PARALLELIZATION OF A NOVEL FREQUENT ITEMSET HIDING ALGORITHM ON A CPU-GPU PLATFORM

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by

Samuel Bacha HEYE

May 2014
IZMIR

We approve the thesis of **Samuel Bacha HEYE**

Examining Committee Members:

Asst. Prof. Dr. Tolga AYAV
Department of Computer Engineering, Izmir Institute of Technology

Asst. Prof. Dr. Belgin ERGENÇ
Department of Computer Engineering, Izmir Institute of Technology

Asst. Prof. Dr. Şevket GÜMÜŞTEKIN
Department of Electrical and Electronics Engineering, Izmir Institute of Technology

29 May 2014

Asst. Prof. Dr. Tolga AYAV
Supervisor, Department of Computer Engineering, Izmir Institute of Technology

Prof. Dr. Sıtkı AYTAÇ
Head of the Department of Computer Engineering

Prof. Dr. R. Tuğrul SENGER
Dean of the Graduate School of Engineering and Sciences

ACKNOWLEDGMENTS

I would like to thank my advisor Assit. Prof. Dr. Tolga Ayav for coming up with the thesis idea and the algorithm for hiding of sensitive frequent itemsets as well as for his guidance and support to make me implement the sensitive frequent itemset hiding algorithm on a CPU-GPU platform. I would also like to thank my family for their continuous support throughout my life. My special thanks go to Juhar Abdella and Tesfa Guchale for helping me to finalize the binding of my Thesis.

ABSTRACT

PARALLELIZATION OF A NOVEL FREQUENT ITEMSET HIDING ALGORITHM ON A CPU-GPU PLATFORM

Data mining is used to extract useful information from large data. But the organizations which mine the data might not be the owner of the data. So, before the owners can make their data accessible for data mining they want to make sure that no sensitive information can be mined from the released data whose discovery by others might harm them. Itemset hiding is one mechanism to prevent the disclosure of sensitive itemsets. In this thesis, a new integer programing based itemset hiding algorithm was developed and a mechanism to speed up the computation time of its implementation was proposed by using parallel computation on Graphical Processing Units (GPUs).

ÖZET

YENI BIR SIK KÜMELERI GIZLEME ALGORITMASININ CPU-GPU PLAFORMU ÜZERINDE PARALLELLEŞTIRILMESI

Veri madenciliği büyük veriden yararlı bilgileri ayıklamak için kullanılır. Ancak veriyi ayıklayan örgütler verinin sahibi olmayabilirler. Bu yüzden,veriyi Veri madenciliği için erişilebilir yapmadan önce veri sahipleri serbest bırakılan veriden hassas bilgilerin ayıklamadığından emin olmak istiyorlar. Itemset gizleme hassas itemset'lerinin açıklanmasını önlenmek için bir mekanizmadır. Bu tezde, yeni bir tamsayı programlama tabanlı Itemset gizleme algoritması geliştirilmiştir ve hesaplama zamanını hızlandırmak için Grafik İşleme Birimi (GPU) üzerinde paralel hesaplama kullanarak bir mekanizma önerilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Data mining is a technique used for extracting information from large data [1]. It is used in a wide range of areas. There are different data mining algorithms and one of them is association rule mining. Association rule mining is concerned with extraction of association rules, which are implications of the form $X \rightarrow Y$ [3]. Association rules are used for various purposes with market basket analysis being one of them. The Apriori algorithm can be used for mining of association rules.

Privacy preserving association rule mining (association rule hiding) is one area of privacy preserving data mining which aims at preventing sensitive association rules from being disclosed as a result of data mining carried out by third parties without authorization or approval[8][10]. There are heuristic, border-based and exact association rule hiding algorithms [9]. Since by hiding the frequent itemsets of a dataset which result in sensitive association rules it is also possible to hide the sensitive association rules, in this paper a new exact algorithm for hiding of such sensitive frequent itemsets is proposed.

The proposedd IP based itemset hiding algorithm aims at minimizing the number of items removed from the database, the number of non-sensitive itemsets removed from transactions and the number of non-sensitive itemsets removed from the database while meeting a number of constraints. The results of the implementation of the proposed IP based sanitization algorithm shows that the algorithm successfully hides sensitive itemsets from the input datasets with the least possible impact on the non-sensitive itemsets.

Our dataset-sanitization algorithm involves solving of an integer programming problem that specifies the goals and the constraints of the sanitization process. There are exact and heuristic algorithms for solving of such integer programming problems[12]. But, since our itemset hiding approach is exact, i.e. since we want to obtain optimal and not approximate solutions, one of the most commonly used exact algorithms called the branch and bound algorithm was used. The bound phase of the branch and bound algorithm requires linear programming(LP) problems to be solved. And among the algorithms for solving linear programming problems the revised simplex method was chosen[16].

Since integer programming problems are NP hard [13] and it may take quite a long time to solve them, the use of GPU to offload some of the computation from the CPU was proposed. The main way in which GPUs differ from CPUs is in the large number of cores they posses which gives them the ability to execute several computations in parallel[18]. Although GPUs were originally created for rendering graphics, they are now also widely used for General Purpose computation. Among the high level languages one can use to write programs for GPUs are Cuda and OpenCL. In this thesis, OpenCL was used because while programs written using OpenCL have the advantage of being portable, their performance was also found not to be affected due to their portability[26]. One research shows that instead of doing all the computations on CPU or GPU alone, if the computations are done on a CPU-GPU platform by switching between the two platforms appropriately, a better performance gain is obtained[22].Accordingly, our proposed CPU-GPU architecture also shows speedup over the sanitization computations that are done only on CPU.

But, the use of GPUs was only feasible for small sanitization problems. This is because for large sanitization problems, the sanitization IP problem will have lots of constraints and variables. Attempting to solve such a large problem on GPU will make the GPU run out of memory since memory efficient representations like sparse matrix representations can’t be implemented on GPUs as GPUs (OpenCL) donot allow dynamic memory allocations [38]. We will suggest at the end how this limitation can be overcome in the future.

The rest of this paper is organized as follows: Section 2 discusses approaches for hiding frequent itemsets(Association Rules) when datasets are mined using a branch of data mining called association rule mining. Approaches for solving integer programming (IP) and linear programming (LP) problems are also discussed in this section since exact itemset hiding algorithms rely on solving such problems. At the end of section 2 an introduction to GPUs is given and how GPUs have been used until now to parallelize optimization problems like Integer Programming. In Section 3, we introduce our proposed IP based sanitization algorithm and how it is implemented on a CPU-GPU platform. In Section 4, we show the results of our implementation and in Section 5, we summarize the achievements of our thesis and how remaining works can be approached in the future.

CHAPTER 2

LITERATURE REVIEW

2.1. Data Mining

Due to advances in technology, large datasets with gigabytes or terabytes of data are created or collected by the computer systems of different companies and institutions [1]. These datasets can range from daily credit card transactions or CCTV recordings to the terabyte of data that is generated by companies like NASA with its Earth Observation satellites [2]. There usually exist potentially valuable information or embedded knowledge within these huge data that can potentially make companies make profits and scientists make discoveries. But extraction of this information is not trivial because it usually gets obscured within the large data and if we can't be able to extract all or most of the embedded knowledge with in the huge data and if we merely store the data, we will miss out on making the best use of the collected data.

Data mining is thus developed to solve this problem and it is a method of analyzing a large data to identify potentially useful and previously unknown relationships and patterns so as to create a useful summary of the data [1]. Without the use of data mining it is difficult to extract those useful information and patterns because they will be hidden with the large amount of data.

Data mining is used in a wide variety of areas such as targeted marketing, weather forecasting, financial forecasting and medical diagnosis [2]. For example, a supermarket can collect data on its customer's transactions and after determining its high value customers by data mining (which for instance may turn out to be people within a certain age group or gender), then it can target its marketing to these customers. Similarly, data mining can be used in medical diagnosis to predict the probability that a cancer patient may respond to chemotherapy so as to avoid unnecessary costs if it is applied to patients who don't respond to it.

There are different types of data mining algorithms or procedures which take data as input and produce output in the form of patterns [2]. The algorithms differ from one another based on the following components i) the purpose of the data mining (e.g. classification, clustering, association rule learning etc.) ii) the type of pattern or model that they try to fit to the data iii) the objective function that they use to determine the quality of fit of the suggested models or patterns iv) the mechanism that they use to optimize (maximize/minimize) the objective function v) the data structures that they use to store and retrieve the data

2.2. Association Rule Mining

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. An association rule is an implication of the form $X \rightarrow Y$ where $X \; U \; Y \subseteq I$, $X \cap Y = \emptyset$ and where X and Y are a set of items (Itemsets) from I that are called antecedent and consequent respectively [3]. The support of an association rule $X \rightarrow Y$ is expressed as $\sup(X \rightarrow Y)$, which is equivalent to $\sup(X \; U \; Y)$, and it denotes the number of transactions in the database that support the rule. A database D is said to support an association rule $X \rightarrow Y$ if there exists a transaction T in the database such that$X \rightarrow Y \subseteq T$.

It is important to note that association rules do not always hold [3]. For example, the association rule $Bread \rightarrow Butter$ can't hold 100% of the time because not everyone who buys bread also buys butter. So, a quantity called *confidence* is used as a measure of the probability of a rule holding at a given time or as the measure of the degree of confidence we have on the correctness of the rule at a given time. For the rule $\rightarrow Y$, its confidence is $\sup(X \; U \; Y)/\sup(X)$ the confidence is the proportion of items that support the rule from the set of itemsets that support its antecedent.

2.2.1. Application Areas of Association Rules

Association rules are mainly used for market basket analysis [3]. Market basket analysis is the analysis of the market basket of customers to find associated products. For example, after analyzing customer's transactions, a store may find out 80% of customers who buy bread also buy butter.

Association rules can help a store sale many items by offering discount to items bought in pair. If the two items forming the pair are closely associated to each other, customers will likely buy both of them when they are offered together, thus increasing the number of items sold by the store. Another way a store can use association rules is that it can place the two associated items side by side so that customers wishing to buy one of the items will be reminded to also buy the other item.

2.2.2. The Apriori Algorithm for Association Rule Mining

There are different algorithms for mining association rules, i.e. rules whose support and cofidence is above a user specified minimum threshol[3]. The most common are Apriori[4], FP-Growth[5] and Eclat[6]. Here we will only discuss the Apriori Algorithm as the focus of our thesis is on hiding frequent itemsets, which can result in association rules we don't want to be known, regardless of with what method they were obtained and since Apriori is a well-known algorithm which can do the work of obtaining frequent itemsets.

The Apriori algorithm finds all frequent itemsets, which are itemsets whose support exceeds a minimum support threshold [3]. Then, it uses the frequent itemsets to determine all rules whose confidence exceeds a minimum confidence threshold.

The Frequent Itemset Mining Phase

The Apriori algorithm starts from an empty set and identifies a set of candidate itemsets of size 1 which can potentially be frequent [3]. Then it scans the database to determine the support of these candidate itemsets. The candidates whose support exceeds the minimum support threshold become frequent. After obtaining frequent itemsets of size 1, it creates candidate itemsets of size 2 from the frequent itemsets of size 1 by adding one more item to them. Again the support of the candidate itemsets is determined by scanning the database, and those candidates whose support exceeds the minimum support threshold will be added to the list of frequent itemsets. The process is repeated using new frequent itemsets until no more frequent itemsets can be generated.

In general, the Apriori algorithm proceeds from a candidate itemsets of size C_k to find frequent itemsets of size k whose support in the database exceeds the minimum support threshold. Then, candidate itemsets of size k+1 are generated from the frequent itemsets of size k by adding one more item to them. Having candidate itemset of size k+1, the above procedures are repeated until no more candidate itemsets can be generated by adding one more item to the previous frequent itemsets.

The Apriori algorithm terminates at the first encounter of no frequent itemsets of size k because if an itemset of size k is infrequent, no superset of it can become frequent [3]. In other words, if we have two itemsets X and Y, where $X \subseteq Y$, it is obvious that $\sup(X) \geq \sup(Y)$ always holds and this implies if $\sup(X)$ is below the minimum support threshold, $\sup(Y)$ will also be below the minimum support threshold.

```
$C_1 = \{i\}\ such\ that\ i \in I$
k = 1
while $C_k \neq \emptyset$
{
        ➢ Find support of $C_k$ in D
        ➢ Find frequent itemsets of size k $F_k$ s.t.
            $F_k = \{ S \in C_k\ s.t. \sup(S) > minSup\}$
        ➢ Generate candidate itemsets of size k+1 $C_{k+1}$ from $F_k$
        ➢ k = k + 1
}
The frequent itemsets of the database are obtained as $F_1 \cup F_2 \cup \ldots . F_{k-1}$
```

Figure 2.1 The Apriori Algorithm (frequent itemset generation phase)

If the set of items in our dataset are $I = \{1,2,3\}$, then the set of itmsets that can be formed from them form the following lattice [3].

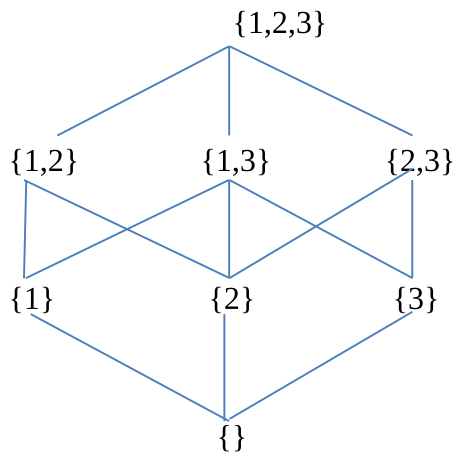


Figure 2.2. Itemset Lattice

And for the above lattice, there is a corresponding tree structure which encodes every itemset in the lattice only once as shown in the figure below [3]. Every node in the tree represents an itemset which contains items in the path from the root of the tree to that node.

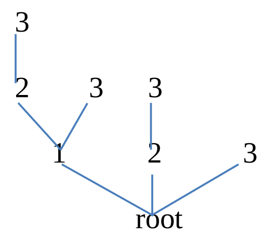


Figure 2.3. Tree structure of the itemsets

The tree structure makes generating candidate itemsets from frequent itemsets efficient because candidate itemsets of size k+1 can be generated from every 2 itemsets of size k having the same parent node in the tree. For instance, if 2 and 3 are frequent itemsets of size 1, a candidate itemset of size 2 can be generated from them by adding 3 as a leaf of node 2.

The above tree structure can also be used to find the support of candidate itemsets as follows. To determine the support of candidate itemsets of size k C_k one pass through the dataset is needed. And at the start of each pass, the support of the candidate itemsets in C_k will be set to zero. Then, for each transaction T in the database, the support of those candidate itemsets which are supported by transaction T will be incremented. This is done by following paths in the tree which contain the items in the transaction T and if we reach nodes at level k of the tree, the support of itemsets in C_k corresponding to those nodes will be incremented.

Example: The steps of the Apriori algorithm can be best understood using the following example. Assume we are given the following database with 5 transactions and 4 items and we want to mine frequent itemsets using a minimum support threshold of 2.

Table 2.1. Example database(dataet)

	i_i	i_2	i_3	i_4
T_1	1	1	0	0
T_2	0	0	1	0
T_3	0	0	0	1
T_4	1	1	1	0
T_5	1	1	1	1

Then

$F_0 = \{\}$

$C_1 = \{1\},\{2\},\{3\},\{4\}$ *with support of* 3,3,3,2

$F_1 = \{1\},\{2\},\{3\},\{4\}$

$C_2 = \{1{,}2\},\{1{,}3\},\{1{,}4\},\{2{,}3\},\{2{,}4\},\{3{,}4\}$ with support of 3,2,1,2,1,1

$F_2 = \{1{,}2\},\{1{,}3\},\{2{,}3\}$

$C_3 = \{1{,}2{,}3\}$ *with support of* 2

$F_3 = \{1{,}2{,}3\}$

$C_4 = \{\}$ //STOP

The frequent itemsets of the database are then obtained as $F_1 \cup F_2 \cup F_3$

The Association Rule generation phase

We already know that for every frequent itemset Z ,and $X \subset Z$, the rule $X \rightarrow (Z - X)$ is added to our set of rules if $\frac{\sup(Z)}{\sup(X)} > minConf$[3]. Let $Z = X1 \cup Y1 = X2 \cup Y2$, then $Y1 = Z - X1$ and $Y2 = Z - X2$. If we have $Y2 \subset Y1 \subset Z$, it implies $X1 \subset X2 \subset Z$ and $\sup(X1) \geq \sup(X2)$. Then, if $conf(X1 \rightarrow Y1) = \sup(Z)/\sup(X1) >$ minConf, then $conf(X2 \rightarrow Y2) = \sup(Z)/\sup(X2)$ will also be above minConf. And if $conf(X2 \rightarrow Y2) = \sup(Z)/\sup(X2) <$ minConf, then $conf(X1 \rightarrow Y1) = \sup(Z)/\sup(X1)$ will also be below minConf. In general, if a confidence of a rule with a (k) consequent Y (i.e. Y containing k items) is below the minimum support threshold, then the confidence of a rule with a (k+1) superset of Y as its consequent will also be below the minimum support threshold.

The Apriori algorithm uses the above fact to prune rules which don't satisfy the minimum confidence threshold. For every frequent itemset, the Apriori algorithm starts from an empty set and identifies candidate consequents of size 1 which are subset of the frequent itemset. Then, for each candidate consequent, it finds subsets of the frequent itemset to become the antecedents for rules and for those antecedents resulting in rules with confidence above the minimum confidence threshold, the rules will be added to our set of rules. Then for the same frequent itemset, we generate candidate consequents of size 2 that are subsets of the frequent itemset. But here, we make sure the candidate consequents are only from those supersets of the candidates of size 1 which were not pruned during the previous step. Then, for each candidate consequent, we again find antecedents resulting in rules with confidence above the minimum confidence threshold and we add those rules to our set of rules. We continue the above procedures until the candidate list is empty for each frequent itemset. The steps of rule generation using Apriori algorithm are shown below [3].

```
For all $Z \in F$ //for all frequent itemsets
{
    R = $\emptyset$ //i.e. initialize rules derived from the current frequent itemset to empty
    $C_1 = \{\{i\} \mid i \in Y\}$ //i.e. generate candidate consequents of size 1
    While $C_k \neq \emptyset$ //i.e. while the list of candidate consequents is not empty
    {
        //Find antecedents resulting in a rule i.e.
        $A_k = \{A \in (Z - C_k) \mid conf(A \rightarrow (Z - A)) > minConf\}$

        //Add rules containing the antecedents $A_k$ to our rules list
        $R = R \cup \{A \rightarrow Z - A \mid A \in A_k\}$

        //generate new candidate consequents of size k+1 using the consequents
        of size k //that were part of the rules added at the previous step
        $C_{k+1}$ = candidateGeneration($Z - A_k$)
    }
}
```

Note that while separating the Apriori algorithm into frequent itemset mining and rule generation phases allows using different varieties of algorithms for each phase, practically, the two steps are combined so that part of the mechanism used for rule generation can also be used for finding frequent itemsets[3].

2.3. Privacy Preserving Data Mining

Most of existing data mining methods suffer from a side-effect in that they don’t keep the privacy of individuals and organizations ([9], pp vii). So, an area of data mining called privacy preserving data mining (PPDM) was developed to try to protect sensitive information from unwanted or unapproved disclosure [8]. Among the early works on privacy preserving data mining include the paper by Agrawal and Srinkat [7] in which the authors suggest a mechanism of mining data to obtain aggregated data without access to sensitive information in the data.

Privacy Preserving Data Mining includes both Data hiding and Knowledge hiding methodologies ([9], pp vii). Data hiding methodologies are used to remove sensitive data in the version of the data that will be given to data mining tools where as knowledge hiding methodologies try to sanitize the data so that sensitive knowledge can’t be mined from the released data using the current data mining tools.

