.005	<0.005
Elbow	<0.005	<0.005
Tee	<0.005	<0.005
Reducer	<0.005	<0.005
Coupling	<0.005	<0.005

Appendix E.

1. Correlation coefficients matrix of pipelines components in the original data:

$$\begin{array}{r}
Pcomponent \\ Instrument \\ Valve \\ Flange \\ Tube \\ Elbow \\ Tee \\ \mathrm{Re}\,ducer \\ Coupling
\end{array}
\begin{bmatrix}
1 & 0.23 & 0.26 & 0.32 & 0.11 & 0.08 & 0.15 & 0.2 & 0.06 \\
0.23 & 1 & 0.64 & 0.64 & 0.46 & 0.33 & 0.54 & 0.41 & 0.12 \\
0.26 & 0.64 & 1 & 0.72 & 0.62 & 0.5 & 0.74 & 0.47 & 0.08 \\
0.32 & 0.64 & 0.72 & 1 & 0.52 & 0.45 & 0.57 & 0.33 & 0 \\
0.11 & 0.46 & 0.62 & 0.52 & 1 & 0.9 & 0.85 & 0.45 & 0.09 \\
0.08 & 0.33 & 0.5 & 0.45 & 0.9 & 1 & 0.67 & 0.43 & 0.08 \\
0.15 & 0.54 & 0.74 & 0.57 & 0.85 & 0.67 & 1 & 0.42 & 0 \\
0.2 & 0.41 & 0.47 & 0.33 & 0.45 & 0.43 & 0.42 & 1 & 0.18 \\
0.06 & 0.12 & 0.08 & 0 & 0.09 & 0.08 & 0 & 0.18 & 1
\end{bmatrix}$$

2. Correlation coefficients matrix of pipelines components in the generated data:

$$\begin{array}{r}
Pcomponent \\ Instrument \\ Valve \\ Flange \\ Tube \\ Elbow \\ Tee \\ \mathrm{Re}\,ducer \\ Coupling
\end{array}
\begin{bmatrix}
1 & 0.5 & 0.61 & 0.68 & 0.69 & 0.67 & 0.63 & 0.52 & 0.22 \\
0.5 & 1 & 0.74 & 0.72 & 0.76 & 0.76 & 0.77 & 0.39 & 0.26 \\
0.61 & 0.74 & 1 & 0.86 & 0.9 & 0.89 & 0.89 & 0.55 & 0.34 \\
0.68 & 0.72 & 0.86 & 1 & 0.89 & 0.87 & 0.83 & 0.54 & 0.3 \\
0.69 & 0.76 & 0.9 & 0.89 & 1 & 0.98 & 0.93 & 0.64 & 0.31 \\
0.67 & 0.76 & 0.89 & 0.87 & 0.98 & 1 & 0.91 & 0.6 & 0.31 \\
0.63 & 0.77 & 0.89 & 0.83 & 0.93 & 0.91 & 1 & 0.6 & 0.31 \\
0.52 & 0.39 & 0.55 & 0.54 & 0.64 & 0.6 & 0.6 & 1 & 0.27 \\
0.22 & 0.28 & 0.34 & 0.3 & 0.31 & 0.31 & 0.31 & 0.27 & 1
\end{bmatrix}$$

Appendix F.

Table F- 1 Feature vectors centroids of pipeline components generated from the original data

Original data	**Cluster #**		
Attribute	**Full data**	**0**	**1**
	841	**793**	**48**
Pcomponent_*n*	0.4281	0.4086	0.75
Pcomponent_*T*	7.8383	4.6494	60.5208
Pcomponent_*D*	13.0392	2.1354	193.1806
Instrument_*n*	0.5268	0.4023	2.5833
Instrument_*T*	21.1225	9.5914	211.625
Instrument_*D*	50.0707	8.9261	729.8142
Valve_*n*	2.4792	1.9912	10.5417
Valve_*T*	97.9382	44.6873	977.6875
Valve_*D*	129.6558	53.8997	1381.2085
Flange_*n*	3.8621	3.0895	16.625
Flange_*T*	127.874	54.2018	1345
Flange_*D*	119.0875	45.5125	1334.6078
Tube_*n*	11.6183	9.4061	48.1667
Tube_*T*	363.4732	183.4704	3337.2708
Tube_*D*	132.9783	59.6594	1344.2678
Elbow_*n*	6.132	5.0984	23.2083
Elbow_*T*	184.5517	97.9269	1615.6667
Elbow_*D*	125.4065	53.4814	1313.6688
Tee_*n*	3.1344	2.425	14.8542
Tee_*T*	93.2033	43.1337	920.3958
Tee_*D*	114.3336	41.1956	1322.6335
Reducer_*n*	0.7705	0.6419	2.8958
Reducer_*T*	22.9964	9.9685	238.2292
Reducer_*D*	30.8026	7.9494	408.356
Coupling_*n*	0.0511	0.0467	0.125
Coupling_*T*	1.2545	0.739	9.7708
Coupling_*D*	0.0963	0.0794	0.375

Table F- 2 Feature vectors centroids of pipeline components generated from the generated data

Generated data	Cluster #		
Attribute	**Full data**	**0**	**1**
	841	**67**	**774**
Pcomponent_*n*	0.7491	2.5821	0.5904
Pcomponent_*T*	13.2033	93.7761	6.2287
Pcomponent_*D*	32.2485	328.8507	6.5736
Instrument_*n*	0.1807	0.7313	0.1331
Instrument_*T*	6.8288	53.3134	2.8049
Instrument_*D*	20.0452	240.4627	0.9651
Valve_*n*	1.6623	5.9403	1.292
Valve_*T*	44.2973	329.597	19.6008
Valve_*D*	88.5731	762.3881	30.2455
Flange_*n*	2.8145	8.0896	2.3579
Flange_*T*	59.2235	396.6269	30.0168
Flange_*D*	103.7277	837.0448	40.2494
Tube_*n*	10.9952	39.2388	8.5504
Tube_*T*	290.5208	2048.403	138.3527
Tube_*D*	95.5505	707.6418	42.5659
Elbow_*n*	5.9501	21.2687	4.624
Elbow_*T*	165.9263	1179.8209	78.1602
Elbow_*D*	94.805	706.7463	41.8333
Tee_*n*	1.2747	5.2687	0.9289
Tee_*T*	33.1641	260.209	13.5103
Tee_*D*	71.8478	681.7612	19.0517
Reducer_*n*	0.6052	1.8955	0.4935
Reducer_*T*	13.2663	78.8507	7.5891
Reducer_*D*	35.6052	360.4328	7.4871
Coupling_*n*	0.0464	0.1791	0.0349
Coupling_*T*	1.2913	9.3582	0.593
Coupling_*D*	0.8347	10.4776	0

Appendix G.

Table G- 1 21 influential critical success factors that leads to an effective use of modularization [22]

No.	Critical Success Factor	Definition	Impact Rate
1	Module Envelope Limitations	Preliminary transportation evaluation should result in understanding module envelope limitations.	3.83
2	Alignment on Drivers	Owner, consultants, and critical stakeholders should be aligned on important project drivers as early as possible in order to establish the foundation for a modular approach.	3.79
3	Owner's Planning Resources and Processes	As a potentially viable option to conventional stick building, early modular feasibility analysis is supported by owner's front-end planning and decision support systems, work processes, and team resources support	3.58
4	Timely Design Freeze	Owner and contractor are disciplined enough to effectively implement timely staged design freezes so that modularization can proceed as planned.	3.58
5	Early Completion Recognition	Modularization business cases should recognize and incorporate the economic benefits from early project completion that result from modularization and those resulting from minimal site presence and reduction of risk of schedule overrun.	3.42
6	Preliminary Module Definition	Front-end planners and designers need to know how to effectively define scope of modules in a timely fashion	3.42
7	Owner-Furnished/Long Lead Equipment Specification	Owner-furnished and long-lead equipment (OFE) specification and delivery lead time should support a modular approach.	3.42
8	Cost Savings Recognition	Modularization business case should incorporate all cost savings that can accrue from the modular approach. Project teams should avoid the knee-jerk misperception that modularization always has a net cost increase.	3.42

No.	Critical Success Factor	Definition	Impact Rate
9	Contractor Leadership	Front-end contractor(s) should be proactive—supporting the modular approach on a timely basis and prompting owner support, when owner has yet to initiate.	3.39
10	Contractor Experience	Contractors (supporting all phases) have sufficient previous project experience with the modular approach.	3.37
11	Module Fabricator Capability	Available, well equipped module-fabricators have adequate craft, skilled in high-quality/tight-tolerance modular fabrication.	3.37
12	Investment in Studies	In order to capture the full benefit, owner should be willing to invest in early studies into modularization opportunities.	3.32
13	Heavy Lift/Site Transport Capabilities	Necessary heavy lift/site transport equipment and associated planning/ execution skills are available and cost competitive.	3.32
14	Vendor Involvement	Original Equipment Manufacturer (OEMs) and technology partners need to be integrated into the modularized solution process in order to maximize related beneficial opportunities.	3.28
15	Operations and Maintenance (O&M) Provisions	Module detailed designs should incorporate and maintain established O&M space/access needs.	3.26
16	Transport Infrastructure	Needed local transport infrastructure is available or can be upgraded/modified in a timely fashion while remaining cost competitive	3.22
17	Owner Delay Avoidance	Owner has sufficient resources and discipline to be able to avoid delays in commitments on commercial contracts, technical scope, and finance matters.	3.16
18	Data for Optimization	Owner and Pre-FEED/ FEED contractor(s) need to have management tools/data to determine the optimal extent of modularization, i.e., maximum net present value (NPV) (that considers early revenue streams) versus % modularization	3.05
19	Continuity through Project Phases	Disconnects should be avoided in any contractual transition between Assessment, Selection, Basic Design, or Detailed Design phases; their impacts can be amplified with modularization.	3.0

No.	Critical Success Factor	Definition	Impact Rate
20	Management of Execution Risks	Project risk managers need to be prepared to deal with any risks shifted from the field to engineering/procurement functions.	3.0
21	Transport Delay Avoidance	Environmental factors such as hurricanes, frozen seas, or lack of permafrost, in conjunction with fabrication shop schedules, do not result in any significant project delay.	3.0

Appendix H.