2.4. Algorithms for Privacy Preserving Association Rule Mining

The three main goals of privacy preserving association rule mining (association rule hiding) are the following [10]. I) Any rule that is considered sensitive and that can be mined from the original dataset at a specified minimum support and confidence values should not be mined from the sanitized database at the same minimum support and confidence values ii) All non-sensitive rules that can be mined from the original database at specified minimum support and confidence values should also be mined from the sanitized database at the same minimum support and confidence values iii) Any rule that can’t be mined from the original database at specified minimum support and confidence values should not be mined from the sanitized database at the same minimum support and confidence values i.e. no ghost rules should be created in the sanitized database. In addition to the above three main goals association rule mining algorithms are also desired to be scalable to handle large amount of data and not to have an exponential time complexity.

Exact association rule hiding approaches try to meet all three main goals of privacy preserving association rule mining listed above [10]. On the other hand, non-exact hiding approaches try to provide an approximate feasible solution.

There are three main types of association rule hiding algorithms i) heuristic algorithms ii) border-based algorithms iii) exact algorithms[9]

2.4.1. Heuristic Algorithms

Heuristic algorithms are fast and efficient sanitization algorithms [11]. They can be either data distortion techniques which work by replacing 1's by 0's or 0's by 1's or they can be data blocking techniques which work by replacing 0's or 1's by unkowns "?". But these approaches suffer from unwanted side-effects in that the heuristics they use usually make locally best decisions which may not be globally best and thus causing heuristic approaches to find approximate solutions whose proximity to the optimal solution can't be guaranteed([9], chapter 3)

2.4.2. Border-based approaches

There exists a border that separates the frequent itemsets from the infrequent ones in the lattice of all itemsets [9]. Moving this border to exclude sensitive itemsets from the frequent itemsets will have an impact on non-sensitive frequent itemsets. So, border-revision approaches try to revise the original border in such a way that sensitive itemsets will be excluded from the frequent itemsets with minimal impact on the non-sensitive itemsets i.e. with minimal impact on the original border.

For example, let us be given the items shown in the figure below where the frequent itemsets are to the left of the border line where as the infrequent itemsets are to the right of it. .And let the sensitive itemsets be $S = \{e, ae, bc\}$

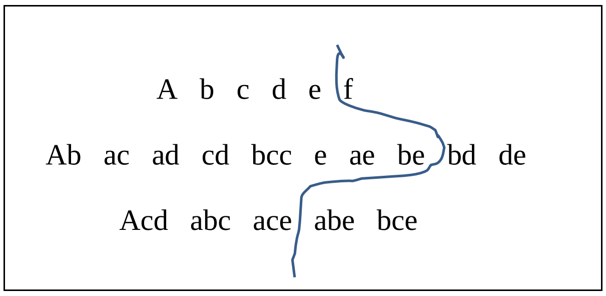


Figure 2.4. The original border separating frequent and infrequent itemsets

The border revision technique revises the above border in such a way that the new border excludes from the frequent itmesets the sensitive itemsets as well as their supersets.

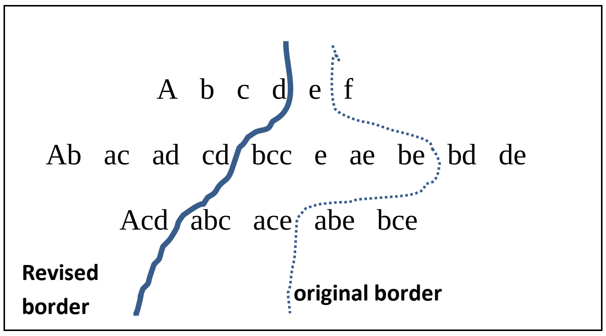


Figure 2.5. Revised border separating frequent and infrequent itemsets

2.4.3. Exact Approaches

In exact approaches, association rule hiding is modelled as a constraint satisfaction problem, which is solved using integer programming ([9], chapter 3). The solutions obtained by these approaches are optimal solutions, which have minimal side-effects during hiding of sensitive itemsets. Most of these approaches are derived from border based approaches in that they try to minimize the effect on the border during sanitization of the database under a set of constraints.

Examples of exact approaches include i) the inline algorithm which formulates a measure of distance between the original and sanitized database and that tries to minimize this distance. ii) the two-phase iterative algorithm in which during phase 1 the inline algorithm is used for sensitive knowledge hiding and if the phase 1 fails, some constraints are removed in phase 2 until the constraint satisfaction problem (CSP) becomes feasible and iii) the hybrid algorithm in which carefully crafted transactions are added to extend the original database that will enable hiding of sensitive patterns.

2.5. Optimization

Optimization problems are aimed at maximizing/minimizing some objectives [15]. They arise in a variety of fields ranging from engineering to everyday activities where the objective is to reduce the costs while attempting to maximize benefits. All optimization problems can be expressed in the following general form.

$$\text{maximize / minimize } f(x) \qquad \text{where } x = (x_1, x_2, \ldots, x_n)$$

$$s.t. \quad \emptyset_i(x) = 0 \qquad \text{for } i = \{1,2,\ldots,K\}$$

$$\varphi_j(x) \geq 0 \qquad \text{for } j = \{1,2,\ldots,L\}$$

The variables x = $\{x_1, x_2, \ldots, x_n\}$ are called decision variables and they can be continuous, discrete or mixed. The function f(x) is called the objective function i.e. the function to be optimized (maximized / minimized) [15]. The range of different combination of values the decision variables can have is called the search space where as the different values the objective function can have within this search space is called solution space. $\emptyset_i(x)$ represents a set of K equality constraints and $\varphi_j(x)$ represents a set of L inequality constraints.

<u>Some classification of optimization problems</u>

If the constraints $\emptyset_i(x)$ and $\varphi_j(x)$ are linear, the optimization problem is called a linearly constrained problem [15]. If in addition to $\emptyset_i(x)$ and$\varphi_j(x)$, the objective function f(x) is also linear, the optimization problem is called a Linear Programming Problem (LIP). If in a linear programming problem, all decision variables are required to be integers, the linear programming is called Integer Programming or Integer Linear Programming (IP or LIP). Note the term programming in the above definitions is not used in the computer programming sense and it is used to imply planning.

2.6. Integer Programming

Integer programming is a mechanism to model and solve discrete optimization problems from various disciplines [13]. It is used to solve many real-life optimization problems like the knap-sack problem and travelling salesman problem [12]. A representation of an Integer Programming problem contains a set of constraints which help to create the set of alternative feasible solutions and an objective function used to determine the optimal solution among the set of candidate feasible solutions.

A Linear Integer Programming Problem (LIP) is an integer programming (IP) problem which has a linear objective function and a set of linear equalities as constraints [12]. In this paper, when we talk about Integer Programming (IP) we are usually talking about Linear Integer Programming (LIP) problem.

The general (canonical) form of Linear Integer Programming is given as follows [12]:

$$\min cx$$

$$\text{s.t. } Ax \leq b$$

$$\mathrm{x} \geq 0, \mathrm{x} \in \mathrm{z}^{\mathrm{n}}$$

where x is a vector of n integers, A is a matrix of mxn dimension, b is a vector of m numbers and c is a vector of n numbers

Linear Integer Programming (LIP) problems are different from Linear Programming (LP) problems in that while LP problems have a convex feasible region, LIP problems have a lattice of feasible integer points [12]. This means while a local solution is also a global solution for LP problems, a local solution may not be global for LIP problems. Thus, for LIP problems, an obtained local solution must be verified that it is the global solution in order to be accepted as the optimal solution for the IP problem.

2.6.1. Formulation of Integer Programming Problems

Optimization problems can be formulated as Integer Programming problems if their feasible region is finite [14]. But it is important to note that there are many optimization problems, even simple ones, which cannot be formulated as Integer Program because their feasible region is infinite. The following example shows how an Integer Programming problem can be formulated from a verbal formulation for the traveling sales man problem.

The travelling salesman problem is verbally formulated as follows [14]. Given a set of N vertices and arcs(i,j) between any two of the vertices s.t. i,j = 1,2,..,N and where d_{ij} is the length of arc(i,j), then the objective is to find the shortest Hamiltonian circuit that passes through all N vertices by touching each vertex at most once.

To convert the above verbal formulation into a combinatorial optimization problem formulation, we need to introduce a binary variable x_{ij} where x_{ij} will be set to 1 if arc (i,j) is chosen to be in the Hamiltonian circuit. So, the travelling salesman problem can now be formulated as the following combinatorial optimization problem.

$$\min \sum_{ij} d_{ij}x_{ij}$$
$$\text{s.t. } x \in S$$

where S is the set 0-1 vector of of x_{ij} variables in a Hamiltonian circuit.

Finally, the above combinatorial optimization problem can be converted into an IP problem as follows

$$\min \sum_{ij} d_{ij}x_{ij}$$
$$\text{s.t. } \sum_{i} x_{ij} = 1\text{ , for } i = 1,2,...,N$$
$$\sum_{j} x_{ij} = 1\text{ , for } j = 1, 2... N$$
$$x_{ij} = 0 \text{ or } 1\text{, for } i, j = 1, 2... N$$

2.6.2. Algorithms for Solving IP problems

Most integer programming problems are NP hard [13].NP hard problems are problems for which no exact algorithm exists to solve them polynomially on the order of the problem size [12]. However, there are a number of exact and approximate non-polynomial algorithms devised to solve Integer Programming problems [12].

2.6.2.1. Exact Algorithms for solving Integer Programs

Exact approaches are based on identifying the mathematical structure exhibited by a problem and analyzing the polyhedron associated with that structure. The major categories of exact approaches are i) Cutting Plane Algorithms ii) Enumerative Approaches, Branch And Bound, Branch And Cut And Branch And Price Methods and iii) Relaxation And Decomposition Techniques[12].

I) Cutting Plane Algorithms

Cutting plane approaches work by representing the set of constraints in the IP problem as a convex set of feasible points of the problem [12]. These set of feasible points form the vertices of a convex polyhedron which is formed by intersection of a finite number of half spaces where each half space comes from a particular constraint.

The steps for solving an IP problem using cutting planes method are as follows [12]. First the integrality constraints on the variables of the IP problem are relaxed. Then, the resulting linear program is solved over the constraints. If the linear program is infeasible / unbounded, the IP problem is also infeasible/unbounded. But, if by luck, the solution of the linear program turns out to be integral solution, then the obtained solution is optimal solution for the IP problem. But, if the linear program doesn't have a feasible integral solution a 'facet-identification problem' will be solved (also called a separation problem because we are trying to find a plane/facet that separates the non-integral solution point from the rest of the feasible region). The solution of the 'facet identification problem' generates a linear inequality which cuts-off the fractional IP solution while keeping all other feasible integral solution points intact. In other words, the fractional solution point is removed from the set of feasible solution vertices composing a polyhedron. The algorithm stops if one of the following three conditions is met:

i) if the solution of the LIP problem which is obtained as a result of relaxation gives an integral solution

ii) if the IP is infeasible because the solution of LIP obtained by relaxing it is infeasible

iii) if the facet-identification problem doesn’t generate a cutting-linear inequality(a cut).

If the algorithm stops as a result of the third condition, it means the search area has been narrowed to the maximum that no additional cut to further narrow it down can be generated [12]. In this case, the solution obtained when the algorithm is terminated is very close to the optimal integer solution value.

II) Enumerative Approaches

A) Explicit enumeration

Explicit enumeration is the simplest of the enumerative approaches [12]. In this approach, all possible feasible solutions to the IP problem are enumerated before the optimal solution is determined. This approach is applicable and feasible only if the list of possible feasible solutions is small in number. The approach is not applicable for large problems because as the size of the problem increases the list of possible solutions increases exponentially. Some better enumerative algorithms for solving IP problems are Branch and Bound, Branch and Cut and Branch and Price algorithms.

B) Branch and Bound

Introduction to Branch and Bound

Branch and bound is the most commonly used of the enumerative approaches [12]. In the name branch and bound, branching refers to the enumeration part of the algorithm while bounding refers to the fathoming of candidate solutions by comparing them with the bounds on the objective function value. When fathoming candidate solutions, the solutions’ objective values are compared with the upper bound for minimization IP problems and with the lower bound for maximization problems. For minimization problems, solutions whose objective value is higher than the upper bound will be fathomed where as for maximization problems, solutions whose objective value is lower than the lower bound will be fathomed.

Branch and bound implicitly (not explicitly) enumerates the possibly many but finite number of feasible solutions of an ILP [13]. Even though the search tree increases exponentially with the size of the problem, branch and bound is able to handle such situations because it is able to eliminate and prune large number of infeasible solutions and feasible solutions which are not optimal.

Branch and Bound Algorithm

The steps for solving IP problems using branch and bound algorithms are given as follows [12]. First, the integrality constraints in the IP problem are dropped. Then, the resulting linear program obtained by the relaxation of the integrality constraints is solved. If the solution of the linear program luckily satisfies all the integrality constraints, then it means an optimal solution has been found to the IP problem. But, if the solution of the linear program is fractional, branching is performed to remove the fractional solution while keeping all other feasible integer solutions. The branching creates a search tree where the optimal solution is going to be searched and a linear program is solved for each node in the tree. Nodes in the search tree are fathomed if their LP solution is infeasible or if their LP solution is an integral solution better than the existing incumbent integral solution or if their LP solution is integral solution but worse than a known (the incumbent) integer solution.

The above steps can be expressed in the following compact algorithm form [13].

1. Choose one or more sub problems from the list of all candidate sub problems that may give the solution of the IP problem.
2. Solve the chosen sub problem/ sub problems without the integrality constraints. (This process is also called relaxation and is needed because the candidate sub problems are usually hard to solve.)
3. Fathom one or more of the chosen sub problems if the sub problems i) are infeasible ii)give non-integral solutions which don't promise a better solution than the current incumbent solution(best integral solution) iii) give an integral solution better than the current incumbent solution iv) give an integral solution which is not better than the current incumbent integral solution.
4. Apply branching on the unfathomed problems to create sub problems that will be added to the list of sub problems. (Branching is done by choosing a branching

variable and dividing the problem based on the possible values the variable can have.)

5. Repeat steps 1 to 4 until the candidate list is empty

Among the factors affecting the performance of branch and bound implementations are the use of efficient means to solve relaxations, the use of a good strategy for choosing the most promising candidate sub-problems at each iteration and the use of efficient branching strategy that will help to limit unnecessary expansion of the search tree [13].

Sequential implementations of the branch and bound algorithm choose only one sub problem for processing in a particular iteration of the branch and bound algorithm while parallel implementations of the branch and bound algorithm, on the other hand, choose multiple sub problems to be processed simultaneously in a single iteration of the branch and bound algorithm[13].

Sub-problem selection strategies for Branch and Bound

Note that in step 1 of the branch and bound algorithm we saw earlier, we initially have only one ILP in the candidate list, the input ILP [13], so we initially branch from the node corresponding to this ILP. But, during subsequent iterations of the branch and bound algorithm, many sub problems of the original ILP will be created. Thus, a selection rule is needed to choose a sub-problem/node for branching from the list of all candidate sub-problems at each iteration. The two most common sub-problem selection rules or candidate sub problem selection strategies are Depth First Search (DFS) and Best First Search (BFS).

DFS chooses the sub-problem/sub-problems recently added to the candidate list. It is a last in first out (LIFO) approach where candidates added last to the candidate list will be processed first. It is called depth first because each chosen candidate sub problem in the search process increases the depth of the search tree [13].

BFS chooses the candidate with the best bound i.e. for minimization IP problems it chooses the candidate with the least lower bound and for maximization IP problems it chooses the candidate with the highest upper bound. If multiple candidates have equal lower or upper bound, then a last in first out rule (LIFO) is used to break the tie [13].

One difference between BFS and DFS approaches is that BFS requires the bounds to be computed for all sub-problems in the candidate list [13]. To ensure that all candidates have their bounds computed, the relaxation of each sub-problem is computed before it is added to the candidate list. Another difference between the two search approaches is that BFS is suitable when our target is to minimize the number of candidate problems solved while DFS is suitable if our target is to reach the end of the search tree fast so that in the best case, in the case where an integral solution is obtained in the first leaf at the end of the search tree, the time to solve the problem is minimized.

Branching variable selection strategies

Some of the most commonly used branching strategies are listed below.

1) Greedy branching

In the greedy branching strategy, branching is done on the first fractional variable that is encountered.

2) Most infeasible branching

In the most infeasible branching strategy, the variable whose fractional part is closest to 0.5 is chosen for branching [27] i.e. the branching strategy tries to select the variable which is difficult to determine if it is close to its rounded down number or it's rounded up number. But, this branching approach doesn't result in a better performance than selecting the branching variable randomly.

3) Pseudo cost branching

Among the fractional variables, pseudo cost branching chooses the one which had the greatest improvement in the objective function when it was previously chosen as branching variable [27]. This branching strategy is a sophisticated branching strategy that needs to store the history of the variables impact on the objective function in order to choose the variable which historically produced the greatest improvement for branching.

4) Strong branching

Strong branching tests all the fractional variables for the maximum improvement that they can provide to the objective function if they are chosen as branching variable. To test the fractional variables, it temporarily branches on them and solves the resulting problems by relaxing the integrality constraints [27]. The variable, if we branch on it and solve its two sub-problems, which gives the biggest improvement in the objective function, will be selected for branching.

In conclusion, among the branching strategies strong branching in general results in small number of branches but the computational time spent on each node is long [27]. The branching strategy which was experimentally determined to result in the least computational time was pseudo cost branching.

Example

The following example shows the steps of the branch and bound algorithm. Assume we are given the following IP problem.

$\min z = 2x_1 - 3x_2$

s.t. $-10x_1 + 2x_2 \leq 5$

$3x_1 + 2x_2 \leq 9$

$x_1, x_2 \geq 0 , x_1, x_2 \in N$

We first begin the branch and bound algorithm by solving the problem as a regular linear programming problem without the integrality constraints. The solution we obtain is $x_1 = 0.31$, $x_2 = 4.04$, min z = -11.5. The lower bound then becomes LB = -11.5 and the upper bound will at first be initialized to infinity. So, node 1 appears like in this figure.

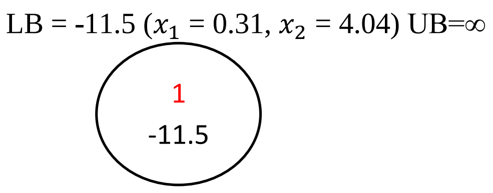


Figure 2.6. The initial node in the branch and bound algorithm

Then, from this relaxed solution, two solution subsets will be created by branching on a fractional variable. Since both of the variables are fractional in the obtained solution, we branch from the first fractional variable i.e. x_1. Since x_1 must be an integer, we add the following two constraints to the problem

$$x_1 \leq 0 \text{ Or } x_1 \geq 1$$

And we obtain the two sub-problems shown in Figure 2.7. The solutions at node 2 and 3 are obtained by relaxing the integrality constraints as in node 1 and the branch and bound tree looks as shown in Figure 2.8 after their solutions are obtained. We can see that an integral solution is obtained for node 3 and hence we update the upper bound to the LB value of node 3 as shown in Figure 2.9.

Node 3 will then be pruned since it gave integral solution and since node2's lower bound is below the new upper bound (since it is promising) we continue the branch and bound procedure on node 2. We branch from node 2 using the only fractional variable at node 2 i.e. x_2. Since x_2 must be an integer, we can add the following two constraints

$$x_2 \leq 2 \text{ Or } x_2 \geq 3$$

The solutions at node 4 and 5 are obtained by relaxing the integrality constraints and they are shown in Figure 2.10.

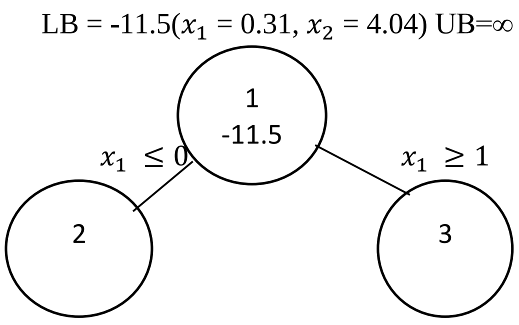


Figure 2.7. Branching on the first fractional variable x_1

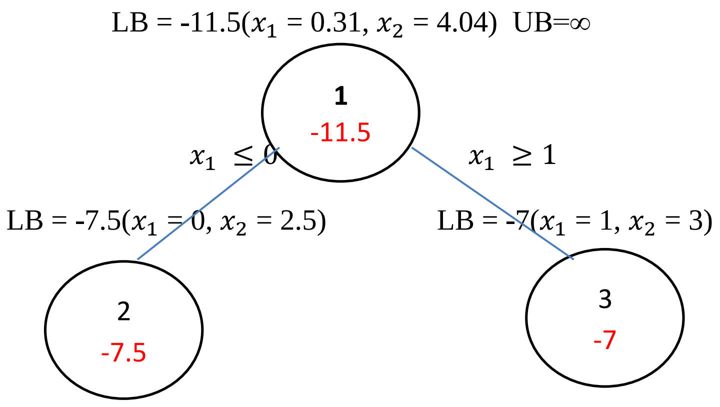
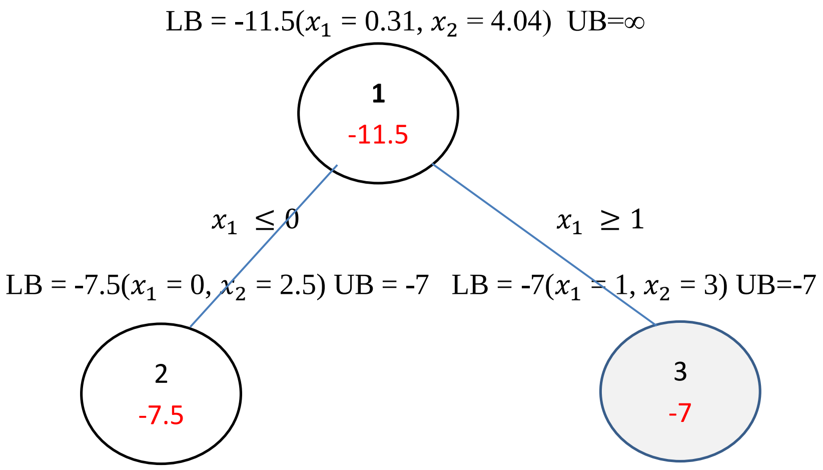
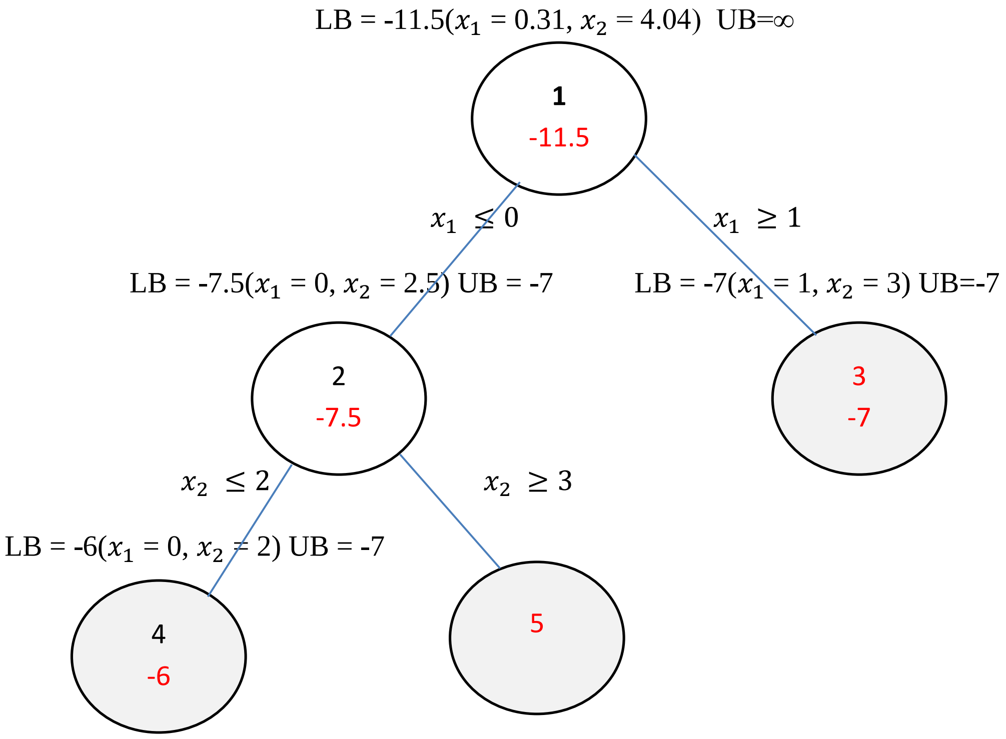

LB = -11.5(x_1 = 0.31, x_2 = 4.04) UB=∞
1
-11.5
$x_1 \leq 0$
$x_1 \geq 1$
LB = -7.5(x_1 = 0, x_2 = 2.5)
LB = -7(x_1 = 1, x_2 = 3)
2
-7.5
3
-7

Figure 2.8. Solution sub-sets after branching on x_1 of node 1



LB = -11.5(x_1 = 0.31, x_2 = 4.04) UB=∞
1
-11.5
$x_1 \leq 0$
$x_1 \geq 1$
LB = -7.5(x_1 = 0, x_2 = 2.5) UB = -7
LB = -7(x_1 = 1, x_2 = 3) UB=-7
2
-7.5
3
-7

Figure 2.9. Pruning of node 3 as it gave integral solution and updating of the upper bound



LB = -11.5(x_1 = 0.31, x_2 = 4.04) UB=∞
1
-11.5
$x_1 \leq 0$
$x_1 \geq 1$
LB = -7.5(x_1 = 0, x_2 = 2.5) UB = -7
LB = -7(x_1 = 1, x_2 = 3) UB=-7
2
-7.5
3
-7
$x_2 \leq 2$
$x_2 \geq 3$
LB = -6(x_1 = 0, x_2 = 2) UB = -7
4
-6
5

Figure 2.10. Solution subsets after branching on x_2 of node 2 and the pruning of node 5 as it is infeasible and the pruning of node 6 as it is non-promising

We can see that the LB of node 4 is above the upper bound. So, node 4 will be pruned since it is non- promising. On the other hand, node 5 will also be pruned since it is infeasible (i.e. the combination of x_1 and x_2 is outside the feasible region.)

Since now all nodes have been pruned because either they are non-promising, infeasible or integral node, the best integral solution we have obtained to this point will become the solution of the IP problem. Thus, the solution to the IP problem is $x_1 = 1$, $x_2 = 3$ with $\min z = -7$.

C) Branch and Cut

Branch and cut is an algorithm which combines the branch and bound algorithm with the cutting planes method in order to create a more powerful algorithm [13]. The algorithm dynamically adds cutting planes to the problem which hold in every part of the search tree. Branch and cut is aimed at further narrowing down the search space in the branch and bound algorithm because the bounds coming from LP relaxations in branch and bound are often weak.

D) Branch and Price

Branch and price is another enumerative approach to find solutions of IP problems [12]. It uses a technique called pricing to tighten the branch and bound algorithm in the same way cutting is used in branch and cut. Pricing is a column generation technique inside the simplex algorithm that is used to compute bounds in a branch and bound algorithm. The column generation technique is used to determine the most influential variable with negative reduced cost so that when it enters the problem (basis) it pushes the solution of the problem towards the optimal solution.

In the simplex algorithm, the only variables which are allowed to enter the basis at each iteration are those variables with negative reduced costs [12]. But, in large problems with large number of variables we will have many variables with negative reduced costs at each iteration. So, choosing the appropriate variable which would drastically improve the objective function value is important. As a result, branch and price uses the column-generation technique which is used to efficiently solve LPs so that a speed up could be obtained in the performance of the branch and bound algorithm.

2.6.2.2. Heuristic Algorithms for Solving Integer Programs

Since LIP optimization problems are NP hard and since it may take a long time to solve them, heuristic approaches are usually used to find approximate solutions. The most commonly used and powerful of the heuristic approaches are Simulated Annealing and Tabu Search [12]

Simulated Annealing

Simulated Annealing is a local search algorithm in which in contrast to traditional local search algorithms it allows occasional movement towards worse solutions in order to avoid local optima which may not be globally optimal [28].

Tabu Search

Tabu search is also a local search based method in which movement towards worse solutions is allowed to avoid local optima. Tabu search uses a short term memory called a tabu list in which solutions already considered are stored so that movement towards them is not allowed in order to avoid cycling [12].

2.7. Linear Programming Problems

Linear programming finds the maximum or minimum value of an objective function under linear constraints [15]. The general form of linear programs is given as follows.

$$\min z = cx$$

$$\text{s.t. } Ax \leq b$$

$$x \geq 0$$

There are three main approaches for solving linear programming problems i) The Graphical Approach ii) The Simplex Method and iii) Interior Point Algorithms

2.7.1. Graphical Approach

The graphical method works by plotting all constraints in the LP problem as straight lines [15]. These set of straight lines form a polygon and the inside of the polygon contains all feasible solutions that satisfy all constraints of the LP problem. The minimum or maximum value of the objective function lies at one of the vertices (extreme points) of the polygon. Thus, the objective function value is computed at each of the vertices of the polygon and the vertex resulting in the maximum/minimum value for the objective function will become the solution point.

Example: If we are given the following LP problem

$$\min z = f(x_1, x_2) = 2x_1 - 3x_2$$
$$\text{s.t.} \quad -x_1 + 2x_2 \le 2$$
$$x_1 + 2x_2 \le 6$$
$$x_1, x_2 \ge 0$$

And if we want to solve the LP problem using the graphical approach, we first draw lines corresponding to the equality form of the constraints. The plot of the straight lines for the above problem is shown below.

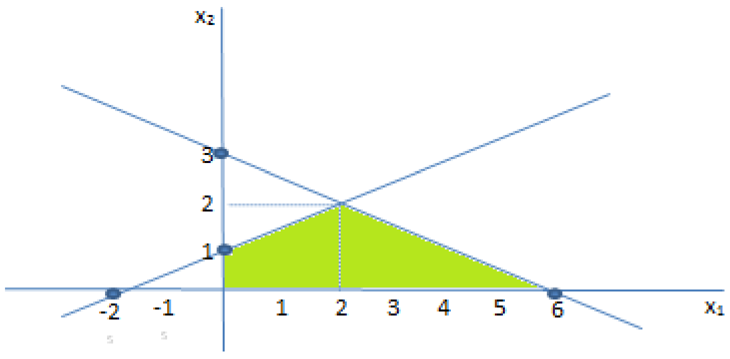


Figure 2.11. The feasible region of the given LP problem.

The shaded region is the inside of the polygon formed by the intersection of the lines corresponding to the constraints. The polygon has four vertices and the optimal solution for the LP problem lies at one of these vertices. To determine which vertex gives the optimal solution, we have to evaluate the objective function at each of the four vertices.

$f(0,0) = 0$, $f(0,1) = -3$ $f(2,2) = -2$ $f(6,0) = 12$

Since the vertex (extreme point) of the polygon which gave the minimum value for the objective function is (0,1) , the optimal solution will be $x_1 = 0$, $x_2 = 1$ and min z = -3

Note however that the graphical approach can be used to solve LP problems only when the number of decision variables and constraints is small. As a result, it is not applicable for most real-world LP problems which contain hundreds of (thousands of variables). For such large scale problems we have to use to use other approaches than the graphical method.

2.7.2. Simplex Method

Generally, the set of constraints in an LP problem form a polyhedron (Note: A polyhedron is a 3d or multidimensional equivalent of a polygon) where the inside of this polygon contains all feasible solutions [15]. The optimum feasible solution (i.e. the solution with maximum/minimum value for the objection function) lies in one of the vertices of the polyhedron. But, even for simple LP problems, the set of vertices (extreme points) of the polyhedron may be quite large [15]. So, checking all the vertices in order to determine the optimal solution point will not be feasible.

So, the simplex method was developed to efficiently traverse along selected vertices of the polyhedron so that the optimal solution point or vertex can be obtained with small number of the vertices of the polyhedron being visited. The method was developed by George B. Dantzig in 1947 as a method of moving from one extreme point to another extreme point of the polyhedron of feasible solutions while improving the objective function value (or at least not making it worse) until an optimal solution is found or the problem is determined to have infinite number of optimal solutions.

The simplex method starts from a given extreme point and tests whether the extreme point is optimal or not using an optimality test derived from the objective function and the constraints [15]. If the extreme point fails the optimality test, an adjacent extreme point will be selected and the optimality test is done on the new extreme point. The above processes are repeated until an optimal extreme point that passes the optimality test is found or until the IP problem is determined to be unbounded.

The two major parts of the simplex method are i) a mechanism to test whether a given basic feasible solution(see the section below for what a basic feasible solution means) is optimal or not and ii) a mechanism to find an adjacent basic feasible solution if the current basic feasible solution is found to be not optimal[15].

2.7.2.1. Basic Feasible Solution

The basic feasible solution for the linear system Ax=b, where A is mxn matrix and x is nx1, which makes the constraints of the linear program min z=f(x) s.t. Ax=b and $x \geq 0$ is obtained by setting n-m variables equal to zero and solving the remaining mxm system of linear equations in order to obtain a unique solution for the m variables which were not set to zero [15]. The n-m variables which were set to zero are called non-basic variables where as the m variables which were not set to zero are called the basic variables of the basic solution.

The basic feasible solution lies at one of the corner points of the polyhedron which contains all feasible solutions [15]. The basic feasible solution has the characteristics that for every basic feasible solution there is a unique vertex of the polyhedron and for every vertex of the polyhedron, there exists exactly one basic feasible solution. Two basic feasible solutions for the system Ax = b, where A is mxn, are said to be adjacent if they share m-1 of their m basic variables.

2.7.2.2. Format of Input LP Problem to the Simplex Method

The simplex method requires the input IP problem to be expressed in standard form as follows [15].

$$\begin{aligned} &\min z = cx \\ &\text{s.t. } Ax = b \\ &\; x \geq 0 \end{aligned}$$

where A is mxn matrix, b is mx1 vector , c is an nx1 vector and x is also nx1 vector.

In the standard form of an LP problem, all the constraints in the LP problem are required to be expressed as equalities. Slack variables (surplus variables) are used to transform constraints containing inequalities into equalities.

For example if we are given an IP problem not in standard form as follows

$$\begin{aligned} &\min z = 2x_1 - 3x_2 \\ &\text{s.t. } -x_1 + 2x_2 \leq 2 \\ &\qquad x_1 + 2x_2 \leq 6 \\ &\qquad x_1, x_2 \geq 0 \end{aligned}$$

We can convert the first inequality in the constraints into equality by adding a new variable x_3 to the left hand side of the inequality to obtain the following equality.

$$x_1 + 2x_2 + x_3 = 2$$

The variable x_3 in the above equation is the slack variable.

Similarly, a slack variable x_4 can be added into the second inequality of the constraints by which point we will have all the constraints expressed as equality as follows, which is the standard form of the given IP problem.

$$\begin{aligned} &\min z = 2x_1 - 3x_2 \\ &\text{s.t.} \quad -x_1 + 2x_2 + x_3 = 2 \\ &\qquad\quad x_1 + 2x_2 + x_4 = 6 \\ &\qquad\quad x_1, x_2 \geq 0 \end{aligned}$$

2.7.2.3. Theory of the Simplex Method

The theory behind the simplex method which is elaborated in [16] is summarized in this paper as follows.

Let us be given the following LP problem

$$\begin{aligned} &\min z = cx \\ &\text{s.t. } Ax = b \\ &x \geq 0 \end{aligned}$$

, where A is mxn matrix, b is mx1 vector, c is an nx1 vector and x is also nx1 vector.

If $\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_n$ are mx1 column vectors that constitute A i.e. A = ($\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_n$), then we will have

$$Ax = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_n\mathbf{a}_n = \mathbf{b} \qquad (2.1)$$

If we are also given the following basic feasible solution to the above LP problem,

$$x = (x_1,x_2,\ldots,x_m,0,0\ldots0)$$

then, **b** and z will be expressed as

$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_m\mathbf{a}_m = \mathbf{b} \qquad (2.2)$$

$$x_1\mathbf{c}_1 + x_2\mathbf{c}_2 + \ldots + x_m\mathbf{c}_m = z \quad (2.3)$$

Since the column vectors $\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_m$ are linearly independent, all of the column vectors of A including those corresponding to the zero-valued x variables can be expressed in terms of $\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_m$ as follows.

$$\bar{a}_{1\mathrm{j}}\boldsymbol{a_1} + \bar{a}_{2\mathrm{j}}\boldsymbol{a_2} + \cdots + \bar{a}_{m\mathrm{j}}\boldsymbol{a_m} = \boldsymbol{a_j} \text{ for j = 1,2,...,n} \quad (2.4)$$

Using the scalars $\bar{a}_{1\mathrm{j}}, \bar{a}_{2\mathrm{j}}, \ldots, \bar{a}_{m\mathrm{j}}$ in the above equation, we can define a new quantity called z_j which is expressed as a linear combination of the objective function coefficients corresponding to the non-zero variables as follows.

$$\bar{a}_{1\mathrm{j}}\boldsymbol{c_1} + \bar{a}_{2j}\boldsymbol{c_2} + \cdots + \bar{a}_{mj}\boldsymbol{c_m} = z_j \text{ for j = 1,2,...,n} \quad (2.5)$$

If we multiply equation (4) by some positive number θ and subtract it from equation (2) and similarly if multiply equation (5) by the same positive number θ and subtract it from equation (3), we will arrive at the following two equations.

$$(x_1 - \theta\bar{a}_{1j})\boldsymbol{a_1} + (x_2 - \theta\bar{a}_{2j})\boldsymbol{a_2} + \cdots + (x_m - \theta\bar{a}_{mj})\boldsymbol{a_m} + \theta\boldsymbol{a_j} = \boldsymbol{b} \quad (2.6)$$

$$(x_1 - \theta\bar{a}_{1j})c_1 + (x_2 - \theta\bar{a}_{2j})c_2 + \cdots + (x_m - \theta\bar{a}_{mj})c_m + \theta c_j$$
$$= z - \theta(z_j - c_j) = z' \quad (2.7)$$

, where θc_j has been added to both sides of (7)

*** If for some j corresponding to non-basic variables all $\bar{a}_{i\mathrm{j}}$ are negative, then for some positive θ, a new basic feasible solution x will be obtained with m+1 basic variables i.e. x = (x_1',x_2', ...,x_m', x_j ,0,0,...,0) , where x_1' = $(x_1 - \theta\bar{a}_{1\mathrm{j}})$, x_2' = $(x_2 - \theta\bar{a}_{2\mathrm{j}})$ ..., $x_j = \theta$ and where the equivalents of equations (2) and (3) for the new basic feasible solution will be given like this.

$$x_1'\mathbf{a}_1 + x_2'\mathbf{a}_2 + \ldots + x_m'\mathbf{a}_m + x_j a_j = \mathbf{b} \quad (2.8)$$

$$x_1'\mathbf{c}_1 + x_2'\mathbf{c}_2 + \ldots + x_m'\mathbf{c}_m + x_jc_j = = z - x_j(z_j - c_j) = z' \qquad (2.9)$$

Theorem : Assuming a basic feasible solution is non-degenerate, if for some j corresponding to a non-basic variable, if $\bar{a}_{ij} > 0$ for at least one i, i = 1,2,...,m, then it is possible to generate a new basic feasible solution with just m-positive variables that give a better(lesser) objective value than the previous basic feasible solution i.e. from a previous basic feasible solution of x = ($x_1,x_2,...,x_m$,0,0...0), a new basic feasible solution of . x = ($x_1',x_2', ...,x_m', x_j$,0,0,...,0) can be generated such that z' < z.

Proof: If we have $\bar{a}_{ij} > 0$ for some i, i = 1,2,...,m, the maximum value θ can attain before making the coefficient of any a_i negative in (6) is , $\frac{x_i}{\bar{a}_{ij}}$. If we have many such positive $\bar{a}_{ij}$ values, then the largest value of θ which can maintain the non-negativity restrictions on the new variables x_1', x_2' ...etc in (8) is

$$\theta = \frac{x_r}{\bar{a}_{rj}} = \min \frac{x_i}{\bar{a}_{ij}} \quad \text{for } \bar{a}_{ij} > 0 \text{ , for i = 1,2,...,m} \qquad (2.10)$$

, where the minimum is obtained for some unique i, i =r.