Table H- 1 Pipe spooling solution and CPU run time for each pipeline instance problem

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
1	13	2	1186	2	1249
2	10	1	516	2	640
3	8	1	750	4	1181
4	14	1	1544	3	1790
5	4	1	767	1	827
6	17	5	4188	7	5475
7	7	1	1370	1	1548
8	40	6	13796	10	16516
9	43	7	14610	12	18354
10	3	1	98	1	97
11	61	5	6411	8	5984
12	20	6	3057	11	3334
13	14	2	979	2	1001
14	8	1	402	1	349
15	8	1	355	5	528
16	8	1	425	2	478
17	18	4	1831	6	1863
18	28	3	3544	6	3523
19	20	2	4938	5	5013
20	30	3	1954	7	2110
21	38	3	5603	8	6257
22	12	1	719	2	952
23	13	2	1743	2	1884
24	49	10	8451	22	12808
25	11	1	721	4	1111
26	29	1	2581	4	3277
27	17	1	1223	2	1505
28	35	3	2851	8	3767
29	6	1	561	5	1193
30	52	7	7607	16	9779
31	7	1	631	1	745
32	9	1	907	3	1119
33	31	10	2233	12	2726
34	37	5	3863	9	4890
35	8	1	754	1	852

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
36	18	2	2637	3	3056
37	3	1	431	1	518
38	38	6	9106	7	9963
39	41	5	5561	14	7686
40	23	3	3223	10	5072
41	10	2	2096	2	2208
42	25	2	4173	4	4736
43	28	2	4617	5	5306
44	22	2	4088	5	5101
45	12	1	1569	3	2168
46	25	1	4113	3	4936
47	46	5	9024	9	10698
48	87	15	33690	25	37887
49	49	3	8491	7	10328
50	12	1	1751	2	2337
51	25	4	4425	7	5760
52	43	6	12397	9	14316
53	20	2	3716	3	4500
54	28	2	5171	3	5973
55	22	2	4284	6	5725
56	31	3	6676	6	8226
57	25	3	4066	3	4741
58	81	9	30906	18	35114
59	26	2	5806	3	6607
60	51	6	10984	10	12983
61	23	2	4198	4	5430
62	28	2	5190	5	6747
63	37	5	7679	19	13723
64	29	2	5481	5	7110
65	12	1	1973	3	2744
66	3	1	750	2	1210
67	10	1	2374	3	3168
68	42	4	8878	10	11717
69	32	3	6847	6	8409
70	10	3	2516	3	2785
71	42	4	10757	12	14813
72	11	1	2420	4	3689
73	17	3	4897	7	7398
74	7	1	1519	1	1711
75	39	7	10994	9	13217

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
76	21	2	4839	2	5274
77	13	2	2891	4	3770
78	18	1	3784	4	5500
79	3	1	812	1	968
80	13	1	2719	2	3738
81	15	1	3317	4	5127
82	8	2	2163	2	2442
83	3	1	971	1	1060
84	54	12	29671	14	29631
85	18	1	4135	1	4783
86	17	1	3591	2	4483
87	83	11	36651	17	41757
88	41	4	13542	12	18481
89	67	6	21020	10	24255
90	9	2	2963	2	3515
91	7	1	1824	1	1948
92	35	3	10178	10	14450
93	60	11	23694	19	29182
94	23	3	7749	4	9203
95	2	1	829	1	964
96	48	6	16184	14	22003
97	34	6	11794	10	15066
98	61	4	21120	16	29806
99	8	1	2205	5	4688
100	24	1	586	3	805
101	17	1	507	2	567
102	58	6	9483	12	9666
103	11	2	544	2	526
104	9	1	361	2	344
105	60	4	8511	9	8662
106	14	3	983	3	1024
107	10	1	315	6	377
108	10	1	229	4	418
109	15	1	930	5	605
110	88	14	13855	26	14988
111	39	4	3425	10	3323
112	17	2	1237	4	1446
113	24	1	1843	5	2061
114	40	2	3267	5	3376
115	23	2	1952	4	1987

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
116	13	2	1130	5	1147
117	61	7	8812	18	9479
118	19	3	4500	5	4611
119	14	1	536	1	391
120	12	1	1506	4	1453
121	55	4	4526	13	4906
122	11	1	633	1	510
123	15	2	1423	2	1315
124	14	1	551	3	573
125	9	1	329	1	325
126	11	1	404	3	520
127	66	6	2858	9	3067
128	93	10	17104	23	21132
129	18	2	1108	2	1433
130	4	1	205	1	170
131	17	4	670	8	920
132	15	2	1527	2	1542
133	7	1	731	5	921
134	7	1	620	1	542
135	18	1	1114	4	1203
136	6	1	509	1	403
137	4	1	197	1	200
138	72	11	13872	21	14354
139	25	3	2490	4	2806
140	13	2	845	4	860
141	29	3	1225	10	1495
142	24	3	3677	5	3859
143	48	6	11144	14	13981
144	29	4	2492	11	3232
145	21	1	1374	2	1443
146	14	1	540	3	559
147	25	2	2244	6	2498
148	8	1	375	1	406
149	118	10	28031	24	29046
150	11	1	441	3	583
151	188	26	49154	53	53786
152	54	7	7452	10	7507
153	15	1	715	1	507
154	29	4	2790	10	2837
155	22	1	1421	4	1607

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
156	18	2	1252	5	1347
157	30	3	1428	5	1503
158	63	8	12508	12	13117
159	3	1	164	1	205
160	32	5	4639	5	4601
161	60	1	8477	1	8676
162	12	1	1854	4	1990
163	2	1	157	1	147
164	9	1	422	4	614
165	36	3	3423	7	3547
166	10	1	661	1	627
167	24	2	1672	6	1913
168	35	3	13792	8	14203
169	37	5	4543	7	4458
170	8	1	362	1	398
171	55	5	6238	9	6696
172	13	2	932	3	858
173	30	3	2902	8	2918
174	124	12	30951	23	31884
175	7	1	394	1	419
176	64	5	7453	13	7927
177	17	3	1739	6	1920
178	32	3	3199	8	3529
179	19	1	4305	3	4453
180	24	3	1757	7	2182
181	88	9	12506	13	13169
182	7	1	377	1	336
183	14	2	1507	2	1323
184	9	2	720	2	716
185	30	3	3639	5	3745
186	8	1	461	1	469
187	76	9	7637	16	8271
188	41	5	5819	8	6209
189	98	7	18177	23	20119
190	14	1	2170	1	2244
191	62	5	4864	15	5938
192	5	1	307	1	311
193	10	1	638	1	651
194	14	1	1038	1	1121
195	19	3	2058	8	2340

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
196	41	3	5077	8	5596
197	11	2	1005	2	837
198	58	6	9853	12	10616
199	19	2	2070	3	2265
200	18	1	1799	3	2092
201	35	3	2967	4	2912
202	74	10	20577	16	21629
203	4	1	261	1	273
204	33	4	4942	9	5630
205	11	3	1090	3	1351
206	61	8	9785	16	10905
207	38	3	4910	4	5132
208	9	1	550	1	547
209	5	2	547	2	655
210	40	4	3442	9	4082
211	6	2	553	2	552
212	26	3	1489	9	2221
213	34	4	3686	7	4057
214	5	1	421	1	470
215	14	1	911	3	1087
216	47	6	4113	13	5116
217	18	1	1804	5	2247
218	20	1	1189	2	1579
219	3	1	296	1	264
220	48	9	4989	19	5403
221	45	14	3602	23	4240
222	37	4	6257	5	6658
223	18	2	1549	8	2178
224	30	1	2597	5	3053
225	46	4	4091	16	5398
226	3	1	311	1	257
227	7	1	466	1	440
228	76	7	10031	13	10894
229	24	3	2685	4	2989
230	68	6	14979	9	15333
231	42	4	4643	9	5501
232	15	1	1573	2	1787
233	4	1	373	1	331
234	29	4	3391	5	3670
235	26	1	1626	3	2045

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
236	5	1	434	1	380
237	5	3	498	4	684
238	27	3	4295	5	4562
239	61	6	12373	9	12761
240	11	1	676	3	882
241	9	1	820	1	870
242	20	2	2220	5	2463
243	5	1	391	1	417
244	17	2	1293	6	1703
245	26	1	2345	4	2767
246	48	5	3622	12	4396
247	38	1	3990	3	4537
248	35	4	5915	7	6269
249	14	1	968	1	1088
250	22	1	2988	3	3304
251	16	1	1514	3	1813
252	13	1	751	1	868
253	195	22	48361	36	53962
254	15	1	1271	4	1638
255	25	1	2342	3	2739
256	11	1	1029	4	1328
257	124	9	30422	24	33729
258	7	1	790	1	821
259	14	1	1082	2	1251
260	117	22	23829	28	24351
261	53	7	8709	11	9538
262	3	1	293	1	319
263	19	2	1820	3	2035
264	8	1	627	3	895
265	30	2	3367	6	4065
266	17	1	1192	3	1634
267	16	3	1802	5	2085
268	16	1	1192	4	1626
269	13	1	1137	2	1232
270	4	1	380	1	419
271	15	1	2949	3	3449
272	107	11	35810	26	38613
273	6	1	464	1	494
274	38	3	5495	9	6507
275	96	11	14296	23	15908

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
276	38	6	4856	10	5895
277	7	1	605	1	655
278	8	1	672	2	841
279	12	1	839	1	880
280	84	8	8898	15	10222
281	48	6	10358	12	11417
282	44	4	6530	8	7170
283	12	1	1244	3	1398
284	19	3	2456	5	3028
285	11	2	1103	2	1244
286	7	1	588	1	603
287	31	3	5292	8	6272
288	50	8	6804	12	7595
289	10	1	708	1	823
290	17	3	1546	11	2817
291	26	1	9310	3	10208
292	46	3	7753	8	8589
293	18	2	1521	6	2103
294	37	6	6551	7	7371
295	128	15	20721	33	26272
296	16	3	1970	4	2236
297	61	11	16663	13	17323
298	83	6	19752	14	21273
299	6	1	788	1	851
300	34	5	5915	8	6862
301	7	1	565	1	632
302	50	4	6709	10	7838
303	33	4	3842	6	4183
304	19	3	2973	3	3076
305	138	24	51578	41	65044
306	30	5	7663	14	12693
307	3	1	366	1	371
308	7	1	642	1	624
309	23	1	2948	3	3436
310	7	1	667	1	611
311	32	5	7378	8	7966
312	38	5	6153	13	7672
313	15	1	1276	3	1610
314	5	1	737	2	744
315	49	6	9881	7	10229