Equation 2.10 is called the minimum ratio test where the minimum is obtained for some unique i, i = r. For a θ value obtained using the minimum ratio test, the coefficient of the $\mathbf{a}_\mathbf{r}$ vector will be zero in equation (8), so the new basic feasible solution will be x = ($x_1',x_2',.., x_r'...,x_m', x_j$,0,0,...,0) where x_r' = 0 for some r in 1 to m corresponding to the minimum ratio test, meaning the solution contains just m basic variables.

So, if $\bar{a}_{ij} > 0$ for at least one i, i = 1,2,...,m, then the equivalent of equations (2.8) and (2.9) will be the following two equations.

$$x_1'\mathbf{a}_1 + x_2'\mathbf{a}_2 + \ldots + 0\mathbf{a}_\mathbf{r} + \ldots + x_m'\mathbf{a}_m + x_ja_j = \mathbf{b} \qquad (2.11)$$

$$x_1'\mathbf{c}_1 + x_2'\mathbf{c}_2 + \ldots + 0\mathbf{a}_\mathbf{r} + \ldots x_m'\mathbf{c}_m + x_jc_j = z - x_j(z_j - c_j) = z' \qquad (2.12)$$

Thus, in the above equations a previously non-basic variable x_j enters the basic variables while a previously basic variable x_r leaves the basis to become non-basic. The above operations which resulted in the exchange of the two variables are called pivoting. If the pivoting operation is applied using some j corresponding to a non-basic variable such that the quantity $(z_j - c_j)$, also called reduced cost , is greater than zero, then the new basic feasible solution will result in an improved(lesser) objective value of z' compared to z of the previous basic feasible solution.

As long as the reduced cost $(z_j - c_j)$ > 0 (or equivalently as long as the negative reduced cost -$(z_j - c_j)$=$(c_j - z_j)$ is less than 0), the objective function can be improved (made less) for some non-zero values of the entering variable x_j (i.e. for non-degenerate cases). Thus, we can continue with the next iteration of the simplex algorithm by performing pivot operations.

But, if we reach an iteration where the reduced cost $(z_j - c_j)$ $\leq$ 0 (or equivalently the negative reduced cost $(c_j - z_j)$ $\geq$ 0), it means the objective function can't be improved by further pivot operations (i.e. by moving to new feasible solution points) and it means the current solution is optimal.

2.7.2.4. The Simplex Tableau

Simplex tableau is a method of expressing a linear program problem which allows the simplex method to be applied efficiently when solving the problem [16]. The tableau displays all the quantities that are needed at each iteration of the simplex algorithm. The general form of the simplex tableau is as shown below.

c_B	a_B	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_j}$	$\mathbf{a_n}$	$X_B = b$	$\frac{b_i}{\bar{a}_{ij}}$
c_{B_1}	$\boldsymbol{a_{B_1}}$	$\bar{a}_{11}$	$\bar{a}_{12}$	$\bar{a}_{1j}$	$\bar{a}_{1n}$	b_1	$\frac{b_1}{\bar{a}_{1j}}$
c_{B_2}	$\boldsymbol{a_{B_2}}$	$\bar{a}_{21}$	$\bar{a}_{22}$	$\bar{a}_{2j}$	$\bar{a}_{2n}$	b_2	$\frac{b_2}{\bar{a}_{2j}}$
c_{B_m}	$\boldsymbol{a_{B_m}}$	$\bar{a}_{m1}$	$\bar{a}_{m2}$	$\bar{a}_{mj}$	$\bar{a}_{mn}$	b_m	$\frac{b_m}{\bar{a}_{mj}}$
c		c_1	c_2	c_j	c_n		
$c_j - z_j$		$c_1 - z_1$	$c_2 - z_2$	$c_j - z_j$	$c_n - z_n$		

Figure 2.12. The simplex Tableau

The column with the heading a_B contains the column vectors of A that are in the basis i.e. the row headings a_{B_1}, a_{B_2},..., a_{B_m} correspond to the column vectors of A that are in the basis. The column with the heading c_B contains the coefficients of the objective function corresponding to the vectors in the a_B column. The column headings **a_1,a_2,..a_j,...,a_n** (not the columns themselves) correspond to the column vectors of A. The coefficients $\bar{a}_{11}$, $\bar{a}_{12}$, ..., $\bar{a}_{mn}$ are used to express **a_1,a_2,..a_j,...,a_n** and z_1,z_2,...,z_n as a linear combination of $\boldsymbol{a_{B_1}}$, $\boldsymbol{a_{B_2}}$,..., $\boldsymbol{a_{B_m}}$. For instance, a_j and z_j can be expressed as follows.

$$\bar{a}_{1j}\boldsymbol{a_{B_1}} + \bar{a}_{2j}\boldsymbol{a_{B_2}} + \cdots + \bar{a}_{mj}\boldsymbol{a_{B_m}} = \boldsymbol{a_j}$$
$$\bar{a}_{1j}c_{B_1} + \bar{a}_{2j}c_{B_2} + \cdots + \bar{a}_{mj}\boldsymbol{c}_{c_{B_m}} = z_j$$

(Note that for the column vectors of A that are in the basis, the vector of $\bar{a}_{ij}s$ used to to express them as a linear combination of $\boldsymbol{a_{B_1}}$, $\boldsymbol{a_{B_2}}$,..., $\boldsymbol{a_{B_m}}$ are unit vectors.)

The column with the heading $X_B = b$ stores the current values of the basic variables. Assuming **a_j** is the vector to enter the basis, the column of the tableau with heading $\frac{b_i}{\bar{a}_{ij}}$ contains the ratios $\frac{b_i}{\bar{a}_{ij}}$. The row heading c corresponds to the coefficients of the variables x_1,..,x_n in the objective function and the row heading $c_j - z_j$ corresponds to the negative reduced cost for the variables.

The tableau shown in the previous figure can be used to select the vector that will enter the basis based on which non-basic column has negative reduce cost $c_j - z_j$ (or the least negative reduced cost if there are multiple non-basic columns with negative reduced cost). If an entering variable cannot be determined, it means the current solution is optimal. After the entering variable x_j is determined the leaving variable, x_r can be determined by choosing a unique row i, i = r , r in 1,2,..m, corresponding to the minimum ratio $\frac{b_i}{\bar{a}_{ij}}$ where $\bar{a}_{ij} > 0$ (i.e. using minimum ratio test). Then, a pivot operation can be applied to make the vector of $\bar{a}_{ij}s$ that are used for expressing the column vector a_j a unit vector.

Example

Let us be given the following LP.

$$\min z = 2x_1 - 3x_2$$
$$\text{s.t.} \quad -x_1 + 2x_2 \leq 2$$

$$x_1 + 2x_2 \leq 6$$
$$x_1, x_2 \ \geq 0$$

We will first express it in standard form as follows by the addition of slack variables.

$$\begin{aligned} &\min z = \ 2x_1 - 3x_2 \\ &\text{s.t.} \qquad -x_1 + 2x_2 + x_3 = 2 \\ &\qquad\qquad x_1 + 2x_2 + x_4 = 6 \\ &\qquad\qquad x_1, x_2 \ \geq 0 \end{aligned}$$

Then the first Tableau will be as follows.

c_B	a_B	**a₁**	**a₂**	**a₃**	**a₄**	X$_B$ = b	$\frac{b_i}{\bar{a}_{ij}}$
0	x_3	-1	2	1	0	2	1 $\longrightarrow$
0	x_4	1	2	0	1	6	3
c		2	-3	0	0		
$c_j - z_j$		2	-3	0	0		
			$\uparrow$				

Figure 2.13. Tableau 1

Where $\uparrow$ indicates the vector which shall enter into basis because of having the least negative reduced cost while $\longrightarrow$ indicates the vector which will leave the basis because it corresponds to the row with minimum non-zero ratio for $\frac{b_i}{\bar{a}_{ij}}$. Then Tableau 2 will look like this.

c_B	a_B	**a₁**	**a₂**	**a₃**	**a₄**	$X_B = b$	$\frac{b_i}{\bar{a}_{ij}}$
-3	x_2	$-1/2$	1	1/2	0	1	
0	x_4	2	0	-1	1	4	
c		2	-3	0	0		
$c_j - z_j$		$1/2$	3	3/2	0		

Figure 2.14. Tableau 2

Since we don't have negative reduced cost in Tableau 2, it means the current solution is optimal, where the optimal solution is given by

$$x_1 = 0, x_2 = 1 \text{ and } \min z = -3$$

2.7.2.5. Algebraic derivation of the simplex method

The algebraic derivation of the simplex method is illustrated in [17] where it is summarized in this paper as follows.

Assume we are given an IP problem in standard as follows

$$\begin{aligned} &\min z = cx \\ &\text{s.t. } Ax = b \\ &x \geq 0 \end{aligned}$$

If B is an initial basic matrix of A and N is the corresponding non-basic matrix of A corresponding to an initial basic feasible solution of $x = (x_B, x_N)$, then the constrains of the LP problem can be expressed as follows.

$$Ax = Bx_B + Nx_N = b \tag{2.13}$$

Similarly, we can partition c into cB and cN to express the objective function as follows

$$z = cx = c_B x_B + c_N x_N \tag{2.14}$$

From equation (2.13), we have

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{2.15}$$

And substituting this into Equation (2.14), we will get

$$z = cx = c_B(B^{-1}b - B^{-1}Nx_N) + c_N x_N = c_B B^{-1}b + (c_N - c_B B^{-1}N)x_N \tag{2.16}$$

Differentiating the above equation with respect to the non-basic variables we will get

$$\frac{\partial z}{\partial x_j} = c_j - c_B B^{-1} a_j$$

, for all j such that x_j is in x_N, where a_j is the j'th column of N (2.17)

The quantity $c_j - c_B B^{-1} a_j$ = negRedCost$_j$ is called the negative reduced cost for the x_j variable.

If one x_j from the x_N vector in the basic feasible solution becomes non-zero while the other elements of x_N remain zero, then the new value of the objective function can be computed using equation (16) as follows

$$z' = z + (c_j - c_B B^{-1} a_j)x_j = z + \text{negRedCost}_j * x_j \tag{2.18}$$

Using the above algebraic representation, the procedures for selecting the entering and leaving variables in the simplex method are as follows [17].

Entering Variable Selection Criteria

For a negRedCost$_j$ < 0 and a positive value of x_j , the new objective function value corresponding to the new feasible solution that is going to be determined will be less than the current objective function value corresponding to the current feasible solution. Thus, we chose the entering variable x_j from the elements of x_N whose $negRedCost_j < 0$ and such that it results in the minimum negative reduced cost so that for a slight increase of x_j from its previous 0 value, the value of the objective function decreases by a large amount.

Leaving variable Selection Criteria

After we chose x_j from x_N as the entering variable, equation (2.15) will now reduce to

$$x_B = B^{-1}b - B^{-1}a_j x_j \text{ , where } a_j \text{ is the j'th column of N} \tag{2.19}$$

Then, equation (2.19) can be re-written as

$$x_B = \bar{b} - \bar{a}_j x_j \text{ , where } \bar{b} = B^{-1}b \text{ and } \bar{a}_j = B^{-1}a_j \tag{2.20}$$

If all elements of the vector $\bar{a}_j$ are less than zero, x_j can be increased indefinitely without violating the non-negativity restrictions. But if at least one element of the vector $\bar{a}_j$ is greater than 0, then for the entering variable x_j not to violate the non-negativity restriction on x's, it can be increased from zero only upto a certain maximum value. This maximum value x_j can attain is given by the minimum ratio test.

$$x_j = \min \left\{ \frac{\bar{b}_i}{\bar{a}_{j_i}} : i = 1,\ldots,m \text{ and } \bar{a}_{j_i} > 0 \right\} \tag{2.21}$$

The basic variable x_i corresponding to the minimum ratio value becomes the leaving variable.

2.7.2.6. The Revised Simplex Method

The main drawback of the simplex method is that it computes and stores many numbers which are not all needed in the next iterations [16]. This makes its computation very time consuming. So, the revised simplex method is developed to handle these drawbacks.

The revised simplex method follows the same general approach as the simplex method. But it differs from the simplex method in the way it makes calculations to move from one iteration to the next, by computing/storing only those quantities that are needed at each iteration [16].

From the algebraic derivation of the simplex method, we can see that the quantities that are required at each iteration of the simplex method are

i) $c_j - c_B B^{-1} a_j$ for all the previous non-basic variables in order to determine which non-basic variable among them enters the basis

ii) $\bar{a}_j = B^{-1} a_j$ which is used in the minimum ratio test to determine which previous variable will leave the basis

iii) The value of the basic variables x_B = $\bar{b} = B^{-1} b$ which is used in the minimum ratio test to determine which previous basic variable will leave the basis

iv) And optionally, the previous objective function value z = $c_B x_B$ = $c_B B^{-1} b$ inorder to determine the new objective function value of the new basic feasible solution that will result after the end of the current iteration i.e. z' = z + negRedCost$_{j*}$ x$_j$

2.7.2.7. Finding an Initial Basis

The simplex method requires an initial basic feasible solution to start its iterations [16]. For simple LP problems, the initial basic feasible solution may be determined readily by manual inspection. But, for most problems, finding an initial basic feasible solution is not trivial. Thus, special techniques are used to find the initial basic feasible solution. The two commonly used methods to find the initial basis for simplex are i) the two-phase method, developed by Dantzig, Orden and Wolfe and ii) The method of penalities, by A. Charnes. In this paper, we will look at the two-phase method.

The two phase method

In the two phase method, the linear programming problem is first expressed in standard form and then the simplex method is applied in two phases [16]. In phase 1, the simplex method is applied to a modified version of the original LP problem and the solution of this auxiliary problem, if it exists, will serve as the basic feasible solution to the original LP problem. In phase 2, using the basic feasible solution obtained from phase 1, the original problem is solved by applying the simplex method.

The steps of the two phase method

In the two phase method, the linear programming problem must first be expressed in standard form by the addition of slack/surplus variables. All the constraints whose right hand side is negative must also be multiplied by -1 to make the right hand side non-negative. Thus, the IP problem shall be expressed as follows

$$\min z = cx$$
$$\text{s.t. } Ax = b$$
$$x \geq 0 \text{ , } b \geq 0 \text{ ,}$$

where A is mxn matrix, b is mx1 vector, c is nx1 vector and x is also nx1 vector.

Phase 1

In phase 1, the original LP problem is modified by the introduction of new variables called artificial variables and by setting the coefficients of the objective function in the original LP problem to zero so that the LP problem becomes

$$\min z = 0x + e^T a$$

$$\text{s.t. } Ax + Ia = b$$

where a is mx1 vector and e^T = (1,1,...,1) is an mx1 vector.

The initial basic feasible solution for this auxiliary problem can be easily obtained by setting x = 0 and a = b. Then, this basic feasible solution is used to solve the auxiliary problem using the simplex method.

The solution of phase 1 may indicate different things. 1) If min z > 0, it means the original LP problem is infeasible 2) If min z = 0 and all artificial variables are non-basic, the solution of phase 1 will serve as the basic feasible solution for phase 2. 3) If min z = 0 but at least one artificial variable is basic, pivoting needs to be applied to exchange the basic artificial variables with the non-basic variables of the original LP problem before proceeding to phase 2. The presence of artificial variables in the solution of phase 1 indicates the original system has redundancies or degenerate solutions.

Phase 2

In phase 2, the basic feasible solutions obtained when min $z = 0$ in phase 1 are used as the initial basic feasible solution to the original LP problem and then, the simplex method is applied to solve the original LP problem. The simplex method stops when an optimal solution is found or an unbounded solution is detected.

2.7.2.7. Special Cases in the Simplex Algorithm

Degeneracy

A basic feasible solution is said to be degenerate, if one or more of the basic variables are zero [16]. A new solution obtained by moving from a degenerate feasible solution will again be degenerate which will produce no improvement in the value of the objective function. If degeneracy is encountered in successive iterations, it may result in a return to a basis already obtained thus creating a cycling that the simplex method can’t come out of. Thus, in the presence of degeneracy, there is no guarantee that the simplex method will terminate in a finite number of steps.

Since degeneracy may result in cycling and since degeneracy is a common experience, one might expect cycling to also be encountered very frequently. But, in practice cycling is a very rare phenomenon occurring mainly in specially constructed test problems in researches.

How to deal with degeneracy

One way degeneracy can be handled in the simplex method is through the use of perturbation. One perturbation technique developed by A. Charnes makes sure the simplex method will get out of degeneracy and ensures it will have a finite number of iterations [16].

Since degeneracy in an LP problem occurs when the right hand side vector b can’t be expressed as a linear combination of the basis vectors formed from A, if we perturb b it might be possible to express the perturbed b as a linear combination of the basis vectors of A. Thus Charne's method works by replacing b_i in the simplex tableau by $b_i + \sum_{k=1}^{n} \bar{a}_{k_i}$

For example, if degeneracy occurs for the system

$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \dots + x_m\mathbf{a}_m = \mathbf{b} \tag{2.22}$$

then, we can perturb the right hand side of b to obtain the following system

$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \dots + x_m\mathbf{a}_m = b + \in^1 a_1 + \in^2 a_2 + \cdots + \in^n a_n = b(\in) \tag{2.23}$$

The advantage of the Charnes' method is that the solution of the perturbed system can be determined without knowing the actual value of ∈. And if the solution to the perturbed system is known, it will also be the solution to the original problem if we set ∈ to zero.

Whereas the solution of the original system in (22) is obtained as $x = B^{-1}b$, the solution for the perturbed system in (23) is obtained as follows

$x(\in) = + \in B^{-1}a_1 + \in^2 B^{-1}a_2 + \cdots . + \in^n B^{-1}a_n$, where $x = B^{-1}b$ is the solution of the unperturbed problem.

Each element of the vector $x(\in)$ is thus obtained as follows

$$x_i(\in) = x_i + \sum_{k=1}^{n} \in^j \bar{a}_{k_i} \quad \text{for i = 1,2,...,m} \tag{2.24}$$

The entering variable in the simplex method for the perturbed method is then found using the following minimum ratio test.

$$\min_i \left\{ \frac{x_i + \sum_{k=1}^{n} \in^j \bar{a}_{k_i}}{\bar{a}_{j_i}}, \ \bar{a}_{j_i} > 0 \right\} \tag{2.25}$$

The Charnes' method chooses the entering variable using equation (25) as follows. First, it compares the ratios $\frac{x_i}{\bar{a}_{j_i}}$ in equation (25). If a minimum ratio is obtained for unique i, there is no degeneracy. But, if a unique i can't be obtained for the minimum ratio of $\frac{x_i}{\bar{a}_{j_i}}$,we go on and compare the ratios $\frac{\bar{a}_{1_i}}{\bar{a}_{j_i}}$ for the rows which had a tie for the ratios of $\frac{x_i}{\bar{a}_{j_i}}$. Again, if we have rows tied for the ratios $\frac{x_i}{\bar{a}_{j_i}}$ and $\frac{\bar{a}_{1_i}}{\bar{a}_{j_i}}$, we go on to the next column and compare the ratios $\frac{\bar{a}_{2_i}}{\bar{a}_{j_i}}$. We go on comparing more ratios until we find a unique ratio. These procedures in the Charnes' method ensure a new basic feasible solution is obtained with all basic variables strictly positive.

Unboundedness

As the simplex method goes from one vertex to an adjacent vertex of the polyhedron of feasible solutions, along the edges of the polyhedron, it might reach an edge along which the objective function value can be reduced indefinitely without reaching to another vertex [16]. In such cases, the LP problem is said to be unbounded and the simplex method will stop without finding a solution.

2.7.3. Interior Point Algorithms

Unlike the simplex method which moves along the edges of the feasible region (polyhedron), interior point algorithms move through the interior of the feasible region. The first significant interior point algorithm was the Ellipsoid algorithm developed by Khachiyan in 1979 and it had a worst case running time complexity of (n^6L^2) , where n is the number of variables and L is the number of bits to encode the input[32]. But, the algorithm didn't have a performance to compete with the simplex method since for almost all inputs its running time complexity is close to its worst case running time complexity.

A better and efficient interior point algorithm was the projective algorithm developed by Kramarka in 1984[32]. This algorithm has a worst case complexity of $(O(n^{3.5}L^2)$ and was found to be efficient in practice.

2.7.4. Simplex vs Interior Point

A comparison of simplex methods with interior point methods shows that good implementations of simplex methods and interior point methods have similar performance although for some applications one may be better than the other [33]

2.8. Literature Review on GPUs

2.8.1. Introduction to Parallel Systems

Concurrent System vs Parallel System

A system is concurrent if it contains several operations that are ready for execution at the same time [25]. If there is only one processing element (processor) for executing the operations, only one operation will be executed at a time while all the other operations are forced to wait until the operation completes. But, if in a concurrent system we also have multiple processing elements (processors), several operations can execute simultaneously and we call such a system a parallel system.

Steps for writing programs to a parallel system

To write programs for a parallel system, first the programmer must identify the concurrency in the problem he/she wants to solve [25]. Then, this concurrency should be expressed in the software they write. Finally, their program is run so that the concurrent parts are run in parallel to give a good performance in terms of speed of execution.

2.8.2. Introduction to GPUs

GPU stands for Graphical processing unit [19]. GPUs are devices mainly used for image processing [18] and they were developed, in particular, for rendering graphics applications [19]. (Rendering is the transformations of vertices into pixels or the process of generating a 2D image from a model composed of thousands or millions of polygons)) GPUs can do graphics processing at significantly higher speed than CPU because they are developed to handle operations that are very common in graphics applications [19].

The first GPU was the GeForce 256 which was released in 1999[22]. Before GPUs, rendering was done on CPU [22]. But, doing rendering on a CPU is very computationally intensive. For instance, a given image may be modelled by millions of triangles where, in turn, each triangle may hold hundreds of pixels. Thus, generating an image from the vertices of the triangles is a very computationally intensive task to be done by the CPU. Due to this, a dedicated Graphical Processing Unit which handles rendering was developed to free the CPU to do other useful computational tasks. At present the main manufacturers of GPUs are Nvidia and AMD/ATI[23]. Nvidia are famous for their GeForce series where as AMD/ATI have their Radeon series.

GPUs were primarily developed to meet the demands of the gaming industry [23] and they were initially single purpose rendering devices [24]. But, now, they are programmable processors [24] that execute programs, including general purpose programs which have nothing to do with graphics, mainly in a SIMD (Single Instruction Multiple Data) way. (SIMD is a programming model where a single instruction is executed in parallel across multiple processing elements with each processing element working on its own data [24]). GPUs are also a cheap parallel architecture to implement data parallel applications as a reasonably powerful GPU costs only a few hundred dollars [18].

2.8.3. GPU vs CPU

One way in which GPUs differ from CPUs is in the number of cores they have. GPUs can be equipped with up to thousands of cores [18] while most CPUs have few cores (most commonly from 4 to 16 cores). GPUs have smaller but large number of processing cores optimized for parallel computing while CPUs have one or few large processing cores optimized for serial computing. The main reason GPUs are equipped with such large number of cores and parallel threads is because of the need for rendering complicated and high resolution 3D scenes at real-time to create interactive frame rates for games [22].

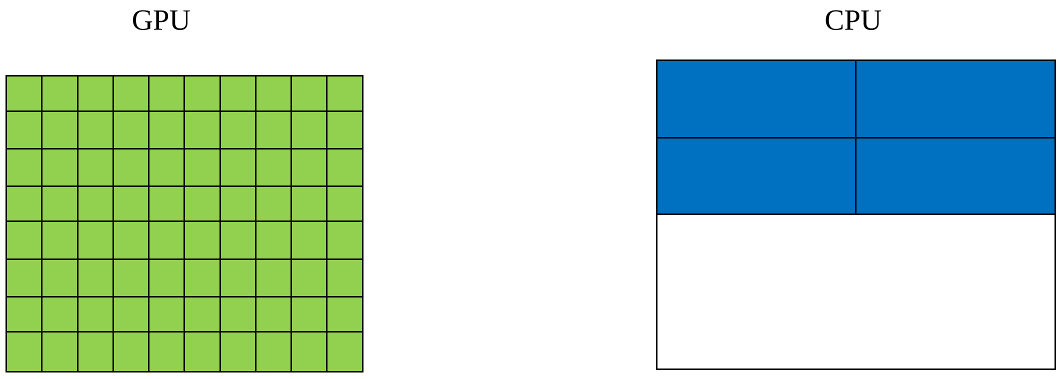


Figure 2.15. GPU vs CPU

Another way in which GPUs differ from CPUs is in the way they handle the latency while accessing memories. While CPUs use caches to hide the latency in accessing memory, GPUs primarily hide this latency by running large number of threads at the same time [23] i.e. while one thread is accessing the memory, there are several other threads still executing on the GPU cores effectively hiding the latency of that one thread.

2.8.4. GPGPU

GPGPU stands for General Purpose Graphics Processing Unit [18]. With the GPGPU paradigm GPU’s began to be used for general purpose scientific computation in addition to their primary use as rendering devices [18]. GPGPU is now becoming a popular choice for developing general purpose parallel applications because GPUs are powerful devices, yet not expensive. GPUs are in fact one of the most powerful computation hardware for their price [20]. Furthermore, GPUs are drawing attention because their performance is growing fast even above Moore’s law i.e. the performance of GPU devices is doubling in less than 18 months. This is because many of the GPU transistors are used for computation rather than non-computation tasks like branch prediction [20].

The following are some of the many areas GPUs have been used for general purpose computation [20].

- Physics simulations e.g. for boiling simulation, cloth simulation and fluid dynamics simulations

- Signal and image processing applications e.g. for segmentation, real-time stereo depth extraction and in computed tomography (CT) to reconstruct an object from its projections.
- Geometric computations e.g. for calculating distance fields which are used for path planning.
- Databases and data mining e.g. for accelerating the performance of database queries.

But not all applications are inherently parallel and able to be parallelized by GPUs. For instance, applications dominated by memory communication, instead of computation, like word processing applications are tough to parallelize in GPUs [20].

Evolution of GPGPU

Before programming GPUs for general purpose applications (GPGPU) was started GPUs, which were fixed function rendering devices, began to be programmed for graphics [24].The graphics computations were expressed using graphical terms such as vertices, textures, and fragments and blending [21].

Then, in the early days of programming a GPU for general purpose programs (GPGPU), general purpose programs were programmed using graphics API .i.e. general purpose computations were expressed in graphics terms such as vertices, textures, fragments and blending. Examples of these early programming languages for general purpose GPU computing include HLSL(High Level Shader Language) and GLSL(OpenGL shading Language).

At present, high-level languages which use general terms (not graphical terms) are used for GPGPU computing. In this new approach, computations are specified as a set of thread which can execute in parallel. Then, a SPMD (Single Program Multiple Data) program is executed on each thread. The computation result of each thread is stored in a buffer (global memory). Then, finally, the value of the buffer is read and can optionally be used for additional computation.

The present approach allows developers to have access to the processing elements of a GPU without being forced to use a graphical interface (a graphics API) when developing general purpose programs on GPUs i.e. it is currently possible to have full access to the powerful GPU hardware using familiar high-level programming languages using the derivatives of the C-syntax (the C programming language). At present, the most common programing language for GPGPU computing are Nvidia's CUDA and OpenCL.

Frameworks for GPGPU programming

Computation toolkits (frameworks) for GPGPU were at first high-level shading languages like Cg and HLSL. But, at present, they are modern programming languages based on C like Cuda and OpenCL[26]. A brief list of some notable languages for writing programs on GPU is shown below [19].

- Cg[29](C for Graphis) : Cg was used for writing shader programs for OpenGL and Direct X.
- Accelerator [30]: Accelerator is a .Net assembly allowing access to the GPU through the Direct X interface.
- HLSL[31](High Level Shader Language) is an application developed by Microsoft for developing shader programs for Direct X with a capabilitiy to run both on Windows and Xbox platforms.
- Cuda(Compute Unified Device Architecture) : Cuda is a programming language for writing programs to Nvidia graphics devices[26].
- OpenCL(Open Computing Language) : OpenCL is a standard for writing parallel applications that can run on heterogeneous platforms with different devices from different vendors[25].

2.8.5. OpenCL vs CUDA

At present, the two most common programing languages for writing GPGPU programs are CUDA and OpenCL. CUDA is a framework for developing general purpose parallel programs on Nvidia GPUs. CUDA was introduced in 2006 and with the advent of CUDA, the use of graphics APIs for writing general purpose programs on GPUs was eliminated [26].

OpenCL is a framework first released in 2008 to give programmers a portable and efficient access to powerful processing elements(GPUs,CPUs,DSPs etc) . The APIs of OpenCL are not vendor specific and one can develop a single program that can run on a wide range of devises from different vendors. OpenCL and CUDA share many core ideas and even terminologies and it is fairly easy to translate CUDA codes to OpenCL and vice versa.

The portability of OpenCL is found not to affect its performance by a recent research[26]. The research was done using 16 bench marks composed of synthetic applications and real-world applications and in a fair-comparision, OpenCL programs were found to perform as fast as the corresponding CUDA programs.

2.8.6. OpenCL

OpenCL was first released in December 2008[25].It is a framework for writing programs which execute on parallel processing platforms [24]. It is not specifically used for programming GPUs only but it is also used across a range of devices including CPUs, DSPs and FPGAs [18]. OpenCL allows developers to have portable and efficient access to the capabilities of various processors from different vendors in a heterogeneous environment [24]. I.e. it allows writing a single program which can run on different types of systems from cellphones to super computers [25]. OpenCL programs are written using a subset of ISO C99 with some extensions and limitations [24] and they can be used for both data parallel and task parallel programming models.

2.8.6.1. Some OpenCL Terminologies

Platform: A platform refers to a host device and a collection of other devices on which an application can execute kernels.

Device: A device is composed of one or more compute units.

Compute Unit: A compute units is made up of one or more processing elements and one work group executes on one compute unit.

Processing element: Processing elements are the components that the compute unit is made of in addition to local memory

Kernel: A kernel is a function that will be executed in an OpenCL device. It is identified from other functions by its __kernel qualifier.

Work Item: A work item is one of the many parallel executions a kernel issues on a device by a command i.e. it is a kernel instance.

Command: A command is the OpenCL operations (e.g. executing kernels, reading and writing memory objects) that are placed into a command queue for execution.

Command queue: A command queue is used for queuing commands to a device.

Workgroup: A workgroup is a group of work items that execute on the same compute unit.

Context: A context is composed of a group of devices, the memory accessible to the devices , the properties of the memories(read only, write only etc) and one or more command queues. It is the environment in which kernels execute.

Host: A host is a device that interacts with the context using OpenCL API and it coordinates the executions of kernels on devices.

Program: An OpenCL program is made up of a set of kernels, other functions called by the kernels and a constant data.

Kernel Object: A kernel object is used to encapsulate a specific __kernel function and its argument values.

Global ID: A global ID can uniquely identify a work item. It is unique to the whole index space.

Local ID: A local ID is a work-item id unique with in a workgroup.

Global memory: A global memory is a region of memory accessible to all work-items executing in a context.

Constant memory: Constant memory is the same as global memory but it remains constant (not updated) while the kernel is being executed.

Local memory: Local memory is a region of memory accessible by all work-items in the same-workgroup but not by work items from other workgroups.

Private memory: Private memory is a region of memory private to a work item.

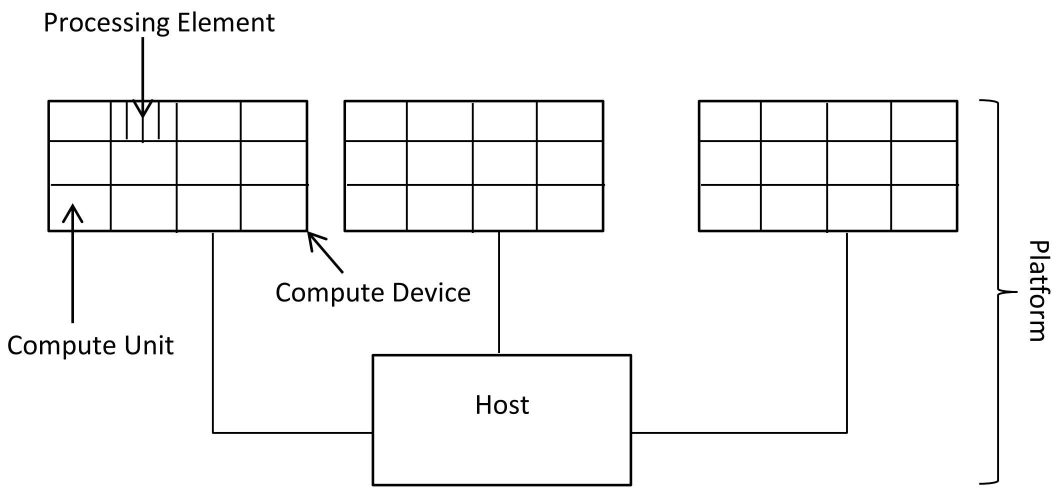


Figure 2.16. OpenCL Platform Model

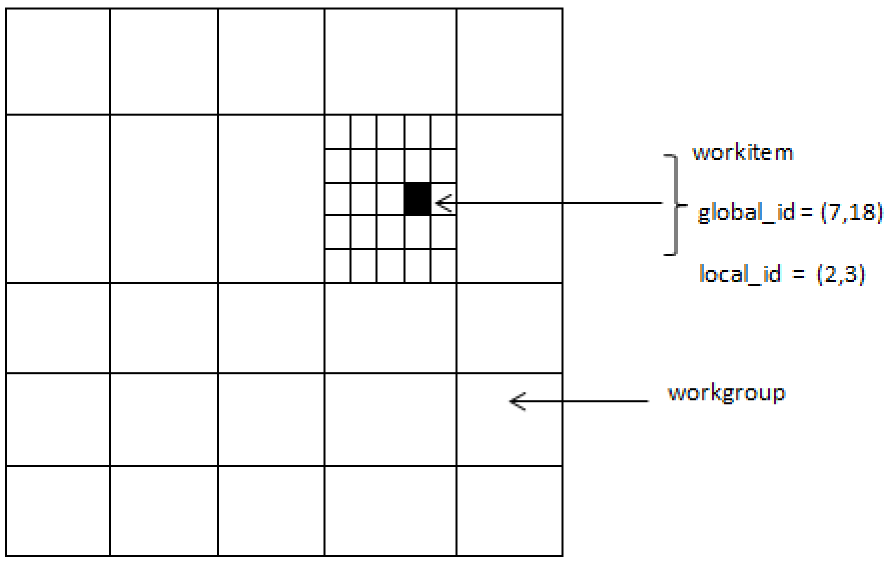


Figure 2.17. OpenCL Execution Model

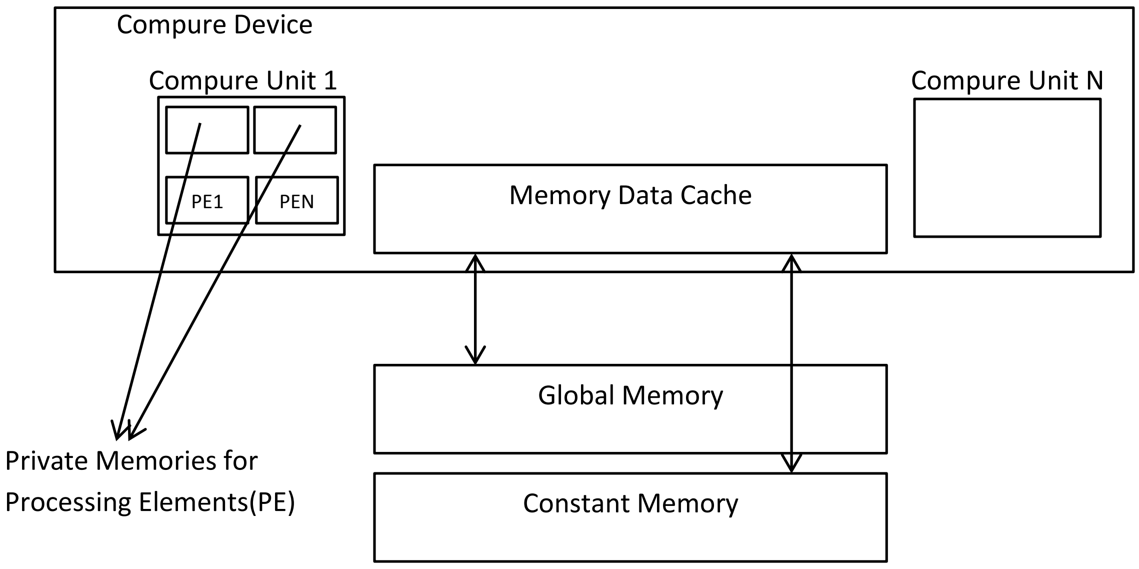


Figure 2.18. OpenCL Memory Model

2.8.6.2. Steps for Writing an Application in OpenCL for a Heterogeneous System

To write a program in OpenCL that is capable of running on a heterogeneous system, first we have to discover what components the heterogeneous system is composed of [25]. Then, we have to get information about the properties of the components so that our software can make best use of the components. Then, we have to create blocks of instructions (kernels) which will run on the platform. Then, we will setup memory objects (buffers) required to do the computation. After this, we execute the kernels in the appropriate components. Finally, we can read the final result.

2.8.7. CPU+GPU Co-processing

Writing applications that execute only on a CPU or a GPU may not be efficient because most applications have serial part suitable for execution on a CPU and a parallel part that benefits from execution on GPUs [22]. The CPU+GPU co-processing approach is developed because CPU and GPU have complimentary attributes (i.e. CPUs are good for serial programs and GPUs are suitable for parallel programs). Most programs that use CPU+GPU co processing have been found to have better performance per power consumed or die area than programs that run on CPU or GPU cores alone.

2.9. Literature review on parallelization of optimization problems on GPU

In [34], the authors implemented a branch and bound algorithm for a class of integer programming problems called Knapsack problems using GPU. The hardware they used in their experiment was a GTX 260 GPU and 3GHZ Xeon Quadro Intel Processor.

Knapsack problems try to maximize the profit of n-items that fill a knapsack where each item has a weight of w_i where $i \in \{1,..n\}$. Knapsack problems can be modeled as Integer Programming problem and can be solved by branch and bound. If the size of sub-problems in the branch and bound tree is small, the authors chose the computation to be done on CPU. But, for large number of sub-problems, a list of sub-problems was selected using breadth first search strategy and transferred to GPU. Branching on the sub-problems was done in the GPU and using the result obtained, bound computation was again done on GPU. And the list of created sub-problems whose bound was computed was returned to CPU for pruning. For a problem size of 500, they obtained a speed up of 9.27

Our approach differs from theirs in that we don’t make two separate kernel calls from the CPU once to do the branching and another to do the bounding. Instead with one call to a kernel, we branch on nodes passed to the GPU and the bound of the resulting nodes(sub-problems) will also be computed in the GPU and then the resulting nodes whose bounds is computed will be returned back to the CPU. The second difference is that while we are solving an LP problem during bound computation they are solving a different formula to obtain the bound. And the third main difference of our approach from them is that while finding the best lower bound, we also perform partial (forward) pruning. And when we do full pruning, the number of nodes we prune will be smaller since some nodes were already pruned while doing partial pruning when the best lower bound is computed. Their approach instead finds the lower bound and after that it will prune nodes and the size of the nodes in their pruning stage will therefore be larger than the size of the nodes in our approach.