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
316	31	2	3594	6	4235
317	2	1	320	1	344
318	16	1	1240	2	1430
319	23	3	2645	6	3364
320	34	4	3767	9	4859
321	13	1	1456	1	1275
322	11	2	1374	2	1324
323	23	2	2537	5	3011
324	62	6	11355	11	12572
325	10	1	849	1	923
326	18	1	2311	3	2762
327	24	3	3220	8	4360
328	22	1	2490	3	2693
329	7	1	727	1	733
330	51	7	5477	15	6349
331	29	7	7655	11	8562
332	3	1	357	1	384
333	35	6	7506	12	9072
334	32	3	4454	5	5076
335	35	2	3963	6	4659
336	7	2	911	2	967
337	4	1	432	1	488
338	13	3	1561	5	1882
339	53	6	10030	14	11700
340	6	2	821	2	860
341	56	7	14110	11	15530
342	24	2	3332	3	3778
343	35	4	5841	8	6656
344	46	4	7692	11	8994
345	28	4	4791	5	5339
346	16	1	1507	2	1754
347	19	1	1677	3	2119
348	3	1	441	1	461
349	26	6	7079	6	7341
350	28	3	11518	4	11989
351	26	1	3398	3	4269
352	39	4	5365	9	6183
353	3	1	368	1	402
354	18	2	3018	4	3654
355	81	9	22383	20	24636

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
356	42	4	6026	10	7725
357	5	1	503	1	569
358	31	4	4118	7	4719
359	8	1	1032	1	1109
360	14	1	1177	3	1868
361	34	3	4212	6	5152
362	40	3	5714	11	7552
363	31	3	4539	5	5079
364	30	4	5337	7	6075
365	20	2	2975	4	3514
366	24	3	3740	6	4012
367	26	2	4528	5	5046
368	37	3	3847	9	5191
369	22	1	2613	3	3102
370	26	1	3784	6	4754
371	33	3	4686	6	5509
372	42	4	6481	7	7291
373	77	8	15318	20	18178
374	34	3	4312	8	5714
375	53	8	13878	8	14553
376	32	2	5340	6	6566
377	26	5	3600	8	4512
378	10	1	1826	1	1899
379	7	1	744	1	806
380	19	1	2665	3	3241
381	12	1	1562	1	1488
382	35	3	3725	12	5642
383	38	3	4885	4	5392
384	31	3	5527	8	6860
385	25	1	2968	3	3803
386	18	1	4775	2	5158
387	62	7	11377	12	12801
388	12	1	1139	3	1618
389	40	2	6146	5	6986
390	4	1	483	1	557
391	13	2	1623	2	1959
392	33	4	4582	5	5244
393	27	2	3489	4	4207
394	34	1	4083	4	5251
395	33	5	6430	10	7809

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
396	13	2	2054	2	2261
397	20	4	3518	6	4041
398	19	1	2485	8	4056
399	31	4	5201	11	6680
400	32	3	5115	6	6055
401	11	2	1270	3	1635
402	28	2	4803	5	5731
403	19	3	3522	4	3965
404	13	1	1732	3	2309
405	24	2	2835	8	4193
406	3	1	435	1	467
407	31	11	4195	16	4628
408	25	3	4613	3	4779
409	43	4	6029	7	7105
410	32	2	5607	7	6853
411	24	2	3931	7	5060
412	15	1	1687	1	1785
413	34	3	5633	7	6721
414	141	18	55143	40	64202
415	7	1	804	2	1134
416	86	9	17461	15	20338
417	23	1	2309	5	3514
418	4	1	672	1	590
419	20	2	3320	3	3647
420	10	1	1587	5	2467
421	10	2	1599	2	1818
422	12	1	2674	3	3168
423	24	3	3176	7	4223
424	20	2	2252	4	3030
425	21	4	2846	4	3029
426	30	2	3619	9	5345
427	58	5	13652	9	14639
428	20	1	2690	7	4065
429	11	1	1263	2	1477
430	4	1	545	1	611
431	10	3	1496	2	1500
432	15	1	1711	4	2496
433	42	1	5573	9	7133
434	17	3	2849	3	2773
435	30	6	6079	12	8061

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
436	19	1	2060	3	2750
437	35	3	4348	4	4860
438	27	5	6130	8	7052
439	71	8	19619	16	23106
440	7	1	888	1	924
441	37	5	8987	12	11033
442	19	2	3659	5	4459
443	61	11	13754	23	17649
444	12	3	3089	3	3346
445	21	2	3952	4	4569
446	25	1	3390	2	4045
447	18	2	2715	6	3676
448	21	3	3832	8	5165
449	33	6	5430	9	6673
450	13	1	1407	3	2176
451	10	1	1162	1	1385
452	17	1	2387	3	2997
453	11	1	1267	1	1757
454	40	6	6485	15	9548
455	29	2	4809	2	5247
456	30	3	5173	8	6814
457	9	1	1329	2	1595
458	52	6	10712	12	12497
459	17	1	2414	2	2910
460	13	1	1918	2	2042
461	20	1	2517	3	3015
462	5	1	802	1	739
463	33	6	6229	12	8642
464	63	5	14902	8	16476
465	14	2	2501	4	2698
466	18	1	2152	4	3088
467	61	6	14827	11	16838
468	150	16	43128	38	52382
469	46	2	9489	5	10982
470	21	1	2879	2	3702
471	39	5	7492	9	8736
472	27	3	4753	4	5251
473	31	2	5283	5	6242
474	34	3	5994	8	7580
475	36	3	6490	7	7603

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
476	13	2	2491	2	2548
477	14	1	1853	3	2480
478	5	1	911	1	921
479	21	3	3767	3	4498
480	13	2	2012	3	2579
481	16	4	3350	4	3662
482	29	4	6830	8	8239
483	31	2	4762	7	6344
484	2	1	69	1	69
485	30	3	5466	10	7509
486	27	3	4669	2	5148
487	15	1	2121	3	2787
488	51	7	8427	9	10131
489	3	1	514	1	610
490	9	1	1885	1	1936
491	50	5	8097	12	10645
492	69	8	13073	17	15907
493	26	1	4928	3	5625
494	39	3	7967	8	9365
495	12	2	2265	2	2121
496	18	4	3332	8	4870
497	21	1	2592	6	4151
498	30	5	7282	6	8125
499	32	3	6444	6	7596
500	26	5	6358	12	8963
501	9	1	1470	1	1503
502	43	4	9777	7	11165
503	18	1	2668	2	3251
504	22	2	4010	4	4711
505	2	1	435	1	499
506	23	3	4443	6	5558
507	24	3	5263	3	5249
508	39	1	5724	8	8664
509	102	10	27771	24	33197
510	24	2	3916	4	4744
511	24	4	4712	12	7197
512	29	2	5942	2	6290
513	21	1	2739	3	3713
514	30	2	6638	5	7712
515	14	1	2083	3	2933

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
516	22	2	4658	3	5224
517	14	1	1790	1	2166
518	47	5	10861	15	15268
519	17	4	4324	8	6403
520	16	3	4977	7	6409
521	12	3	3537	5	4433
522	52	6	11134	6	12233
523	27	11	2907	14	3052
524	86	10	22790	25	31442
525	13	1	1882	4	3194
526	33	5	8852	11	11717
527	21	1	3336	2	3988
528	8	1	1281	1	1319
529	34	5	8914	5	9834
530	6	1	950	1	1030
531	16	1	2213	2	2769
532	10	1	2107	3	2712
533	18	2	3644	2	4430
534	17	1	2753	3	3557
535	22	4	5636	4	6091
536	18	2	2630	5	3870
537	39	5	9501	11	12096
538	27	3	4051	4	5189
539	16	4	3600	6	4760
540	38	5	7653	11	10114
541	152	21	70005	35	79834
542	7	1	1041	1	1163
543	20	2	3584	2	3830
544	79	7	22131	18	27366
545	3	1	580	1	666
546	32	3	7005	8	9000
547	33	4	7589	4	8340
548	57	5	13605	13	16974
549	31	4	4487	10	6952
550	10	1	1682	2	2144
551	84	10	24329	18	28226
552	8	1	1228	3	1895
553	54	6	13255	11	15485
554	9	1	1955	1	2106
555	12	1	1693	1	1945

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
556	4	1	755	1	980
557	28	6	5056	10	6887
558	24	3	6031	5	6932
559	28	2	5374	5	6976
560	26	5	5072	8	6909
561	19	4	4072	6	5250
562	58	6	18230	9	20130
563	27	3	4331	6	5924
564	19	2	3397	6	5047
565	59	7	17239	14	20713
566	23	2	4694	3	5308
567	32	4	8096	7	9911
568	45	4	11997	7	14069
569	65	8	15125	13	18050
570	54	5	12091	7	14128
571	5	1	844	1	982
572	6	1	1057	2	1472
573	51	4	12648	8	14458
574	3	1	627	1	697
575	4	1	945	1	1062
576	9	11	770	14	821
577	19	1	3249	5	4949
578	13	3	3118	3	3359
579	18	2	3860	3	4137
580	30	4	6281	5	7198
581	105	8	37945	19	43189
582	4	1	740	1	865
583	31	2	6257	7	8151
584	15	2	3567	2	3673
585	12	1	1744	4	2741
586	23	3	5502	4	6197
587	15	1	2472	3	3314
588	27	4	5424	7	6962
589	34	5	8863	8	10623
590	30	3	5777	5	6812
591	51	6	18086	17	21723
592	37	2	7538	7	10269
593	37	5	9640	6	11263
594	21	5	4084	8	17894
595	59	3	15685	10	18680

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
596	14	1	2427	2	2877
597	17	3	3937	5	4810
598	29	1	5504	8	8348
599	24	4	6618	8	8296
600	16	2	3153	2	3476
601	14	1	2908	1	2922
602	22	1	4318	4	5814
603	43	4	11010	8	12877
604	8	3	2544	6	3809
605	39	4	9386	13	13015
606	9	1	1663	1	1855
607	25	2	5626	7	8039
608	14	2	2873	2	3349
609	57	5	11039	9	13596
610	15	1	2288	3	3353
611	10	1	2048	1	2185
612	18	2	3924	4	4871
613	10	1	1792	6	4031
614	9	1	1495	1	1701
615	21	1	7447	2	8560
616	5	1	961	1	1106
617	14	1	2515	1	2930
618	11	1	1766	1	2153
619	34	2	6950	7	9368
620	18	2	3632	6	5330
621	14	2	2854	4	3840
622	21	2	4002	2	4698
623	9	2	1924	4	2974
624	19	1	3500	3	4523
625	18	2	3066	4	4080
626	12	2	2815	2	3127
627	9	1	2354	1	2591
628	24	1	5471	2	6195
629	18	3	4654	4	5408
630	24	3	4489	8	6620
631	40	7	11392	10	13345
632	38	3	9190	7	11187
633	37	3	7724	5	9632
634	22	2	4209	6	6138
635	21	2	4173	5	5582

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
636	53	2	10552	5	13207
637	16	1	2631	3	3540
638	26	3	6917	4	8206
639	88	8	28711	19	34826
640	56	5	16998	9	20050
641	9	1	1580	1	1717
642	31	6	7214	16	12845
643	8	1	1454	1	1678
644	25	1	4030	5	5888
645	23	2	5043	4	6240
646	9	2	2035	2	2411
647	29	3	6517	4	7683
648	23	3	4290	6	6168
649	24	1	3770	5	5720
650	58	6	13237	16	18130
651	6	1	1111	1	1294
652	51	4	12305	9	15572
653	4	1	1081	1	1119
654	50	6	10727	10	13297
655	10	2	2189	2	2415
656	10	1	1708	1	2036
657	7	1	1340	4	2615
658	6	3	1679	3	1873
659	5	1	973	1	1112
660	17	2	3603	5	4923
661	19	2	4091	2	4326
662	9	1	1586	2	2108
663	20	3	3980	10	6823
664	85	8	21790	14	25053
665	89	6	25380	14	29882
666	11	9	1798	15	2546
667	19	2	3964	6	5951
668	58	7	18513	17	23717
669	7	1	1561	1	1711
670	43	6	12335	10	14565
671	19	2	4078	4	5123
672	29	2	7092	3	8187
673	34	5	8682	5	8933
674	29	3	6770	6	8299
675	86	10	31624	25	39147

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
676	3	1	710	1	809
677	11	1	2189	1	2424
678	31	3	5953	7	7842
679	16	3	4403	4	5114
680	20	4	5135	10	8256
681	266	34	72852	60	90921
682	7	1	1410	2	1857
683	32	4	6150	8	8493
684	25	3	5926	6	7535
685	16	1	2710	4	4151
686	2	1	680	1	666
687	18	2	3820	4	4885
688	16	1	3031	4	4413
689	68	5	16885	11	20615
690	46	4	11846	9	14606
691	63	10	18667	20	24980
692	5	1	1194	1	1252
693	15	2	3202	3	4018
694	21	3	4851	8	7693
695	9	1	2074	1	2304
696	18	1	3200	1	3659
697	20	3	4021	3	4525
698	28	3	6872	7	8882
699	36	6	10650	8	12624
700	47	4	12159	8	14779
701	9	1	1907	1	2118
702	44	5	12331	6	14420
703	5	1	1050	1	1211
704	33	4	6944	13	11342
705	32	4	7127	8	9576
706	6	1	1315	1	1499
707	15	2	3654	5	5334
708	27	5	7297	9	9683
709	24	2	5644	4	6833
710	10	1	1959	1	2053
711	52	4	12214	12	16401
712	6	1	1249	1	1378
713	20	2	5375	2	5777
714	50	6	13566	12	16904
715	10	1	1823	3	2724

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
716	16	3	4318	8	6810
717	13	1	2273	3	3234
718	15	1	4641	1	4985
719	10	2	2445	2	2797
720	21	2	4994	2	5513
721	83	5	29191	17	34465
722	25	4	6451	7	8227
723	27	2	5558	5	7480
724	31	2	6903	3	8435
725	40	3	11059	6	13268
726	8	1	1612	4	2978
727	9	1	1764	1	2071
728	19	3	4521	3	5140
729	19	2	4269	2	4846
730	15	1	4076	3	5288
731	29	6	13008	11	16804
732	45	3	8425	11	12614
733	31	2	7362	8	10059
734	8	1	1762	1	1774
735	32	4	9394	8	11494
736	23	2	6859	3	7494
737	10	1	1881	2	2517
738	33	5	9312	14	14244
739	21	3	4612	3	5108
740	18	3	4081	4	4885
741	6	1	1294	4	2677
742	6	1	1327	1	1484
743	30	4	7752	5	9170
744	8	1	1689	1	1943
745	121	13	58441	28	69789
746	23	3	4816	6	7051
747	12	1	2157	3	3250
748	6	1	1275	1	1486
749	31	2	7441	6	9420
750	9	1	1987	3	2793
751	40	4	11137	12	15185
752	18	3	6865	3	7352
753	34	4	8093	15	13460
754	25	1	4651	5	6915
755	33	5	9493	8	11618

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
756	5	1	1147	1	1279
757	33	1	9739	5	11910
758	5	1	1193	5	2976
759	36	3	9608	7	11907
760	27	4	7869	8	10378
761	22	4	5598	9	8192
762	8	1	1647	1	1809
763	30	5	7583	10	10401
764	13	1	2545	4	4165
765	20	1	4026	5	5549
766	33	2	7857	7	10637
767	21	2	5339	3	6022
768	84	7	21504	18	29796
769	4	1	966	1	1079
770	116	12	33325	27	42730
771	57	7	21131	13	25031
772	32	3	7287	3	8105
773	50	9	19514	13	23658
774	20	3	4544	6	6094
775	31	2	6750	6	9167
776	20	4	5210	9	8110
777	2	1	647	1	739
778	48	3	12850	12	17940
779	14	1	3192	2	3750
780	22	3	4757	7	7537
781	19	3	4819	4	5695
782	14	1	3049	2	3791
783	8	2	2363	2	2464
784	18	2	4506	6	6829
785	6	1	1321	1	1521
786	10	2	2770	2	3052
787	22	1	4191	3	5557
788	34	1	7198	3	8765
789	35	3	8898	7	11729
790	9	1	2543	1	2717
791	61	9	24517	19	30758
792	12	1	2320	2	3098
793	7	1	1610	1	1693
794	26	4	6873	8	9360
795	13	1	2489	1	2883

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
796	36	3	10245	11	14666
797	5	1	1171	4	2621
798	23	3	6829	3	7662
799	19	3	5550	6	7433
800	27	2	6409	6	8244
801	22	4	5990	10	9550
802	18	3	4407	7	6594
803	18	3	4227	8	6753
804	42	5	13254	7	15134
805	3	1	959	1	979
806	15	1	3476	3	4545
807	13	3	3598	6	5028
808	5	1	1356	1	1447
809	21	1	4821	2	5354
810	22	3	6587	3	7150
811	17	1	3506	3	4628
812	61	5	14006	13	18934
813	6	1	1340	1	1578
814	22	1	4837	2	5770
815	7	1	1927	3	2835
816	12	1	2610	4	4367
817	142	14	72187	28	82434
818	11	1	2570	1	2678
819	13	2	3581	7	6259
820	20	5	6502	11	10253
821	4	1	1051	1	1236
822	21	1	5198	2	6450
823	56	5	20867	9	24266
824	9	2	2483	2	2899
825	14	1	3590	1	3867
826	27	1	5637	3	6601
827	11	1	2368	3	3310
828	50	4	14359	11	19320
829	4	1	1088	1	1249
830	3	1	901	1	1070
831	38	5	11808	11	15144
832	39	4	11324	6	13694
833	23	5	7619	8	9665
834	23	3	7552	5	9197
835	24	4	7061	12	11526

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
836	15	2	3859	5	5550
837	17	1	3770	3	5267
838	21	5	7068	5	8134
839	7	1	1657	1	1805
840	11	1	3326	1	3641
841	58	7	19962	14	24449
842	5	1	1270	1	1416
843	14	2	3437	4	5273
844	24	3	6527	3	7800
845	96	6	33099	15	40255
846	23	3	6896	4	8365
847	14	1	2886	3	4304
848	10	1	2166	1	2522
849	21	3	5217	5	6808
850	43	6	14271	11	18332
851	6	1	1545	1	1774
852	23	2	6330	5	8742
853	13	1	3092	5	5571
854	5	1	1314	1	1379
855	34	4	9550	9	13439
856	27	2	6649	4	8316
857	14	1	3258	1	3678
858	23	1	5389	3	6885
859	16	3	5218	3	5846
860	7	1	1725	3	2569
861	5	1	1281	1	1518
862	54	5	18615	8	21183
863	56	6	21445	10	25508
864	16	4	4457	4	5209
865	21	5	6072	9	9160
866	3	1	894	1	1032
867	10	1	2311	3	3372
868	19	2	4567	5	6267
869	70	8	21483	17	27126
870	18	1	4648	3	6135
871	10	1	2218	1	2581
872	42	4	14589	8	17792
873	20	1	4908	3	6026
874	29	7	5602	11	7501
875	35	4	9985	6	12000

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
876	10	1	3089	3	4380
877	17	1	4207	1	4567
878	115	13	45484	30	56696
879	13	2	3800	2	4331
880	8	1	2022	3	2953
881	93	9	27477	31	42522
882	6	1	1496	1	1652
883	17	1	3546	3	5174
884	57	4	16105	13	22793
885	21	1	5626	5	8029
886	7	1	1734	2	2342
887	26	1	6258	3	7900
888	59	11	4077	13	5136
889	13	1	2864	2	3885
890	31	1	8058	4	10123
891	67	10	24681	18	31472
892	22	2	5909	4	7205
893	47	6	16952	11	20737
894	17	3	4794	4	5984
895	13	1	2765	1	3409
896	108	11	43664	23	52006
897	20	1	5052	4	6701
898	21	2	5561	2	5856
899	25	5	7452	10	10871
900	19	4	5731	4	5979
901	36	6	13474	12	18072
902	37	8	12720	19	21126
903	6	1	1492	1	1784
904	18	1	3918	2	5274
905	28	2	7611	5	10128
906	21	5	6990	11	11027
907	13	2	3609	2	4161
908	32	2	8216	10	13134
909	54	5	21508	9	24848
910	8	1	2003	2	2705
911	9	1	2198	1	2508
912	24	4	8152	6	9796
913	24	1	5734	3	7494
914	45	4	13237	11	17754
915	29	3	9315	6	11485

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
916	22	2	5849	4	7154
917	56	3	18307	13	23922
918	16	1	4549	6	7311
919	23	2	6864	8	10239
920	10	1	2890	1	3187
921	5	1	1574	4	3203
922	15	1	3525	5	6202
923	140	14	67796	31	83108
924	9	1	2196	1	2531
925	12	1	2718	1	3214
926	14	1	3164	4	5110
927	23	1	5085	3	6794
928	18	2	5267	4	7020
929	12	4	5146	4	5547
930	46	8	18778	15	23540
931	22	3	5670	5	7776
932	18	2	5260	2	5868
933	12	2	3147	3	4221
934	7	1	2081	2	2887
935	21	4	7967	5	9035
936	5	1	1364	1	1613
937	24	2	8272	8	11813
938	28	3	8912	8	12524
939	115	19	61397	26	67960
940	9	1	3084	5	5835
941	3	1	1106	1	1170
942	81	8	34906	16	41199
943	24	3	6285	2	6830
944	16	2	6613	3	7456
945	20	1	4990	2	6207
946	19	1	4297	2	5490
947	42	7	16505	13	21066
948	10	2	3057	2	3451
949	8	1	2115	3	3324
950	117	13	58465	22	67269
951	2	1	839	1	944
952	6	4	3232	6	4933
953	8	1	2024	1	2340
954	41	6	14270	13	19790
955	115	9	36575	27	49320