In [35], Chakroun and Melab tried to parallelize the branch and bound algorithm which they used for solving the Flow Shop Scheduling Problem (FSP) by performing parallel computation of bounds. (FSP is related to scheduling of n jobs in m machines.) They used Nvidia Tesla T10 GPUs each with 240 CUDA cores and 4GB global. They took a pool of nodes from the branch and bound tree using depth first search strategy and they performed the bound computations for these pool inside GPUs and then they performed elimination and branching on the CPU. They obtained an acceleration of x78 for a 200x20 problem instance using a single GPU and an acceleration of x105 for the same problem instance using two GPUs.

They also dynamically tuned the size of sub-problems fed into the GPU depending on the problem because the performance of GPU acceleration depended on the input problem. In the dynamic tuning, they solved the problems with progressively larger sizes of sub-problems until the maximum number of sub problems that the GPU can handle at a time was reached. And, for each sub-problem size that was used, the speed up per sub-problem was computed and the size that resulted in greatest speedup was selected. And it was verified that using such dynamic tuning heuristic, instead of using fixed size pool of sub problems, produces a better speedup.

Our approach differs from theirs in that we do branching on GPU in addition to bound computation. And our approach uses best first search strategy (BFS) to choose group of unexplored nodes from the branch and bound tree while their approach uses depth first search (DFS).

In [36], the authors suggest parallelizing of the standard simplex method to solve non-sparse LP problems. Although the revised simplex method generally outperforms the standard simplex method, they chose the standard simplex method since for non-sparse matrices both the standard and revised simplex methods give the same performance. Since most of the standard simplex method procedures are spent on pivoting, they suggested doing the pivoting on GPU. In their approach, the simplex tableau is transferred to a GPU and the minimum ratio test and pivoting are done in GPU where as finding the index of the entering and leaving variable are done on CPU.

They used a GTX 260 board and for large simplex tableau instances i.e. 7000x7000, they obtained a speed up of 12.5 over the corresponding CPU implementation, where as for small size problems i.e. 500x500, they obtained a speed up of 2.66.

Our approach differs from them in that we parallelize IP problems instead of LP problems i.e. instead of parallelizing a single simplex instance; we are parallelizing computations of several simplex instances to perform bound computations in parallel. And instead of using simplex during bound computation, we use the revised simplex method.

In [37], the authors presented a GPU implementation of the revised simplex method to parallelize solving of linear programming problems. They used an nVidida GeForce 9600 GT GPU with 1.0 GB RAM and with 64 shader processors. They used as a bench mark randomly generated linear programming problems in canonical form. They compared their implementation with an open source Linear Programming solver called GLPK and for large problem instances they obtained up to 18x speedup over the corresponding CPU implementation by GLPK.

Our approach differs from theirs' in that instead of using graphics library to write programs for GPU we use OpenCL which is a general purpose programming language and we parallelize IP problems instead of LP problems. And another difference is that they use the steepest edge method to select entering variables while we use an approach where non-basic variables corresponding to the largest decrease in the objective function value are chosen as entering variable.

CHAPTER 3

METHODOLOGY

3.1. The Proposed IP based Itemset Hiding (Sanitization) Algorithm

3.1.1. Inputs to the Proposed Itemset Hiding (Sanitization) Algorithm

Before using our IP based itemset hiding algorithm to sanitize datasets, we have to make sure that the input datasets are represented in binary format. If not, we have to first convert them into binary format. For example, the sample dataset in Table 3.1 which is not in binary format (it is expressed in market basket format) has to be converted into the binary format shown in Table 3.2.

Table 3.1. A sample dataset in market-basket format

TID (Transaction ID)	Items making the transaction
1	1 2
2	3
3	4
4	1 2 3
5	1 2 3 4

Table 3.2. A sample dataset in binary format

d_{ij}	Item 1	Item 2	Item 3	Item 4
T1	1	1	0	0
T2	0	0	1	0
T3	0	0	0	1
T4	1	1	1	0
T4	1	1	1	1

In general a dataset that is to going to be sanitized by our sanitization algorithm has to be first expressed in the following binary format.

Table 3.3. Format of an input dataset

d_{ij}	i_1	i_2	....	i_m
T_1				
T_2				
. .				
T_n				

where the values in the cells of the above table can have only a binary value of 0 or 1 corresponding to the absence or the presence of an item in a transaction.

The goal of the sanitization procedure is to make sure sensitive itemsets cannot be mined from the sanitized database. Thus, after mining the input dataset and obtaining the non-singleton frequent itemsets, the frequent itemsets are split into sensitive and non-sensitive itemsets depending on which itemsets are sensitive so that we don't want them to be shown when mining the sanitized dataset.

The sensitive and non-sensitive itemsets are the other inputs to our sanitization algorithm. These itemsets, like the input dataset, also have to be expressed in binary format before being fed into the sanitization algorithm. In general, the sensitive and non-sensitive itemsets shall be expressed in the binary formats shown in Table 3.4 and 3.5 when being used in the sanitization algorithm. The values in the cells of thebtables can have only a binary value of 0 or 1 corresponding to the absence or the presence of an item in an itemset.

Table 3.4. Sensitive Itemsets

s_{kj}	i_1	i_2	….	i_m
I_1				
I_2				
. .				
I_o				

Table 3.5. Non-Sensitive Itemsets

n_{rj}	i_1	i_2	….	i_m
I_1				
I_2				
. .				
p				

3.1.2. Terminologies Used To Express the Proposed Itemset Hiding Algorithm

n :	The total number of transactions in the dataset
m :	The total number of unique items in the dataset

ψ : The minimum support threshold i.e. itemsets having support above the minimum support threshold are called frequent where as itemsets whose support is below the minimum support threshold are called infrequent.

o :	The total number of non- singleton sensitive itemsets
p :	The total number of non-singleton non-sensitive itemsets
$\delta(I_k)$:	The support of the kth sensitive itemset
$\delta(I_r)$:	The support of the rth non-sensitive itemset

i : A transaction number ranging from 1 to n

j : An item number ranging from 1 to m

k : A sensitive itemset number ranging from 1 to o

r : A non-sensitive itemset number ranging from 1 to p

d_{ij} : A binary variable used to indicate whether the jth item is present in the ith transaction or not where $i \in \{1,...n\}$ and $j \in \{1,...m\}$. If d_{ij} is 1, it means the jth item is present in the ith transaction where as if it is 0, it indicates the jth item is not found in the ith transaction. A matrix of d_{ij} values are used to represent the input dataset in binary format.

s_{kj}: A binary variable used to indicate whether the jth item is present in the kth sensitive itemset or not where $k \in \{1,...o\}$ and $j \in \{1,...m\}$. If s_{kj} is 1, it means the jth item is present in the kth sensitive itemset where as if it is 0, it indicates the jth item is not found in the kth sensitive itemset. A matrix of s_{kj} values are used to represent all the sensitive itemsets.

n_{rj}: A binary variable used to indicate whether the jth item is present in the rth non-sensitive itemset or not where $r \in \{1,...p\}$ and $j \in \{1,...m\}$.If n_{rj} is 1, it means the jth item is present in the rth non-sensitive itemset where as if it is 0, it indicates the jth item is not found in the rth non-sensitive itemset. A matrix of n_{rj} values are used to represent all the non-sensitive itemsets.

x_{ij} : A binary decision variable used to determine which items of the input dataset will be removed during sanitization where $i \in \{1,...n\}$ and $j \in \{1,...m\}$. If x_{ij} is set to 1, d_{ij} will be set to 0 when sanitizing the database. But, if x_{ij} is set to 0, the original value of d_{ij} will not change when sanitizing the database.

z_{ir} : A binary decision variable used to determine whether or not the rth non-sensitive itemset will be removed from transaction i during sanitization where $i \in \{1,...n\}$ and $r \in \{1,...p\}$. If z_{ir} is set to 1, it means the rth non-sensitive will be removed from the ith transaction during sanitization. But, if z_{ir} is set to 0, it means the rth non-sensitive itemset will not be removed from transaction i. (But note that a non-sensitive itemset cannot be removed from a transaction which didn't contain the non-sensitive itemset in the first place. So, z_{ir} can be set to 1 in only those transactions which contained the rth non-sensitive itemset. i.e. sup[i,r] must be 1 for z_{ir} to attain a value of 1.)

R_r: The amount by which the support of the rth non-sensitive itemset will be reduced below the minimum support threshold as a side effect of the sanitization process. Note that the maximum amount the support of the rth nonsensitive itemset can be reduced below the minimum support threshold(ψ) is ψ, by which point the support of the rth non sensitive itemset will be 0.

U_r: A binary decision variable used to determine whether or not the rth non-sensitive itemset will be removed from the dataset (i.e. whether or not its support will be reduced below the minimum support threshold ψ) during sanitization. If U_r is set to 1, it means the r_{th} non-sensitive will have its support reduced below the minimum support threshold (i.e. removed from the database) during sanitization. But, if U_r is 0, it means the r_{th} non-sensitive itemset will remain frequent after sanitization.

3.1.3. Objective of the Proposed Itemset Hiding Algorithm

Objective

The objectives of the proposed sanitization algorithm when sanitizing the database are (i.e. minimizing the number of non-sensitive itemsets whose support will be reduced below the minimum support threshold as a side effect of removing sensitive itemsets)

3.1.4 Constraints on the Proposed Itemset Hiding(Sanitization) Algorithm

Constraint 1

To remove the kth sensitive itemset from the dataset, the sensitive itemset must be removed from at least $\delta(I_k)-\psi+1$number of transactions. And to remove a sensitive itemset from a particular transaction at least one item of the itemset must be removed from the transaction. Thus, to remove the kth sensitive itemset from the database by removing it from $\delta(I_k)-\psi+1$transactions at least a total of $\delta(I_k)-\psi+1$ items of the sensitive items must be removed from the dataset. (But note that we have to make sure that we are attempting to remove sensitive itemsets from transactions which contained the sensitive itemsets in the first place (i.e. sup[i, k] = 1))

Constraint 2

To remove the kth sensitive itemset from transaction i, it is sufficient to remove only one item of the kth sensitive itemset from transaction i. Removing just one item of the sensitive itemset to remove the sensitive itemset from a particular transaction ensures minimum disturbance to the original database during sanitization. Thus, we can impose the constraint that while sanitizing the database remove only one item of a sensitive itemset to remove the sensitive itemset from a particular transaction.(But note that we have to make sure that we are attempting to remove sensitive itemsets from transactions which contained the sensitive itemsets in the first place (i.e. sup[i, k] = 1))

Constraint 3

In ideal sanitization process which doesn't have any side-effect on the non-sensitive itemsets, the support of the rth non-sensitive itemset can be decreased by upto $\delta(I_r) - \psi$ without making the non-sensitive itemset infrequent in the sanitized dataset. But, practically, it is usually impossible to sanitize a dataset without removing some non-sensitive itemsets as a side-effect. Thus the support of the rth non-sensitive itemset may be reduced by R_r amount below the minimum support threshold during sanitization thus making the non-sensitive itemset infrequent for non-zero values of R_r. Thus, we can impose a constraint on the sanitization process so that the number of transactions from which the rth non-sensitive itemset is removed must not exceed $\delta(I_r) - \psi + R_r$ where R_r is a quantity coming from the side-effect of removing non-sensitive itemsets in the attempt to remove sensitive itemsets from the dataset.

Constraint 4

If the rth non-sensitive itemset will not be removed from transaction i during sanitization, then no item of the non-sensitive itemset should be removed from the transaction. But, if the rth non-sensitive itemset will be removed from transaction i during sanitization, any number of items of the non-sensitive itemset can be removed from transaction i. Since the rth non-sensitive itemset can have upto m items, upto m items of the non-sensitive itemset can be removed from transaction i, if the non-sensitive itemset is to be removed from transaction i during sanitization.

Constraint 5

If the rth non-sensitive itemset will not be removed from the dataset during sanitization (i.e. $U_r = 0$), then its support cannot be reduced below the minimum support threshold (ψ) by any non-zero amount R_r i.e. R_r must be zero. But, if the rth sensitive itemset will be removed from the dataset during sanitization (i.e. $U_r = 1$), the support of the rth non-sensitive itemset will be reduced by an amount R_r not exceeding ψ.

Constraint 6

x_{ij} is a binary variable i.e. it can have a value of 0 and 1 only.

Constraint 7

z_{ir} is a binary variable. i.e. it can have a value of 0 and 1 only.

Constraint 8

U_r is a binary variable. i.e. it can have a value of 0 and 1 only.

3.1.5. The IP Problem for Sanitizing Datasets with Minimum Side-Effect

The above constraints on the sanitization procedure and the objective of the sanitization process can be expressed into the following Integer Programming (IP) problem.

$$\text{Min } C_1\sum_{i=1}^{n}\sum_{j=1}^{m} d_{ij}x_{ij} + C_2\sum_{i=1}^{n}\sum_{r=1}^{p} z_{ir} + C_3\sum_{r=1}^{p} U_r \tag{3.1}$$

$$\text{s.t } \sum_{i=1}^{n}\sum_{j=1}^{m}(d_{ij} * s_{kj} * x_{ij}) \geq (\delta(I_k) - \psi + 1)$$

$$, \forall k \;:\; I_k \in F^s(D,\psi) \text{ and } \sup[i,k] = 1 \tag{3.2}$$

$$\text{s.t. } \sum_{j=1}^{m}(d_{ij} * s_{kj} * x_{ij}) \leq 1$$

$$, \forall (i,k) \ : \ I_k \in F^s(D,\psi), \ i \in \{1,...n\} \text{ and } \sup[i,k] = 1 \quad (3.3)$$

s.t. $\sum_{i=1}^{n} z_{ir} \leq (\delta(I_r) - \psi + R_r)$

$, \ \forall r \ : \ I_r \in F^n(D,\psi) \text{ and } \sup[i,r] = 1$ (3.4)

s.t. $\sum_{j=1}^{m} (d_{ij} * n_{rj} * x_{ij}) \leq (m * z_{ir})$

$$, \quad \forall (i,r) \ : \ I_r \in F^N(D,\psi), \ i \in \{1,...n\} \text{ and } \sup[i,r] = 1 \quad (3.5)$$

s.t. $R_r \leq \psi U_r \quad \forall r \ : \ I_r \in F^n(D,\psi)$ (3.6)

s.t. $x_{ij} \leq 1 \ , \quad \forall (i,j) \ : \ i \in \{1,...n\} \text{ and } j \in \{1,...m\}$ (3.7)

s.t. $z_{ir} \leq 1 \ , \quad \forall (i,r) \ : \ I_r \in F^N(D,\psi) \text{ and } i \in \{1,...n\}$ (3.8)

s.t. $U_r \leq 1 \ , \ \forall r : \ I_r \in F^n(D,\psi)$ (3.9)

3.1.6 .The Standard Form of the Proposed IP Problem

The above IP problem can be converted into the following standard form by the addition of slack and surplus variables.

Min $C_1 \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij} x_{ij} + \ C_2 \sum_{i=1}^{n} \sum_{r=1}^{p} z_{ir} + \ C_3 \sum_{r=1}^{p} U_r$ (3.10)

s.t $(\sum_{i=1}^{n} \sum_{j=1}^{m} (d_{ij} * s_{kj} * x_{ij})) + s1_k = (\delta(I_k) - \psi + 1),$

$$\forall k \ : \ I_k \in F^s(D,\psi) \text{ and } \sup[i,k] = 1 \ (3.11)$$

s.t. $(\sum_{j=1}^{m} (d_{ij} * s_{kj} * x_{ij})) + s2_{ik} = 1 \ ,$

$$\forall(i,k): I_k \in F^s(D,\psi), i \in \{1,...n\} \text{ and } \sup[i,k]=1 \quad (3.12)$$

$$\text{s.t. } (\sum_{i=1}^{n} z_{ir}) - R_r + s3_r = (\delta(I_r)-\psi),$$

$$\forall r : I_r \in F^n(D,\psi) \text{ and } \sup[i,r]=1 \quad (3.13)$$

$$\text{s.t. } (\sum_{j=1}^{m}(d_{ij} * n_{rj} * x_{ij})) - (m * z_{ir}) + s4_{ir} = 0,$$

$$\forall(i,r) : I_r \in F^N(D,\psi),\ i \in \{1,...n\} \text{ and } \sup[i,r]=1 \quad (3.14)$$

$$\text{s.t. } R_r - (\psi U_r) + s5_r = 0,\ \forall r : I_r \in F^n(D,\psi) \quad (3.15)$$

$$\text{s.t. } x_{ij} + s6_{ij} = 1\ .\ \forall(i,j) : i \in \{1,...n\} \text{ and } j \in \{1,...m\} \quad (3.16)$$

$$\text{s.t. } z_{ir} + s7_{ir} = 1\ .\ \forall(i,r) : I_r \in F^N(D,\psi) \text{ and } i \in \{1,...n\} \quad (3.17)$$

$$\text{s.t. } U_r + s8_r = 1\ .\ \forall r : I_r \in F^n(D,\psi) \quad (3.18)$$

3.2. Steps for Sanitizing an Input Dataset Using the Proposed Itemset Hiding (Sanitization) Algorithm

The steps for sanitizing an input dataset using the proposed IP based itemset hiding (sanitization) algorithm are shown below.

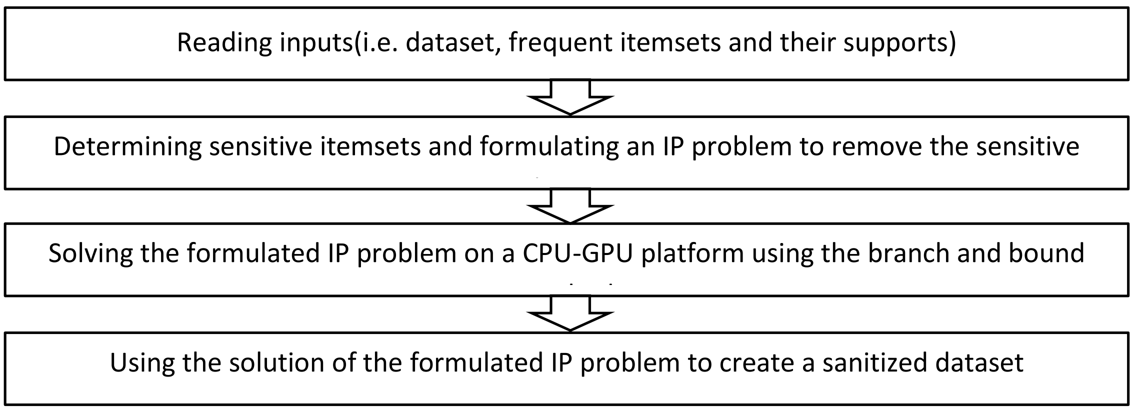

Reading inputs(i.e. dataset, frequent itemsets and their supports)
Determining sensitive itemsets and formulating an IP problem to remove the sensitive
Solving the formulated IP problem on a CPU-GPU platform using the branch and bound
Using the solution of the formulated IP problem to create a sanitized dataset

Figure 3.1. The procedures for sanitizing an input dataset

The branch and bound algorithm was used to solve an IP problem. And within the branch and bound algorithm, the algorithm used for solving linear programming problems in order to obtain bounds was the revised simplex algorithm.

3.3. Generation of the Inputs to our Itemset Hiding Algorithm

In order to obtain the frequent itemsets and their supports, the Apriori algorithm implemented in Java by the authors shown below was used with slight modification to format its outputs. After an input data set is given to the code and the minimum support threshold is set, this code outputs the non-singleton frequent itemsets, their supports, the number of transactions and unique items in the dataset as well as the number of non-singeleton frequent itemsets which are used as inputs in our sanitization algorithm.

```
 * @author Martin Monperrus, University of Darmstadt, 2010
 * @author Nathan Magnus and Su Yibin, under the supervision of Howard Hamilton,
 *         University of Regina, June 2009.
 * @copyright GNU General Public License v3
```

3.4. The Proposed Architecture for Solving the Formulated IP Problem

The proposed architecture for solving the formulated IP problem on a CPU-GPU platform is shown in Figure 3.2.

3.5. Summary of tasks done on CPU

The tasks done on a CPU are

- Initialization of the GPU
- Reading the inputs(i.e. dataset, frequent item sets and their supports) and formulation of an IP problem which is used to sanitize the input dataset
- Solving the formulated IP problem with the branch and bound method or delegating some part of the branch and bound steps to be done on GPU if the search area in the branch and bound tree gets large.
- Using the solution of the formulated IP problem to sanitize the input dataset.