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
956	32	2	9116	7	12493
957	4	1	1247	4	3052
958	31	6	11331	15	17926
959	18	4	5614	8	8525
960	12	3	3615	3	4184
961	16	4	5298	5	6697
962	13	1	3037	3	4494
963	124	17	49891	32	61140
964	36	6	15600	10	18730
965	19	1	4898	3	6446
966	39	3	11065	10	16051
967	5	1	1466	1	1631
968	34	3	10998	8	14957
969	26	2	7615	3	9099
970	28	3	9264	6	11633
971	42	4	13660	10	18573
972	11	3	3987	5	6061
973	27	2	7721	7	11221
974	35	3	9880	9	14079
975	37	3	11828	7	15445
976	15	1	4070	5	6229
977	15	2	5665	2	6070
978	26	1	6067	3	8038
979	100	10	30392	24	39969
980	6	1	1807	1	2053
981	9	1	2413	1	2851
982	11	1	2734	1	3167
983	23	1	6494	4	8996
984	36	6	13392	12	18096
985	71	9	27721	12	32363
986	43	3	13904	8	17976
987	39	3	11820	5	14162
988	20	4	6804	9	11104
989	4	1	1293	1	1489
990	42	3	14495	8	18177
991	14	3	4471	3	4899
992	39	6	13237	15	20239
993	12	3	3821	8	7626
994	25	2	7508	4	9529
995	35	8	12200	16	19422

Pipeline No.	No. of Components	Alberta Size		Overseas Size	
		No. of Spools	Solution Time (ms)	No. of Spools	Solution Time (ms)
996	48	5	14409	11	19328
997	5	1	1531	1	1723
998	21	1	5736	5	9131
999	49	4	18621	12	25344
1000	107	14	22410	20	23037

Appendix I.