But note that in order to solve IP problems with the branch and bound method the CPU uses a function called lpsolve which uses the two phase method(refer to section 2.7.2.7) to solve LP problems that result from relaxing the integrality constraints on the IP problems, which will then be used to obtain the lower bounds on the objective function value of the sub-problems(nodes).

lpsolve is used to obtain a solution of LP problems without being given an initial basic feasible solution. But lpsolve depends on a function called revisedsimplexlu which is used to solve LP problems only after being given an initial basic solution.

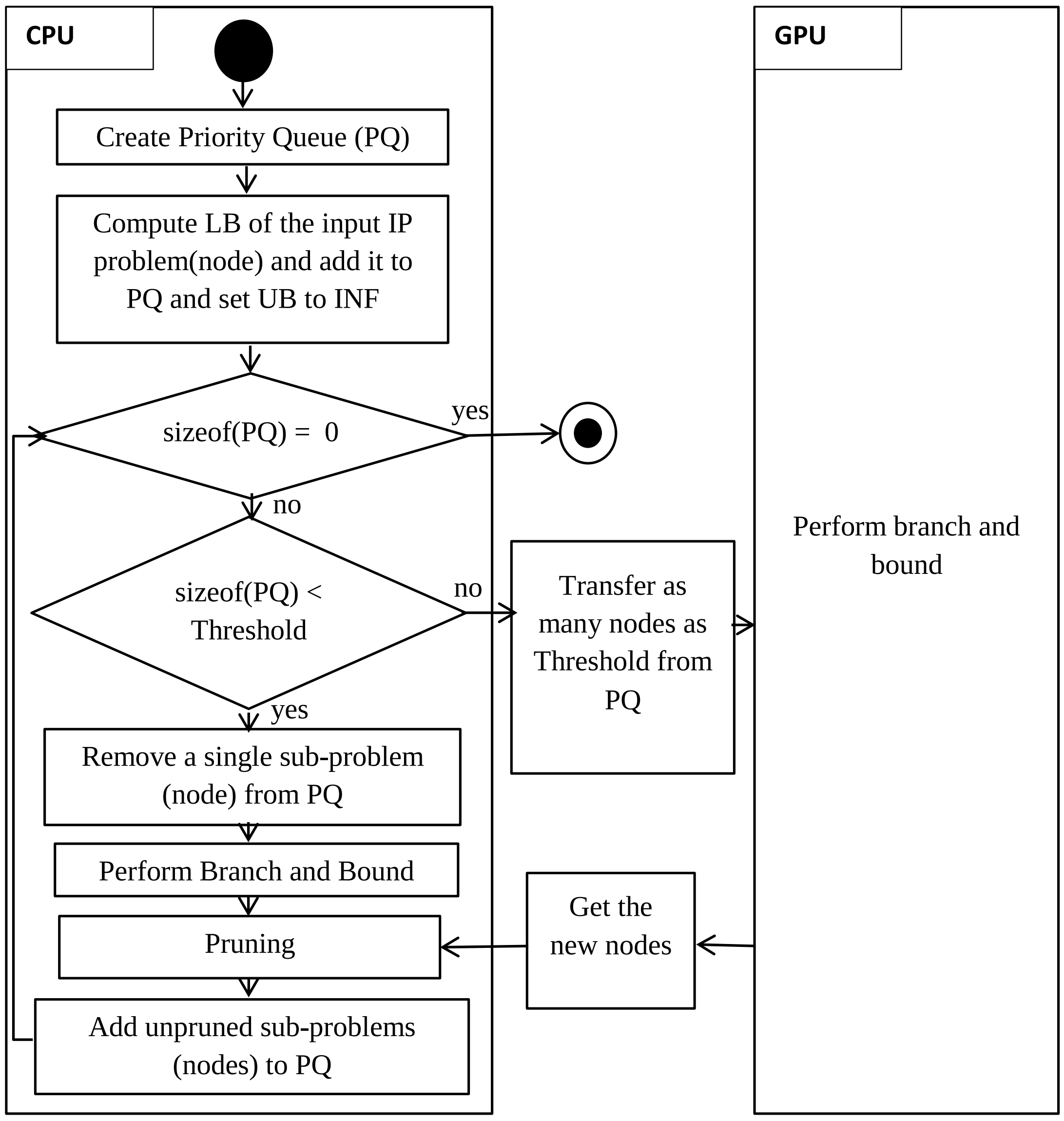

CPU
GPU
Create Priority Queue (PQ)
Compute LB of the input IP problem(node) and add it to PQ and set UB to INF
sizeof(PQ) = 0
yes
no
sizeof(PQ) < Threshold
no
yes
Transfer as many nodes as Threshold from PQ
Perform branch and bound
Remove a single sub-problem (node) from PQ
Perform Branch and Bound
Pruning
Get the new nodes
Add unpruned sub-problems (nodes) to PQ

Figure 3.2. A CPU-GPU platform for parallelizing the branch and bound method

The revisedsimplexlu function needs to solve system of linear equations and for this purpose it uses a function call lusolve which solves linear system of equations using lu decomposition. So, the function lusolve in turn uses a function called lu to perform LU decomposition of square matrices.

The inputs to the program are specified within a function called input_selector where the paths of the input and output files as well as charachterstics of the input dataset and frequent itemsets are specified. So, the parameters within this function must be set before running the program.

More description of the components of the CPU code is shown in the tables below.

Table 3.6. sparse_matrix structure

sparse_matrix : A c structure used to store a sparse matrix in row major form			
Members	**Name**	**Data type**	**Description**
	values	float **	Non zero values in the original matrix
	col_indices	int **	Column indices of the non zero values in the original matrix
	size_per_row	Int *	The number of non-zero elments per row in the original matrix

Table 3.7. sparse-matrix2 structure

sparse_matrix2 : A c structure used to store a sparse matrix in column major form			
Members	**Name**	**Data type**	**Description**
	values	float **	Non zero values in the original matrix
	row_indices	int **	Row indices of the non zero values in the original matrix
	size_per_column	int *	The number of non-zero elments per column in the original matrix

The following example demonstrates the above row-major and column-major sparse matrix representations.

Assume we are given the following matrix

$$\begin{matrix} 1 & 0 \\ 2 & 3 \end{matrix}$$

Then its row-major sparse representation using sparse_matrix looks as follows.

Values col_indices size_per_row

$$\left|\begin{matrix} 1 & \square \\ 2 & 3 \end{matrix}\right. \qquad \left|\begin{matrix} 0 & \square \\ 0 & 1 \end{matrix}\right. \qquad \left|\begin{matrix} 1 \\ 2 \end{matrix}\right.$$

And its column-major sparse representation using sparse_matrix2 looks as follows

$$\underset{\begin{matrix} 1 & 3 \\ 2 & \square \end{matrix}}{\text{Values}} \qquad \underset{\begin{matrix} 0 & 0 \\ 1 & \square \end{matrix}}{\text{row\_indices}} \qquad \underset{\begin{matrix} 2 & 1 \end{matrix}}{\text{size\_per\_column}}$$

Table 3.8. cl_node structure

cl_node : a c structure used to store a sub-problem(node) corresponding to an IP or LP problem		
Name	Data type	Description
A	sparse_matrix2	A matrix in the constraints Ax=b
b	float *	A vector in the constraints Ax=b
c	float *	A vector in the objective function min z = cx
num_constraints	int	Number of constraints
num_vars	int	Number of variables
B	int *	Index of the basic variables
N	int *	Index of the non-basic variables
X	float *	An array to hold the Xs in the solution of IP/LP problem of the form min Z=CX s.t.AX=b)

(cont. on next page)

Table3.8. (cont.)

objVal	float	A variable to hold the objective function value(Z) in the solution of IP/LP problem problem of the form min Z=CX s.t.AX=b)
flag	int	A variable to hold the state of the solution computed for the LP problem. Possible values it can be assigned are 0 (FEASIBLE), 1 (FEASIBLE_INTEGRAL) , 2 (INFEASIBLE) and 3(UNBOUNDED)
branching_var_indx	int	A variable to hold the index of the most fractional variable in X if the slution obtained for the LP problem is not integral

Table 3.9. A function for initialization of a GPU

gpu_initialization()	
Purpose	To setup the GPU
Parameters	None : But uses global variables
Steps	➢ Gets a list of OpenCL platforms and chooses a particular platform ➢ Gets a GPU device from the chosen platform ➢ Creates memory buffers on the GPU device to hold the inputs that will be passed to the GPU and to store outputs which result from execution of a kernel ➢ Creates and builds program from the kernel source (i.e. OpenCL file contacting the kernel) ➢ Creates a kernel object which encapsulates the kernel function and makes it accessible from the CPU ➢ Creates a command queue to queue a command for execution on the GPU

Table 3.10. A function for formulation of the an IP problem which helps in sanitization of an input dataset

ip_formulator (.....)				
Purpose	To formulate the IP problem whose solution is used to sanitize the input dataset			
Parameters	***Name***	***Data type***	***Parameter type***	***Description***
	Dataset	String	input	Path to the input dataset file where the input dataset must be in market-basket format
	binaryDataset	String	input	Path to a file where the binary version of the input dataset will be stored
	frequentItemsets	String	input	Path to a file containing the frequent itemsets in the the dataset where the frequent itemsets must be in market-basket format
	frequentItemsets Supports	String	Input	Path to a file containing the support of the frequent itemsetts
	N	Int	input	The number of transactions in the input dataset
	M	Int	Input	The number of different items in the input datset

(cont. on next page)

Table 3.10. (cont.)

	minItem	Int	Input	The minimum item in the dataset (allowed values are 0 or 1)
	Psi	Int	Input	The minimum support threshold
	Q	Int	Input	The number of non-singleton frequent itemsets
	Node	cl_node*	output	A structure to store the formulated IP problem
	AA	String	Input	Path to a file where the A in the formulated IP problem will be stored for debugging purposes(A where min z= cx s.t. Ax=b, x>=0)
	Bb	String	Input	Path to a where the b in the formulated IP problem will be stored for debugging purposes
	Cc	String	input	Path to a file where the c in the formulated IP problem will be stored for debugging purposes
Steps	➢ Reads the input dataset which must be in market basket format and converts it into binary format ➢ Also reads the frequent itemsets and their supports from a file ➢ Asks the user which itemsets are sensitive and splits the frequent itemsets and their supports into sensitive and non-sensitive parts ➢ Determines A, b and c of the sanitization IP problem whose solution indicates the items that need to be removed from the original dataset to sanitize it with a minimum side-effect.			

Table 3.11. A function for solving IP problems using the branch and bound method

best_first_branch_and_bound (...)				
Purpose	To obtain the solution of the formulated IP problem			
Parameters	**Name**	**Data Type**	**Parameter Type**	**Description**
	RootNode	Cl_node*	Input/output	A structure which holds the IP problem which is going to be solved and also which stores the solution of the IP problem
Steps	➢ Solves the IP problem without the integrality constraints ➢ If the solution of the above relaxed solution is infeasible or unbounded, it STOPS without returning a solution as the IP problem doesn't have a solution ➢ Otherwise, if the relaxed solution of the IP is integral, it STOPS since an integral solution has been found ➢ Otherwise, it PUSHES the problem whose relaxed solution is already obtained into a priority queue where the lower bound of the problem is the objective function value in the relaxed solution ➢ Initializes the upperbound to INFINITY ➢ While the priority queue is not empty { ❖ If the size of the priority queue is less than the threshold for using GPU (gpuThreshold), it does the following computations on CPU { ▪ Removes a single sub-problem from the priority queue ▪ Checks whether the removed sub-problem is promising i.e. whether it has a lower bound below the upper bound. If not, it PRUNES the sub-problem as non-promising and it proceeds to another sub-problem			

(cont. on next page)

Table 3.11. (cont.)

	▪ Branches from the first fractional variable in the sub-problem's relaxed solution to create two sub-problems which we call the left and right sub-problems (nodes) ▪ Solves the left and right sub-problems (nodes) without the integrality constraints. ▪ Sets the lower bounds of the left and right sub-problems to their relaxed solutions' objective function values ▪ If the left or right nodes are infeasible or non-promising (with a lower bound above the upper bound), it PRUNES them. ▪ If the left or right nodes have feasible and integral solution , it UPDATES the upperbound to their lower bound in its way and PRUNES them after making them as the current best integral solution ▪ Otherwise it means the left or right nodes have feasible relaxed solution which is not integral but promising with a possibility of giving better integral solution. So, it PUSHES them to the priority queue. } ❖ On the other hand, if the size of the priority queue is equal or greater than the threshold for using GPU(gpuThreshold), it does the following computations. { ▪ It removes nodes from the priority queue upto the threshold for using GPU(gpuThrehsold). ▪ Checks if all of the removed nodes are still promising. If not, it PRUNES the non-promising nodes and removes additional nodes from the priority queue until the number of removed and not-pruned nodes reaches the gpuThreshold. ▪ If the number of nodes removed but not-pruned is below the threshold for using GPU(gpuThreshold), it returns the

(cont. on next page)

Table 3.11. (cont.)

	▪ removed nodes back into the priority queue so that they can be processed with CPU and it GOES to the beginning of the while loop by escaping the computations below. ▪ If the number of removed and not-pruned reaches gpuThreshold, it transfers the removed nodes to GPU where they are branched into left and right nodes and their bounds is computed by solving them without the integrality constraints. ▪ Then, it transfers the nodes(sub-problems) (2*gpuThreshold number of them) whose bound is already computed to CPU ▪ If any of these nodes are infeasible or non-promising (with a lower bound above the upper bound), it PRUNES them. ▪ If it encounters a node with a feasible and integral relaxed solution while traversing the retrieved nodes, it UPDATES the upper bound to its lower bound and it prunes it after making it the current best integral solution ▪ Otherwise it means the unpruned nodes have feasible ▪ relaxed solutions which are not integral but promising with a possibility of giving better integral solution. So, it PUSHES them to the priority queue. } } It returns the solution marked as the current best integral solution as the solution of the IP

Table 3.12. A function for solving LP problems using the revised simplex method without being given an initial basic feasible solution

lpsolve (...)				
Purpose	To solve LP problems (It solves IP problems without the integrality constraints i.e. it solves relaxed IPs which are basically LPs)			
Parameters	**Name**	**Data Type**	**Parameter Type**	**Description**
	phase2	Cl_node*	Input/output	A structure which holds an LP problem which is going to be solved and also which stores the solution of the LP problem
Steps	➢ It considers the given LP problem as an LP problem in phase 2 of the two phase simplex method ➢ It creates the LP problem of phase 1 by the addition of artificial variables to the given LP problem and by setting the objective function coefficients in the given LP problem to zero ➢ It solves the LP problem of phase 1 by the revised simplex method using the artificial variables as basic variables ➢ If the solution of the LP problem in phase 1 is infeasible, it means the original LP problem in phase 2(the relaxed IP problem) is also infeasible. So, it EXITS without returning a solution ➢ Otherwsie,it uses the basic variables in the solution of phase 1 as the basic variables for the original LP problem in phase 2			

(cont. on next page)

Table 3.12. (cont.)

	➢ If the basic variables in the solution of phase 1 contain artificial variables, it applies pivoting to exchange the basic artificial variables with the appropriate non-basic variables which are not artificial ➢ It solves the LP problem in phase 2 using the revised simplex method and using the basic variables obtained from the solution of the phase 1 LP problem

Table 3.13. A function for solving LP problem if given an initial basic feasible solution

revisedsimplexlu ()				
Purpose	To solve an LP problem given the index of the basic variables corresponding to an initial basic feasible solution			
Parameters	**Name**	**Data Type**	**Parameter Type**	**Description**
	node	cl_node*	Input/output	A structure which holds an LP problem which is going to be solved and also which stores the solution of the LP problem given an initial basic feasible solution to the LP problem
Steps	➢ It calculates the index of non-basic variables from the index of basic variables ➢ It splits A and C of the given LP problem into basic and non-basic parts (i.e. AB,AN,CB,CN) ➢ Computes the current basic variable values as $bBar = x_B = AB^{-1}b$ ➢ Initializes the negative reduced cost(negRedCost) to a – negative value e.g. -1			

(cont. on next page)

Table 3.13.(cont.)

	➢ While negRedCost is less than zeron it does the following
	{
	❖ Selects the entering variable. To do this
	• It selects as a pivot column the non-basic column of the simplex tableau corresponding to the least negative negative-reduced-cost(negRedCost). To do this
	• It first computes the negative reduced cost of all non-basic variables as $wN = CN' - (CB * AB^{-1})AN$ or $wN = CN' - u'AN$ $where$ $u = (AB')^{-1}CB$
	• Then it determines the index of the column of the simplex tableau that gives the least negative negative-reduced-cost value of negRedCost and then it computes the entries in the pivot column of the simplex tablue as $aBar = AB^{-1}AN_{pivotColumnIndex}$
	• If there is no non basic variable (column of the simplex tableau) with negative reduced cost, then an optimal solution has been found and no entering non-basic variable can improve the solution, so it RETURNS the current solution.
	❖ Selects the leaving variable. To do this
	• Applies minimum ratio test to determine the leaving variable
	• If a minimum ratio value can’t be found, it means the LP problem is unbounded (i.e. it is possible to increase the value of the entering variable to infinity without making any of the basic variables negative). So, it RETURNS without a solution by indicating the LP problem is unbounded
	• If the minimum ratio is obtained by more than one

(cont. on next page)

Table 3.13.(cont.)

	• rows(or basic variables), it applies Charne's perturbation to determine which basic variable leaves the basis ❖ Applies pivoting to exchange the entering and leaving variables ❖ Updates AB,AN,CB,CN and index of the basic and non-basic variables B and N ❖ Computes the current solution i.e. compute $x_B = bBar\ x_N = 0\ and\ objVal = \ c_B x_B$ } ➢ Determines whether the current solution is feasible and integral. If not, it means the current solution is only feasible but not integral and thus determines the index of the fractional variable(which will be used to perform branching in B&B Returns the solution

Table 3.14. A function for solving linear system of equations using LU decomposition

lu_solve (...)				
Purpose	➢ To solve linear system of equations that arise in the revisedsimplexlu function(It is used to compute $bBar = x_B = AB^{-1}b$, $u = \ AB^{-1}CB$ and $aBar = \ AB^{-1}AN_{pivotColumnIndex}$)			
Parameters	**Name**	**Data Type**	**Paramete r Type**	**Description**
	node	cl_node *	Input	A structure which holds the A matrix of an LP problem(i.e. A where min z=cx s.t. Ax=b)

(cont. on next page)

Table 3.14.(cont.)

	b	float *	input	An array which holds the b vector of an a system of linear equations(i.e. b s.t. Ax=b)
	x	float *	output	An array which holds the x vector of an a system of linear equations(i.e. the solution vector x s.t. Ax=b)
	size	int	input	The dimension of the basic matrix(AB) formed from A of the LP problem, which is the same as the number of constraints in the LP problem
	useTranspose	int	input	If set to 1, it means lusolve will solve AB'*x=b other wise it solves AB*x=b where AB is the basic part of the A matrix in the LP problem
Steps	➢ Given a system Ax=b , it finds L,U and P such that PAx = LUx = Pb ➢ Then it lets Ux=y and forward solves Ly=Pb to obtain y ➢ Then backward solves Ux=y to obtain x			

Table 3.15. A function that performs the LU decomposition of a square matrix

lu()				
Purpose	➢ To find the LU decomposition of a given matrix A(It is used in particular to compute the LU decomposition of AB).			
Parameters	**Name**	**Data type**	**Parameter type**	**Description**
	Node	cl_node*	input	A structure which holds the A matrix of an LP problem(i.e. A where min z=cx s.t. Ax=b)
	L	sparse_matrix *	output	A row-major sparse matrix used to hold L in the LU decomposition the basic part of A, i.e. AB, of an LP problem
	U	sparse_matrix *	output	A row-major sparse matrix used to hold U in the LU decomposition the basic part of A, i.e. AB, of an LP problem
	P	sparse_matrix *	output	A row-major sparse matrix used to hold P in the LU decomposition the basic part of A, i.e. AB, of an LP problem
	size	int	input	The dimension of the basic matrix(AB) formed from A of the LP problem, which is the same as the number of constraints in the LP problem

(cont. on next page)

Table 3.15. (cont.)

	useTranspose	int		If set to 1, it means lu will find the LU decomposition of AB'*. Otherwise, it finds the LU decomposition of AB, where AB is the basic part of the A matrix in the LP problem AB originates
Steps	➢ It returns L,U and P such that PA=LU			

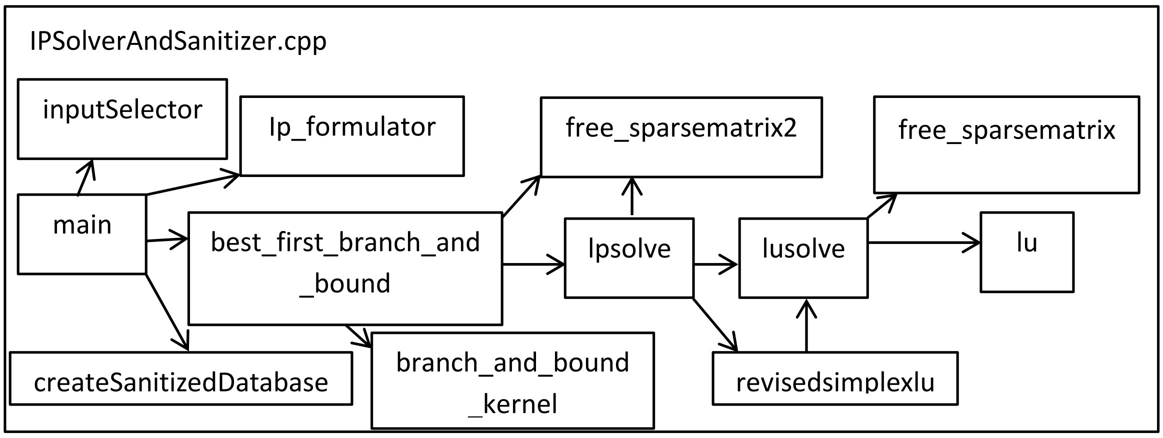


Figure 3.3. Summary of Function calls on the CPU code

3.6. Summary of All Tasks Done on GPU

The tasks done on GPU are

- ➢ To branch on sub-problems (nodes) those were passed to the GPU
- ➢ To compute the bounds of the new sub-problems (nodes) produced by the branching.