Industrial Pipeline Data Generator-Python Code

```
import numpy as np
import math
import os, sys
import pyodbc
import random
import sqlite3
from collections import deque
import csv
from scipy import stats
from scipy.stats import burr

def PipelineGenerator(number_of_pipelines):

   def starting_component_state_main_line():

     # Initialize the starting component in the main line(excluding start and finish
components)
     # This function is made to generate starting component, in case if the generator
stop
     initialize_starting_component=np.random.uniform(0,100)

     if initialize_starting_component <= 7.1:

           starting_component="REDUCER"
           return starting_component

     elif initialize_starting_component <= 18.9 :

           starting_component="VALVE"
           return starting_component

     elif initialize_starting_component <= 31.5:

           starting_component="TEE"
           return starting_component

     elif initialize_starting_component <= 47:

           starting_component="FLANGE"
           return starting_component

     elif initialize_starting_component <= 70:

           starting_component="ELBOW"
           return starting_component
```

```
      else:
        starting_component="TUBE"
        return starting_component

    def starting_component_state_branch():

      # Initialize the starting component in the branch(excluding start and finish
components)
      # This function is made to generate starting component, in case if the generator
stop

      initialize_starting_component=np.random.uniform(0,100)

      if initialize_starting_component <= 0.96:

              starting_component="PCOMPONENT"
              return starting_component

      elif initialize_starting_component <= 3.06 :

              starting_component="REDUCER"
              return starting_component

      elif initialize_starting_component <= 5.66:

              starting_component="INSTRUMENT"
              return starting_component

      elif initialize_starting_component <= 20.2:

              starting_component="TEE"
              return starting_component

      elif initialize_starting_component <= 31.0:

              starting_component="VALVE"
              return starting_component

      elif initialize_starting_component <= 44.0:

              starting_component="FLANGE"
              return starting_component

      elif initialize_starting_component <= 59.37:

              starting_component="ELBOW"
              return starting_component

      else:

        starting_component="TUBE"
        return starting_component


    ###############Components Diameter##############################
    c,d= 2.8574,0.76059

    mean, var, skew, kurt = burr.stats(c, d, moments='mvsk')

    rv = burr(c, d)
    #########################

    ######################## Main Line Generator ######################

    file = open("Pipelines_Data_Set.txt", "w")

file.write("Line_Number"+","+"Type_of_Branch"+","+"Component_No"+","+"Previosu
ly_Connected_to"+","+"Component_Type"+","+"Diameter"+","+"Length"+","+"Runnin
g_Direction"+"\n")

print("Line_Number",",","Type_of_Branch",",","Seq_in_branch",",","Previosuly_Conn
ected_to",",","Component_Type",",","Diameter",",","Length",",","Running_Direction")

    #(1)  Generate the number of components in the main line

    for line_no in range (1,int(number_of_pipelines)+1):

      # Initialize components first step value:

      cap_step=1
      instrument_step=1
      tube_step=1
      valve_step=1
      fblind_step=1
      ftube_step=1
      flange_step=1
      clousre_step=1
      pcomponent_step=1
      tee_step=1
      reducer_step=1
      coupling_step=1
      elbow_step=1

      component_number=1

      running_direction= "x"

      running_direction_list=deque()

      components_diameter = burr.rvs(c, d, loc=0, scale=124.3, size=1)

      no_of_main_line_components=int(math.ceil(np.random.gamma(2, 10.5)))
```

```
 if no_of_main_line_components <3:
  no_of_main_line_components=3
#(2)  Initialize the starting component of the main line
 initialize_starting_component=np.random.uniform(0,100)
 if initialize_starting_component <= 0.1:
     starting_component="TEE"
 elif initialize_starting_component <= 0.4:
     starting_component="FTUBE"
 elif initialize_starting_component <= 1.6:
     starting_component="ELBOW"
 elif initialize_starting_component <= 3.9:
     starting_component="INSTRUMENT"
 elif initialize_starting_component <= 6.3:
     starting_component="CAP"
 elif initialize_starting_component <= 13.0:
     starting_component="TUBE"
 elif initialize_starting_component <= 20.9:
     starting_component="VALVE"
 elif initialize_starting_component <= 30.2:
     starting_component="FBLIND"
 elif initialize_starting_component <= 48.5:
     starting_component="PCOMPONENT"
 elif initialize_starting_component <= 71.8:
     starting_component="FLANGE"
 else:
       starting_component="CLOSURE"

  # Use Markov-chain transition matrix and state distribution to generate the rest of
components

  # Create probability transition matrix


main_line_transition_matrix=np.matrix([[7.17,1.69,0,7.2,1.27,0,3.81,1.28,61.18,11.3
9,1.28,0,3.8],[5.45,0.45,0,7.73,0,0,4.1,0,65.91,12.27,0,0,4.09],[0,3.67,0,0,1.83,0,0,1.
84,33.94,54.13,1.85,2.75,0],[32.01,1.92,0.44,15.29,1.7,0.27,14.61,1.69,0.75,14.53,1
.76,0.24,14.78],[32.43,0.19,0.5,16.33,0.11,0.27,16.22,0.1,0.9,16.23,0.12,0.25,16.38]
,[0.78,0,12.34,49.87,0.51,5.67,1.54,0,8.23,14.65,0,5.66,0.77],[5.28,0.01,29.68,8.7,0,
14.86,2.67,0.13,17.45,3.13,0,14.84,3.25],[3.02,0.34,0,0.69,0.68,0,0.67,0,53.02,22.8
2,0,0,18.79],[19.91,0.45,0,17.3,0,0,16.29,0,0,16.31,0,0,30.77],[23.92,0.38,5.35,12.8
4,0.44,4.93,11.11,0.17,4.15,18.98,0.16,4.18,13.42],[14.26,13,0.1,6.8,5.42,1,6.68,2.7
1,29.32,7.98,3.51,1.1,8.33],[0,0,0,43.62,0,0,0,0,0,56.38,0,0,0],[25.76,0.13,5.6,12.61,
0,2.7,11.13,0.27,13.49,13.96,0,2.8,11.67]])

  # Create column legends of probability transition matrix


matrix_legend=np.array([["TUBE","FTUBE","ELBOW","TEE","FBLIND","INSTRUME
NT","CAP","COUPLING","FLANGE","VALVE","CLOSURE","PCOMPONENT","RED
UCER"]])

   main_line=[]

   branch=[]

   Tee=deque()

   tee_diameter=deque()

   branch_count=0

  # Start generating components sequence

   for i in range (1,1200):

    if (len(main_line)+1) <= (no_of_main_line_components):

      initialize_sequence=np.random.uniform(0,100)

  ############################ Component CAP #####################

      if starting_component == "CAP":
        # Define the next CAP state step
         # No state distribution was found for Component State Cap, which
means it is found once in any pipeline
```

```
        next_component=main_line_transition_matrix[0,:]

        # Sort from smallest to largest

        next_component0=np.sort(next_component)

        # Transpose the row

        next_component1=next_component0.T

        x=[]

        if i == 1:


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","CAP",",",
components_diameter,",",int(np.random.lognormal(0.83986,4.4939)),",",running_dire
ction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"CAP"+","+str(components_diameter)+","+str(int(np.random.lognormal(0
.83986,4.4939)))+","+str(running_direction)+"\n")

         main_line.append("CAP")

         component_number+=1

         # Find the location of the next component in legends matrix

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]
         elif len(main_line) == (no_of_main_line_components):

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","CAP",",",
components_diameter,",",int(np.random.lognormal(0.83986,4.4939)),",",running_dire
ction)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"CAP"+","+str(components_diameter)+","+str(int(np.random.lognormal(0
.83986,4.4939)))+","+str(running_direction)+"\n")
          main_line.append("CAP")
          component_number+=1
          # Find the location of the next component in legends matrix
          if initialize_sequence > np.max(next_component1):
            location= np.max(next_component1)
          else:
            for i in next_component1:
             if initialize_sequence < i:
              x.append(i)
              location=(x[0])
          x.clear()
          a=np.array(next_component)
          y=np.where(a==location)
          starting_component=matrix_legend[y]
         else:
          cap_step+=1
          starting_component= starting_component_state_main_line()
  ############################ Component Instrument ################
      elif starting_component == "INSTRUMENT":
        next_component=main_line_transition_matrix[1,:]

        next_component0=np.sort(next_component)

        next_component1=next_component0.T

        x=[]

        if instrument_step <=(len(main_line)+1):


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","INSTRU
MENT",components_diameter,",",350,",",running_direction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"INSTRUMENT"+","+str(components_diameter)+","+str(350)+","+str(run
ning_direction)+"\n")

          main_line.append("INSTRUMENT")

          component_number+=1

          if initialize_sequence > np.max(next_component1):

            location= np.max(next_component1)

          else:

            for i in next_component1:

             if initialize_sequence < i:

              x.append(i)

              location=(x[0])

          x.clear()

          a=np.array(next_component)

          y=np.where(a==location)

          starting_component=matrix_legend[y]

          # Define the next INSTRUMENT state step

          instrument_next_step=len(main_line)+ int(np.random.gamma(2.0857,
7.042))

          instrument_step= instrument_next_step
```

```
       else:
        starting_component=starting_component_state_main_line()
  ############################ Component Fblind ##################
      elif starting_component == "FBLIND":
       next_component=main_line_transition_matrix[2,:]
       next_component0=np.sort(next_component)
       next_component1=next_component0.T
       x=[]
       # Limit the location of Cap to either the first of the last in the sequence
       if i == 1:

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","FBLIND",
components_diameter,",",int(np.random.uniform(122,777.19)),",",running_direction)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"FBLIND"+","+str(components_diameter)+","+str(int(np.random.uniform(
122,777.19)))+","+str(running_direction)+"\n")#revise
         main_line.append("FBLIND")
         component_number+=1
         if initialize_sequence > np.max(next_component1):
           location= np.max(next_component1)
         else:
           for i in next_component1:
            if initialize_sequence < i:
             x.append(i)
             location=(x[0])
         x.clear()
         a=np.array(next_component)
         y=np.where(a==location)

         starting_component=matrix_legend[y]

       elif len(main_line)== no_of_main_line_components:


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","FBLIND",
components_diameter,",",int(np.random.uniform(122,777.19)),",",running_direction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"FBLIND"+","+str(components_diameter)+","+str(int(np.random.uniform(
122,777.19)))+","+str(running_direction)+"\n")

         main_line.append("FBLIND")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

       else:

         fblind_step+=1

         starting_component=starting_component_state_main_line()

  ############################ Component Tee #####################

      elif starting_component == "TEE":
            next_component=main_line_transition_matrix[3,:]
            next_component0=np.sort(next_component)
            next_component1=next_component0.T
            x=[]
            if running_direction=="x":
              coming_from_x=["y","z"]

running_direction_list.append([component_number,random.choice(coming_from_x)]
)
            elif running_direction=="y":
                coming_from_y=["x","z"]

running_direction_list.append([component_number,random.choice(coming_from_y)]
)
            else:
              coming_from_z=["x","y"]

running_direction_list.append([component_number,random.choice(coming_from_z)]
)

            if tee_step <=(len(main_line)+1):


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","TEE",",",
components_diameter,",",int(np.random.uniform(57,432)),",",running_direction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"TEE"+","+str(components_diameter)+","+str(int(np.random.uniform(57,
432)))+","+str(running_direction)+"\n")
                tee_diameter.appendleft([component_number,components_diameter])
                main_line.append("TEE")
                Tee.append(component_number)
```

```
         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

         # Define the next TEE state step

         tee_next_step=len(main_line)+ int(np.random.gamma(1.4539,
5.3071))## revise the distribution

         tee_step= tee_next_step

       else:

        starting_component=starting_component_state_main_line()

  ############################ Component Elbow #####################

      # we assume elbow as free floating element, therefore no state distribution is
required

      elif starting_component == "ELBOW":

       next_component=main_line_transition_matrix[4,:]

       next_component0=np.sort(next_component)

       next_component1=next_component0.T

       x=[]
       if elbow_step<=(len(main_line)+1):

         if running_direction=="x":

           coming_from_x=["y","z"]

           running_direction= random.choice(coming_from_x)

         elif running_direction=="y":

           coming_from_y=["x","z"]

           running_direction= random.choice(coming_from_y)

         else:

           coming_from_z=["x","y"]

           running_direction= random.choice(coming_from_z)

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","ELBOW"
,",",
components_diameter,",",int(np.random.laplace(339.86,0.00111)),",",running_directi
on)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"ELBOW"+","+
str(components_diameter)+","+str(int(np.random.laplace(339.86,0.00111)))+","+str(r
unning_direction)+"\n")

         main_line.append("ELBOW")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])
         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

         elbow_next_step=len(main_line)+
int(np.random.laplace(0.51675,3.1097))

         elbow_step= elbow_next_step

       else:

        starting_component=starting_component_state_main_line()

  ############################ Component Ftube ####################

      elif starting_component == "FTUBE":

       next_component=main_line_transition_matrix[5,:]

       next_component0=np.sort(next_component)

       next_component1=next_component0.T

       x=[]


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","FTUBE",
",",components_diameter,",",int(np.random.uniform(75.394,76.985)),",",running_dire
ction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"FTUBE"+","+str(components_diameter)+","+str(int(np.random.uniform(7
5.394,76.985)))+","+str(running_direction)+"\n")

       main_line.append("FTUBE")

       component_number+=1

       if initialize_sequence > np.max(next_component1):

         location= np.max(next_component1)

       else:

         for i in next_component1:
```

```
                    if initialize_sequence < i:

                      x.append(i)

                      location=(x[0])

              x.clear()

              a=np.array(next_component)

              y=np.where(a==location)

              starting_component=matrix_legend[y]

    ############################ Component Tube #####################

            # Component tube is assumed as a free floating element, therefore no state
distribution is required

            elif starting_component == "TUBE":

              next_component=main_line_transition_matrix[6,:]

              next_component0=np.sort(next_component)

              next_component1=next_component0.T

              x=[]

              if tube_step <=(len(main_line)+1):


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","TUBE",",
",components_diameter,",",int(np.random.wald(2323.2,489.58)),",",running_direction
)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"TUBE"+","+str(components_diameter)+","+str(int(np.random.wald(2323
.2,489.58)))+","+str(running_direction)+"\n")

                  main_line.append("TUBE")

                  component_number+=1

                  if initialize_sequence > np.max(next_component1):

                      location= np.max(next_component1)

                  else:
```

```
           for i in next_component1:
            if initialize_sequence < i:
             x.append(i)
             location=(x[0])
         x.clear()
         a=np.array(next_component)
         y=np.where(a==location)
         starting_component=matrix_legend[y]
         tube_next_step=len(main_line)+ 1
         tube_step= tube_next_step
       else:
        starting_component=starting_component_state_main_line()
  ############################ Component Pcomponent ################
      elif starting_component == "PCOMPONENT":
       next_component=main_line_transition_matrix[7,:]
       next_component0=np.sort(next_component)
       next_component1=next_component0.T
       x=[]
       if pcomponent_step <=(len(main_line)+1):

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","PCOMPONENT",",",components_diameter,",",int(np.random.wald(726.79,1072.6)),",",running_direction)#int(np.random.exponential(0.0017)))

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main_line)))+","+"PCOMPONENT"+","+str(components_diameter)+","+str(int(np.random.wald(726.79,1072.6)))+","+str(running_direction)+"\n")
         main_line.append("PCOMPONENT")
         component_number+=1
         if initialize_sequence > np.max(next_component1):
           location= np.max(next_component1)
         else:
           for i in next_component1:
             if initialize_sequence < i:
               x.append(i)
               location=(x[0])
         x.clear()
         a=np.array(next_component)
         y=np.where(a==location)
         starting_component=matrix_legend[y]
         # Define the next PCOMPONENT state step
         pcomponent_next_step=len(main_line)+
int(np.random.exponential(0.19245))
         pcomponent_step= pcomponent_next_step
       else:
        starting_component=starting_component_state_main_line()
  ############################ Component Coupling ##################
      elif starting_component == "COUPLING":
       next_component=main_line_transition_matrix[8,:]
       next_component0=np.sort(next_component)
       next_component1=next_component0.T
       x=[]

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","COUPLI
NG",",",components_diameter,",",int(np.random.beta(0.02229,0.42713)),",",running_
direction)
```

```
file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"COUPLING"+","+str(components_diameter)+","+str(int(np.random.beta(
0.02229,0.42713)))+","+str(running_direction)+"\n")
       main_line.append("COUPLING")
       component_number+=1
       if initialize_sequence > np.max(next_component1):
         location= np.max(next_component1)
       else:
         for i in next_component1:
          if initialize_sequence < i:
           x.append(i)
           location=(x[0])
       x.clear()
       a=np.array(next_component)
       y=np.where(a==location)
       starting_component=matrix_legend[y]
  ############################ Component Flange ###################
      elif starting_component == "FLANGE":
       next_component=main_line_transition_matrix[9,:]
       next_component0=np.sort(next_component)
       next_component1=next_component0.T
       x=[]
       if flange_step <=(len(main_line)+1):

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","FLANGE
",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"FLANGE"+","+str(components_diameter)+","+str(350)+","+str(running_
direction)+"\n")

         main_line.append("FLANGE")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

             if initialize_sequence < i:

               x.append(i)

               location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

         # Define the next FLANGE state step

         flange_next_step=len(main_line)+ int(np.random.pareto(0.82625,1))

         flange_step= flange_next_step

       else:

        starting_component=starting_component_state_main_line()

  ############################ Component Valve ###################

      elif starting_component == "VALVE":

       next_component=main_line_transition_matrix[10,:]

       next_component0=np.sort(next_component)

       next_component1=next_component0.T

       x=[]
       if valve_step <=(len(main_line)+1):

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","VALVE",
",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"VALVE"+","+str(components_diameter)+","+str(350)+","+str(running_dir
ection)+"\n")

         main_line.append("VALVE")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

         valve_next_step=len(main_line)+
int(np.random.lognormal(0.7945,1.8998))

         valve_step= valve_next_step

       else:

        starting_component=starting_component_state_main_line()

  ############################ Component Closure ##################

      elif starting_component == "CLOSURE":

       next_component=main_line_transition_matrix[11,:]
       next_component0=np.sort(next_component)

       next_component1=next_component0.T

       x=[]

       if i == 1:

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","CLOSURE",",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main_line)))+","+"CLOSURE"+","+str(components_diameter)+","+str(350)+","+str(running_direction)+"\n")

         main_line.append("CLOSURE")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

            if initialize_sequence < i:

             x.append(i)

             location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

       elif len(main_line) == no_of_main_line_components:

print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","CLOSURE",",",components_diameter,",",350,",",running_direction)
```

```
file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"CLOSURE"+","+str(components_diameter)+","+str(350)+","+str(running
_direction)+"\n")
         main_line.append("CLOSURE")
         component_number+=1

         if initialize_sequence > np.max(next_component1):
           location= np.max(next_component1)
         else:
           for i in next_component1:
            if initialize_sequence < i:
             x.append(i)
             location=(x[0])
         x.clear()
         a=np.array(next_component)
         y=np.where(a==location)
         starting_component=matrix_legend[y]

       else:
         clousre_step+=1
         starting_component=starting_component_state_main_line()
  ############################ Component Reducer #################
      elif starting_component == "REDUCER":
       next_component=main_line_transition_matrix[12,:]
       next_component0=np.sort(next_component)
       next_component1=next_component0.T
       x=[]
```

```
        reducer_diameter=int(burr.rvs(c, d, loc=0, scale=124.3, size=1))

        components_diameter= reducer_diameter


        if reducer_step <=(len(main_line)+1):


print(line_no,",","Main_Line",",",component_number,",",(len(main_line)),",","REDUC
ER",",",components_diameter,",",int(np.random.gamma(3.4645,51.749)),",",running_
direction)


file.write(str(line_no)+","+"Main_Line"+","+str(component_number)+","+str((len(main
_line)))+","+"REDUCER"+","+str(components_diameter)+","+str(int(np.random.gam
ma(3.4645,51.749)))+","+str(running_direction)+"\n")

         main_line.append("REDUCER")

         component_number+=1

         if initialize_sequence > np.max(next_component1):

           location= np.max(next_component1)

         else:

           for i in next_component1:

             if initialize_sequence < i:

              x.append(i)

              location=(x[0])

         x.clear()

         a=np.array(next_component)

         y=np.where(a==location)

         starting_component=matrix_legend[y]

         # Define the next REDUCER state step

         reducer_next_step=len(main_line)+
int(np.random.lognormal(0.80974,2.0731))

         reducer_step= reducer_next_step

        else:
```

```
        starting_component=starting_component_state_main_line()

      else:

       initialize_sequence=np.random.uniform(0,100)

    else:

     break

  ################################ Branches Generator ################

   try:


   while len(Tee)>0:


     for i in range (len(Tee)):

       branch_count+=1

       cap_step=1
       instrument_step=1
       tube_step=1
       valve_step=1
       fblind_step=1
       ftube_step=1
       flange_step=1
       clousre_step=1
       pcomponent_step=1
       tee_step=1
       reducer_step=1
       coupling_step=1
       elbow_step=1

       no_of_branch_components=int(math.ceil(np.random.lognormal(0.88575,
1.2818)))

       if no_of_branch_components <1:

        no_of_branch_components=1

      # Initialize the starting component for each branch

       initialize_starting_component=np.random.uniform(0,100)

      # The following statement constrains the type of component to be processed
in case of the number of components=1
```

```
if no_of_branch_components <= 1:
 if initialize_starting_component <=0.59:
  starting_component="CAP"
 elif initialize_starting_component <=1.2:
  starting_component="FTUBE"
 elif initialize_starting_component <=2.9:
  starting_component="REDUCER"
 elif initialize_starting_component <=5.4:
  starting_component="TUBE"
 elif initialize_starting_component <=15.9:
  starting_component="INSTRUMENT"
 elif initialize_starting_component <=29.2:
  starting_component="FLANGE"
 else:
  starting_component="CLOSURE"
else:
    if initialize_starting_component <= 0.2:
        starting_component="COUPLING"
    elif initialize_starting_component <= 0.35:
        starting_component="INSTRUMENT"
    elif initialize_starting_component <= 1.01:
        starting_component="VALVE"
    elif initialize_starting_component <= 2.13:
        starting_component="TEE"
    elif initialize_starting_component <= 3.79:
        starting_component="REDUCER"
```

```
          elif initialize_starting_component <= 9.44:

              starting_component="ELBOW"

          elif initialize_starting_component <= 25.62:

              starting_component="FLANGE"

          elif initialize_starting_component <= 59.84:

              starting_component="FTUBE"

          else:

              starting_component="TUBE"


      # Use Markov-chain transition matrix and state distribution to generate the
rest of components

      # Create probability transition matrix (modified to branches)


main_line_transition_matrix=np.matrix([[0,37.5,0,0,18.81,0,0,25,0,0,18.8,0,0],[4.8,0,
0,9.5,0,0,2.4,0,77.4,2.41,0,1.2,2.411],[0,38.5,0,0,19.2,0,0,19.21,3.8,0,19.21,0,0],[30.
7,1.9,0.4,15.8,0.94,0.21,15.6,0.93,1.1,15.4,1.,0.2,15.7],[31.9,0.6,0.5,16.0,0.31,0.2,1
5.9,0.3,1.7,15.91,0.311,0.301,16.1],[0.2,0,6,0.41,0,3.2,6.8,0,3.5,76.2,0.4,3.1,0.1],[5.
9,0.0,27.4,10.4,0,13.8,3,0,18.3,3.3,0,13.7,4.0],[6.3,0,0,3.2,3.21,1.6,3.211,0,58.7,20.
6,0,0,3.211],[20.3,13.6,0,10.2,6.8,3.4,10.21,6.81,0,10.211,6.811,0,11.9],[17.1,2.8,3.
8,9.4,3.4,6.7,8.6,1.4,5.4,26.3,1.41,4,9.8],[3.4,5.9,0.2,1.7,17.9,8.5,1.71,3.0,15.4,2,35.
8,2.4,2.1],[0,33.3,0,0,16.7,0,0,16.71,0,16.711,16.7111,0,0],[22.7,0,3.3,12.3,0,1.7,11.
3,0.8,15.2,17.3,0,2.8,12.7]])

      # Create column legends of probability transition matrix


matrix_legend=np.array([["TUBE","FTUBE","ELBOW","TEE","FBLIND","INSTRUME
NT","CAP","COUPLING","FLANGE","VALVE","CLOSURE","PCOMPONENT","RED
UCER"]])

      # Start generating components sequence

       for i in range (1,2000):


        if i ==1:

         previously_connected=Tee.popleft()

        else:
         previously_connected= component_number-1


        for row in tee_diameter:

         if row[0]== previously_connected:

          components_diameter=row[1]

        for row in running_direction_list:

          if row[0]==previously_connected:

           running_direction=row[1]


        if (len(branch)) <= (no_of_branch_components-1):


          initialize_sequence=np.random.uniform(0,100)


      ############################ Component CAP #################

          if starting_component == "CAP":


            # define the next CAP state step

            # No state distribution was found for Component State Cap, which
means it is found once in any pipeline

             next_component=main_line_transition_matrix[0,:] #

            # Sort the above array from smallest to largest

             next_component0=np.sort(next_component)

            # Transpose the array

             next_component1=next_component0.T

             x=[]

             # Limit the location of Cap to either the first of the last in the
sequence

             if i == 1:
```

```
print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","CAP",",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"CAP"+","+str(components_diameter)+","+str(350)
+","+str(running_direction)+"\n")
              branch.append("CAP")

              component_number+=1

              # Find the location of the next component in the sorted array

              if initialize_sequence > np.max(next_component1):

                location= np.max(next_component1)

              else:

                for i in next_component1:

                 if initialize_sequence < i:

                  x.append(i)

                  location=(x[0])

              x.clear()

              a=np.array(next_component)

              y=np.where(a==location)

              starting_component=matrix_legend[y]

             elif len(branch) == (no_of_branch_components):

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","CAP",",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"CAP"+","+str(components_diameter)+","+str(350)
+","+str(running_direction)+"\n")
              branch.append("CAP")
              component_number+=1

              # Find the location of the next component in legends matrix

              if initialize_sequence > np.max(next_component1):

                location= np.max(next_component1)


              else:

                for i in next_component1:

                 if initialize_sequence < i:

                  x.append(i)

                  location=(x[0])

              x.clear()

              a=np.array(next_component)

              y=np.where(a==location)

              starting_component=matrix_legend[y]

             else:

              cap_step+=1

              starting_component= starting_component_state_branch()

      ############################ Component Instrument ############

          elif starting_component == "INSTRUMENT":

           next_component=main_line_transition_matrix[1,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]

           if instrument_step <=(len(branch)+1):

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_connected,",","INSTRUMENT",",",components_diameter,",",350,",",running_direction)
```

```
file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"INSTRUMENT"+","+str(components_diameter)+",
"+str(350)+","+str(running_direction)+"\n")
             branch.append("INSTRUMENT")
             component_number+=1
             if initialize_sequence > np.max(next_component1):
               location= np.max(next_component1)
             else:
               for i in next_component1:
                if initialize_sequence < i:
                 x.append(i)
                 location=(x[0])
             x.clear()
             a=np.array(next_component)
             y=np.where(a==location)
             starting_component=matrix_legend[y]
             # Define the next INSTRUMENT state step
             instrument_next_step=len(branch)+ int(np.random.uniform(-2.381,
47.492))
             instrument_step= instrument_next_step
           else:
            starting_component=starting_component_state_branch()
      ############################ Component Fblind ################
          elif starting_component == "FBLIND":
           next_component=main_line_transition_matrix[2,:]
           next_component0=np.sort(next_component)
           next_component1=next_component0.T
```

```
        x=[]

        # Limit the location of Cap to either the first of the last in the sequence

        if i == 1:

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","FBLIND",",",components_diameter,",",int(np.random.uniform(19,50)),",
",running_direction)

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"FBLIND"+","+str(components_diameter)+","+str(in
t(np.random.uniform(19,50)))+","+str(running_direction)+"\n")

          branch.append("FBLIND")

          component_number+=1

          if initialize_sequence > np.max(next_component1):

            location= np.max(next_component1)

          else:

            for i in next_component1:

             if initialize_sequence < i:

              x.append(i)

              location=(x[0])

          x.clear()

          a=np.array(next_component)

          y=np.where(a==location)

          starting_component=matrix_legend[y]

        elif len(branch)== no_of_branch_components:

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","FBLIND",",",components_diameter,",",int(np.random.uniform(19,50)),",
",running_direction)
```

```
file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+","+str(previously_connected)+","+"FBLIND"+","+str(components_diameter)+","+str(int(np.random.uniform(19,50)))+","+str(running_direction)+"\n")

             branch.append("FBLIND")

             component_number+=1

             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:

                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()

             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

           else:

             fblind_step+=1

             starting_component=starting_component_state_branch()

      ############################ Component Tee #################

          elif starting_component == "TEE":

           next_component=main_line_transition_matrix[3,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]

           if running_direction=="x":
            coming_from_x=["y","z"]

running_direction_list.append([component_number,random.choice(coming_from_x)]
)
           elif running_direction=="y":
             coming_from_y=["x","z"]

running_direction_list.append([component_number,random.choice(coming_from_y)]
)
           else:
            coming_from_z=["x","y"]

running_direction_list.append([component_number,random.choice(coming_from_z)]
)
           if tee_step <=(len(branch)+1):

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","TEE",",",components_diameter,",",int(np.random.uniform(57,432)),",",r
unning_direction)#,int(np.random.gamma(1.3782,145.74)))

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"TEE"+","+str(components_diameter)+","+str(int(n
p.random.uniform(57,432)))+","+str(running_direction)+"\n")#
             branch.append("TEE")

tee_diameter.appendleft([component_number,components_diameter])
             if len(Tee)==0:
              Tee.append(component_number)
             else:
              Tee.appendleft(component_number)
             component_number+=1
             if initialize_sequence > np.max(next_component1):
```

```
         location= np.max(next_component1)
       else:
         for i in next_component1:
          if initialize_sequence < i:
           x.append(i)
           location=(x[0])
       x.clear()
       a=np.array(next_component)
       y=np.where(a==location)
       starting_component=matrix_legend[y]
       # Define the next TEE state step
       tee_next_step=len(branch)+ int(np.random.gamma(0.95639, 5.3605))
       tee_step= tee_next_step
     else:
      starting_component=starting_component_state_branch()
############################ Component Elbow ################
    elif starting_component == "ELBOW":
     next_component=main_line_transition_matrix[4,:]
     next_component0=np.sort(next_component)
     next_component1=next_component0.T
     x=[]
     if elbow_step<=(len(branch)+1):

       if running_direction=="x":
         coming_from_x=["y","z"]
         running_direction= random.choice(coming_from_x)
```

```
             elif running_direction=="y":

               coming_from_y=["x","z"]

               running_direction= random.choice(coming_from_y)

             else:

               coming_from_z=["x","y"]

               running_direction= random.choice(coming_from_z)


print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","ELBOW",",",components_diameter,",",int(np.random.laplace(339.86,0.
00111)),",",running_direction)#int(np.random.gamma(1.0652,257.04)))


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"ELBOW"+","+str(components_diameter)+","+str(in
t(np.random.laplace(339.86,0.00111)))+","+str(running_direction)+"\n")

             branch.append("ELBOW")

             component_number+=1

             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:

                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()

             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

             elbow_next_step=len(branch)+
int(np.random.laplace(0.51675,3.1097))
```

```
             elbow_step= elbow_next_step

           else:

             starting_component=starting_component_state_branch()

      ############################ Component Ftube #################

          elif starting_component == "FTUBE":

           next_component=main_line_transition_matrix[5,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]


print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","FTUBE",",",components_diameter,",",int(np.random.uniform(56,99.82))
,",",running_direction)


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"FTUBE"+","+str(components_diameter)+","+str(int
(np.random.uniform(56,99.82)))+","+str(running_direction)+"\n")

           branch.append("FTUBE")

           component_number+=1

           if initialize_sequence > np.max(next_component1):

             location= np.max(next_component1)

           else:

             for i in next_component1:

               if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

           x.clear()

           a=np.array(next_component)
           y=np.where(a==location)

           starting_component=matrix_legend[y]

      ############################ Component Tube #################

          # Component tube is assumed as a free floating element, therefore no
state distribution is required

          elif starting_component == "TUBE":

           next_component=main_line_transition_matrix[6,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]

           if tube_step <=(len(branch)+1):


print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","TUBE",",",components_diameter,",",int(np.random.wald(2323.2,489.58
)),",",running_direction)#int(math.ceil(np.random.gamma(0.21074,11024))+50))#
revise no 50


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"TUBE"+","+str(components_diameter)+","+str(int(
np.random.wald(2323.2,489.58)))+","+str(running_direction)+"\n")

             branch.append("TUBE")

             component_number+=1

             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:

                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()
             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

             tube_next_step=len(branch)+ 1

             tube_step= tube_next_step

           else:

            starting_component=starting_component_state_branch()

      ############################ Component Pcomponent ############

          elif starting_component == "PCOMPONENT":

           next_component=main_line_transition_matrix[7,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]

           if pcomponent_step <=(len(branch)+1):


print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","PCOMPONENT",",",components_diameter,",",int(np.random.wald(726.
79,1072.6)),",",running_direction)#int(math.ceil(np.random.lognormal(0.86246,6.266
))+50)) #revise


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"PCOMPONENT"+","+str(components_diameter)+
","+str(int(np.random.wald(726.79,1072.6)))+","+str(running_direction)+"\n")

             branch.append("PCOMPONENT")

             component_number+=1

             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:
                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()

             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

             # Define the next PCOMPONENT state step

             pcomponent_next_step=len(branch)+
int(np.random.exponential(0.31376))

             pcomponent_step= pcomponent_next_step

           else:

            starting_component=starting_component_state_branch()

      ############################ Component Coupling ##############

          elif starting_component == "COUPLING":

           next_component=main_line_transition_matrix[8,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]


print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","COUPLING",",",components_diameter,",",int(np.random.beta(0.02229,
0.42713)),",",running_direction)


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"COUPLING"+","+str(components_diameter)+","+s
tr(int(np.random.beta(0.02229,0.42713)))+","+str(running_direction)+"\n")

           branch.append("COUPLING")

           component_number+=1
           if initialize_sequence > np.max(next_component1):
             location= np.max(next_component1)
           else:
             for i in next_component1:
              if initialize_sequence < i:
               x.append(i)
               location=(x[0])
           x.clear()
           a=np.array(next_component)
           y=np.where(a==location)
           starting_component=matrix_legend[y]
      ############################ Component Flange ###############
          elif starting_component == "FLANGE":
           next_component=main_line_transition_matrix[9,:]
           next_component0=np.sort(next_component)
           next_component1=next_component0.T
           x=[]
           if flange_step <=(len(branch)+1):

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","FLANGE",",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"FLANGE"+","+str(components_diameter)+","+str(
350)+","+str(running_direction)+"\n")
             branch.append("FLANGE")
             if initialize_sequence > np.max(next_component1):
               location= np.max(next_component1)
               component_number+=1
             else:
               for i in next_component1:
                if initialize_sequence < i:
                 x.append(i)
                 location=(x[0])
             x.clear()
             a=np.array(next_component)
             y=np.where(a==location)
             starting_component=matrix_legend[y]
             # Define the next FLANGE state step
             flange_next_step=len(branch)+ int(np.random.pareto(0.94902,1))
             flange_step= flange_next_step
           else:
            starting_component=starting_component_state_branch()
      ############################ Component Valve ################
          elif starting_component == "VALVE":
           next_component=main_line_transition_matrix[10,:]
           next_component0=np.sort(next_component)
           next_component1=next_component0.T
           x=[]
           if valve_step <=(len(branch)+1):

print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","VALVE",",",components_diameter,",",350,",",running_direction)

file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"VALVE"+","+str(components_diameter)+","+str(35
0)+","+str(running_direction)+"\n")

             branch.append("VALVE")

             component_number+=1

             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:

                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()

             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

             valve_next_step=len(branch)+ int(np.random.gamma(1.437,5.584))

             valve_step= valve_next_step

           else:

            starting_component=starting_component_state_branch()

      ############################ Component Closure ###############

          elif starting_component == "CLOSURE":

           next_component=main_line_transition_matrix[11,:]

           next_component0=np.sort(next_component)

           next_component1=next_component0.T

           x=[]

           if i == 1:
```

```
print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","CLOSURE",",",components_diameter,",",350,",",running_direction)


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"CLOSURE"+","+str(components_diameter)+","+st
r(350)+","+str(running_direction)+"\n")

             branch.append("CLOSURE")

             component_number+=1


             if initialize_sequence > np.max(next_component1):

               location= np.max(next_component1)

             else:

               for i in next_component1:

                if initialize_sequence < i:

                 x.append(i)

                 location=(x[0])

             x.clear()

             a=np.array(next_component)

             y=np.where(a==location)

             starting_component=matrix_legend[y]

           elif len(branch) == no_of_branch_components:



print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","CLOSURE",",",components_diameter,",",350,",",running_direction)


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"CLOSURE"+","+str(components_diameter)+","+st
r(350)+","+str(running_direction)+"\n")

             branch.append("CLOSURE")

             component_number+=1
```

```
       if initialize_sequence > np.max(next_component1):
         location= np.max(next_component1)
       else:
         for i in next_component1:
          if initialize_sequence < i:
           x.append(i)
           location=(x[0])
       x.clear()
       a=np.array(next_component)
       y=np.where(a==location)
       starting_component=matrix_legend[y]

     else:
       clousre_step+=1
       starting_component=starting_component_state_branch()
############################ Component Closure ################
    elif starting_component == "REDUCER":
     next_component=main_line_transition_matrix[12,:]
     next_component0=np.sort(next_component)
     next_component1=next_component0.T
     x=[]

     reducer_diameter=int(burr.rvs(c, d, loc=0, scale=124.3, size=1))
     components_diameter= reducer_diameter
     if reducer_step <=(len(branch)+1):
```

```
print(line_no,",","Branch"+str(branch_count),",",component_number,",",previously_c
onnected,",","REDUCER",",",components_diameter,",",int(np.random.gamma(3.464
5,51.749)),",",running_direction)


file.write(str(line_no)+","+"Branch"+str(branch_count)+","+str(component_number)+"
,"+str(previously_connected)+","+"REDUCER"+","+str(components_diameter)+","+st
r(int(np.random.gamma(3.4645,51.749)))+","+str(running_direction)+"\n")

            branch.append("REDUCER")

            component_number+=1


            if initialize_sequence > np.max(next_component1):

              location= np.max(next_component1)

            else:

              for i in next_component1:

               if initialize_sequence < i:

                x.append(i)

                location=(x[0])

            x.clear()

            a=np.array(next_component)

            y=np.where(a==location)

            starting_component=matrix_legend[y]

            reducer_next_step=len(branch)+
int(np.random.lognormal(0.86124,2.1598))

            reducer_step= reducer_next_step

           else:

            starting_component=starting_component_state_branch()


          else:

           initialize_sequence=np.random.uniform(0,100)
```

```
                else:

                  break

              cap_step=1
              instrument_step=1
              tube_step=1
              valve_step=1
              fblind_step=1
              ftube_step=1
              flange_step=1
              clousre_step=1
              pcomponent_step=1
              tee_step=1
              reducer_step=1
              coupling_step=1
              elbow_step=1


              branch.clear()

      except IndexError:

        break

print('Please inter the number of industrial pipelines!')
userinput= [input()]
PipelineGenerator(int(userinput[0]))
```