The description of the functions that make up the GPU code is shown below

Table 3.16. A function to branch on several IP nodes and compute their bounds in parallel

branch_and_bound_kernel (...)					
Purpose	To solve a list of LP sub-problems in parallel on GPU				
Parameters	**Name**	**Data type**	**qualifier**	**Parameter type**	**Description**
	nAP2L	cl_node *	__global	Input/output	nAP2L is a buffer which refers to node array of phase 2 LP problems in the two phase method(refer section 2.7.2.7) which initially holds input LP problems and which finally stores the new nodes corresponding to left branches on the input LP problems by replacing the input LP problems, where the new nodes have their bounds computed
	nAP2R	cl_node *	__global	output	nAP2L is a buffer which refers to node array of phase 2 LP problems in the two phase method(refer section 2.7.2.7) is used to hold the new nodes corresponding to right branches of the input LP problems, where the new nodes have their bounds computed

(cont. on next page)

Table 3.16. (cont.)

	size	Size	__global	input	The number of LP problems in nAP2L which were passed to the GPU to be processed in parallel, which is the same as the number of right nodes in nAP2R which result from branching on the LP problems which were initially stored in nAP2L.
Steps	➢ It performs branching on the sub-problems(nodes) that were passed to the GPU ➢ It computes the bounds of the new sub-problems produced by branching				

Table 3.17. A function for branching on several IP nodes in parallel

branch (...)					
Purpose	To branch on the fractional variables of the sub-problems that were passed to it				
Parameters	**Name**	**Data type**	**qualifier**	**Paramet type**	**Description**

(cont. on next page)

Table 3.17. (cont.)

	L	cl_node*	__global	Input/output	L is a buffer which refers to node array of phase 2 LP problems in the two phase method(refer section 2.7.2.7) which initially holds input LP problems and which finally stores the new nodes corresponding to left branches on the input LP problems by replacing the input LP problems
	R	cl_node*	__global	output	R is a buffer which refers to node array of phase 2 LP problems in the two phase method(refer section 2.7.2.7) which is used to hold the new nodes corresponding to right branches of the input LP problems
Steps		Given a list of sub-problems(L), it branches on them and stores half of the new sub-problems in the original buffer(L) which is used to hold the input sub-problems and stores the other half of the new sub-problems into another buffer.			

The GPU code also has functions called lpsolve, revisedsimplexlu, lusolve and lu with similar purposes as the corresponding functions in the CPU.

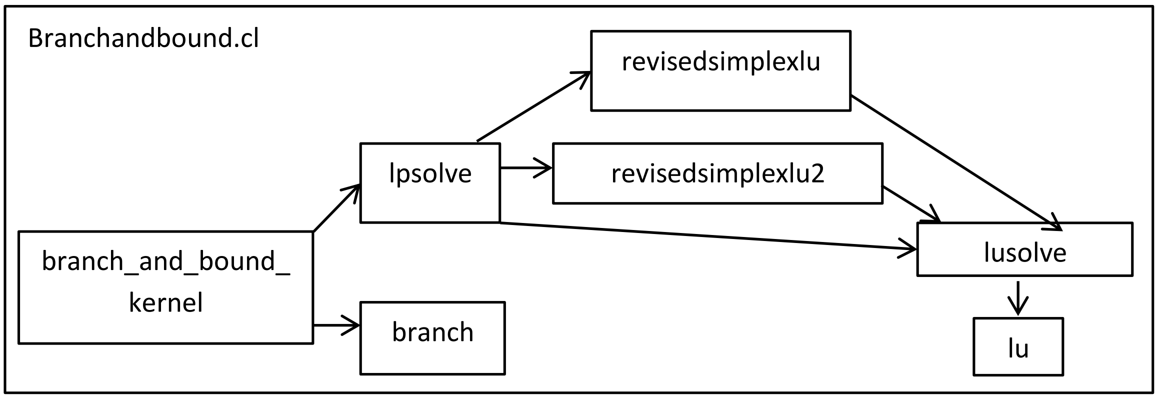


Figure 3.4. Summary of function calls on the GPU code

3.7. Sample Interactions with the Itemset Hiding(Sanitization) Program

Assume we are given the given a small dataset of 5 transactions and 4 itemsets as shown in the figure 3.5 below which is then mined at minimum support threshold of 2 to give the frequent non-singleton itemsets in the figure 3.6 where the support of the frequent itemsets is shown in the figure 3.7.

If we choose the itemset {1,3}in figure 3.6 as sensitive, which is found at row 2 in the figure, then a user’s interaction with our program for sanitizing an input dataset looks as shown in the Figure 3.8.

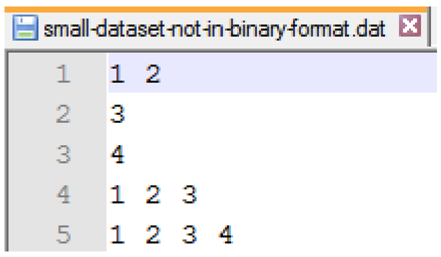
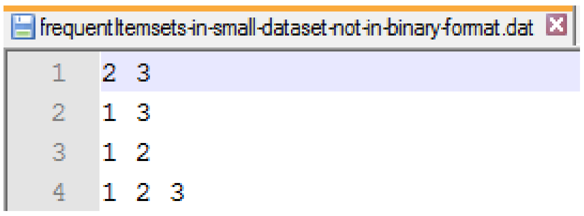
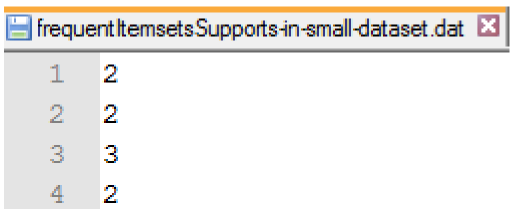

```
small-dataset-not-in-binary-format.dat
1 2
3
4
1 2 3
1 2 3 4
```

Figure 3.5. An input data set in market-basket format



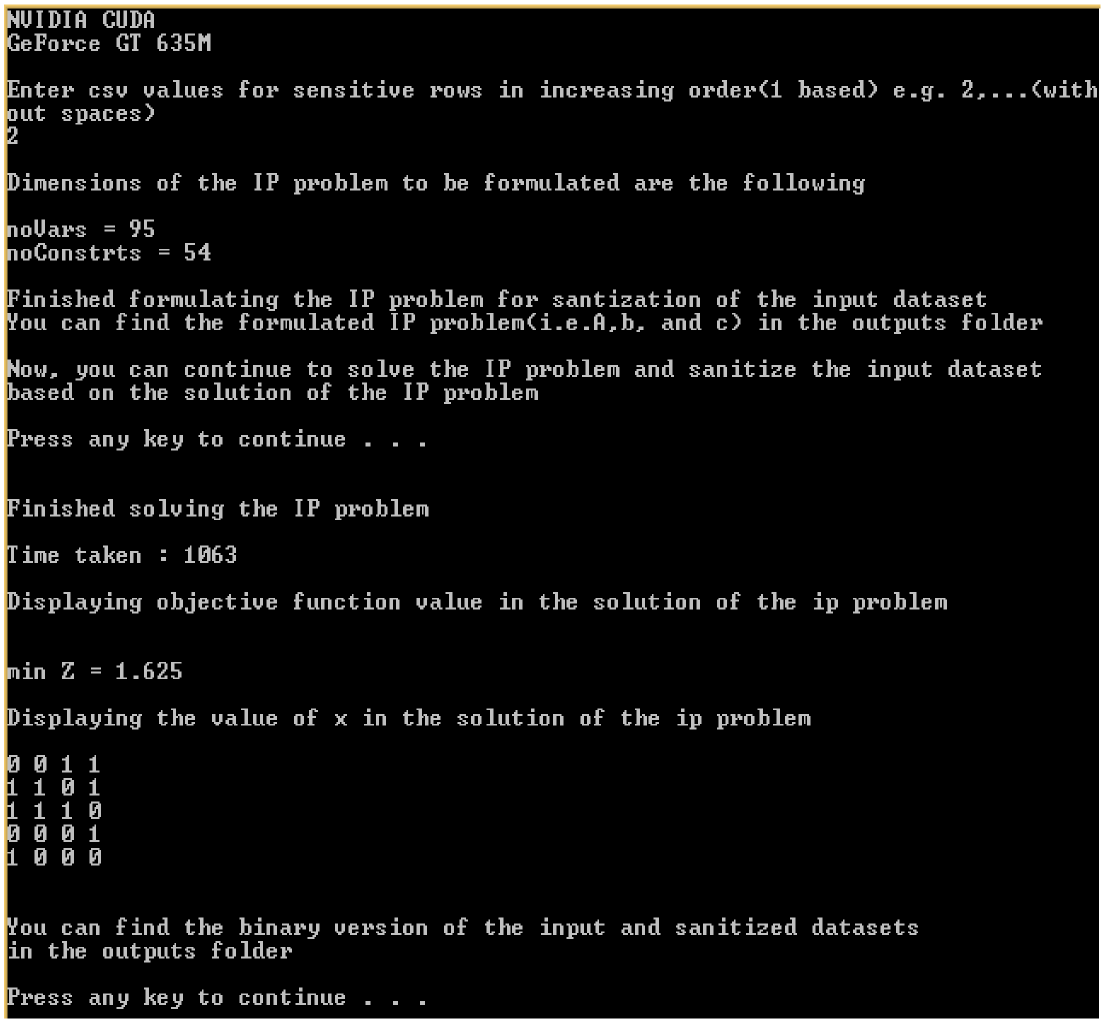
```
2 3
1 3
1 2
1 2 3
```



```
2
2
3
2
```

Figure 3.6. Frequent itemsets

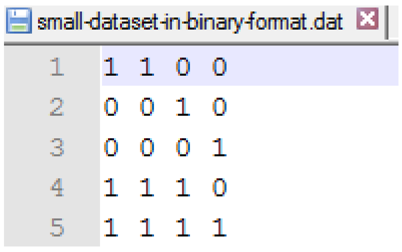
Figure 3.7. Support of the frequent itemsets



```
NVIDIA CUDA
GeForce GT 635M

Enter csv values for sensitive rows in increasing order(1 based) e.g. 2,...(with
out spaces)
2

Dimensions of the IP problem to be formulated are the following

noVars = 95
noConstrts = 54

Finished formulating the IP problem for santization of the input dataset
You can find the formulated IP problem(i.e.A,b, and c) in the outputs folder

Now, you can continue to solve the IP problem and sanitize the input dataset
based on the solution of the IP problem

Press any key to continue . . .


Finished solving the IP problem

Time taken : 1063

Displaying objective function value in the solution of the ip problem


min Z = 1.625

Displaying the value of x in the solution of the ip problem

0 0 1 1
1 1 0 1
1 1 1 0
0 0 0 1
1 0 0 0


You can find the binary version of the input and sanitized datasets
in the outputs folder

Press any key to continue . . .
```

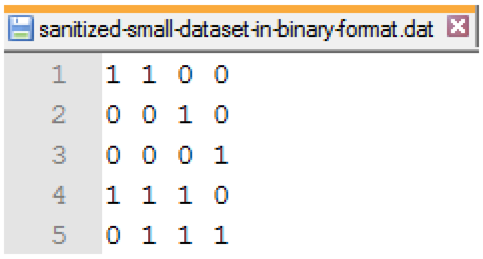
Figure 3.8. Snapshot of the command window during sanitization of a small dataset

At the end of the program, we will find the input dataset in binary format as well as a sanitized version of the input dataset also in binary format as shown in Figure 3.9 and Figure 3.10.


small-dataset-in-binary-format.dat

```
1 1 0 0
0 0 1 0
0 0 0 1
1 1 1 0
1 1 1 1
```

Figure 3.9. The binary form of the input dataset which was in market-basket format


sanitized-small-dataset-in-binary-format.dat

```
1 1 0 0
0 0 1 0
0 0 0 1
1 1 1 0
0 1 1 1
```

Figure 3.10. The sanitized dataset in binary format

3.8. Hardware Used

The computer used to write the sanitization program had an Intel® Core™ i5-3230M CPU @ 2.60GHz. The computer's RAM was 8GHz. The GPU used was NVIDIA GeForce GT 635M which has 90 cores @ 675 MHz and with 2048 MB of memory.

3.9. Software Used

The Integrated Development Environment (IDE) used to write the programs was Visual Studio 2012. The CPU side code was written using C++. And the GPU side code was written using OpenCL.

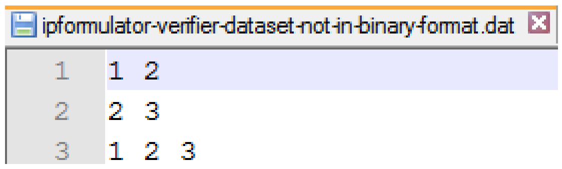
CHAPTER 4

RESULTS

4.1. Formulation of the Sanitization IP problem

As discussed in section 3.1, we have come up with a new IP based sanitization algorithm whose solution is used for sanitization of datasets. Before we try to solve the formulated IP problem and use its solution to sanitize datasets, it was important to first verify whether the IP problem formulated by the program we wrote matches with our proposed sanitization IP problem shown in section 3.1.6. So, different small sample datasets with their corresponding frequent itemsets and sensitive itemsets were given to the part of our program that formulates the sanitization IP problem and this formulation was compared with the IP formulation that can be obtained manually by using the input datasets, frequent itemsets, sensitive itemsets and using the equations in section 3.1.6. Here for demonstration purposes we will use a sample dataset of only 3 transactions and 3 items and show that the sanitization IP problem that was formulated by our program is as expected and as can be manually derived using the equations in section 3.1.6.

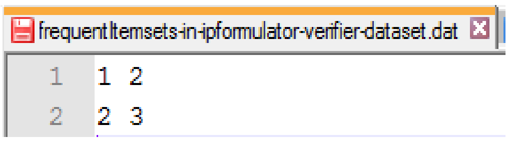
First, the following dataset was given as input to an implementation of the Apriori algorithm to determine the non-singleton frequent itemsets.



```
1 2
2 3
1 2 3
```

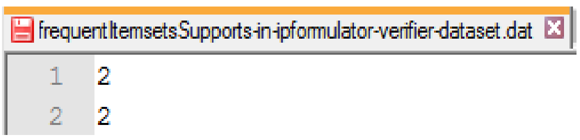
Figure 4.1. A dataset to verify the IP formulation

When the above data set was mined using a minimum support threshold (ψ) of 2, the following frequent itemsets were found and they were then fed as input to the IP formulator together with the input dataset.


frequentItemsets-in-ipformulator-verifier-dataset.dat

```
1 2
2 3
```

Figure 4.2. Frequent itemsets


frequentItemsetsSupports-in-ipformulator-verifier-dataset.dat

```
2
2
```

Figure 4.3. Support of the frequent itemsets

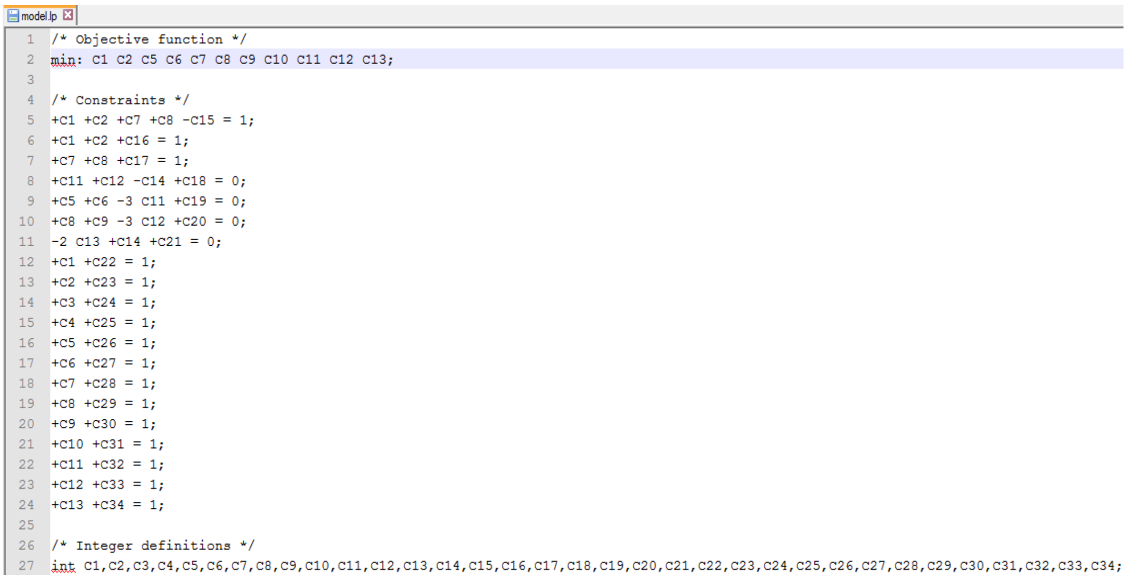
Out of the two frequent itemsets shown above, the first itemset i.e. {1,2} was chosen as sensitive.

So, from the above inputs, we had

- number of transactions in the dataset(n) = 3;
- number of unique items in the dataset(m) = 3;
- number of sensitive itemsets(o) = 1;
- number of non-sensitive itemsets(p) = 1;
- The sum of the support of sensitive itemsets(sensSupSum) = 2;
- The sum of the support of the non-sensitive itemsets(nonSensSupSum) = 2;

Then, when our sanitization IP problem of section 3.1.6 is formulated by our program using the above inputs, the sanitization IP problem that is obtained looked as shown in the Figure 4.4. The IP problem had 34 variables and 20 constraints which was as expected and which was verified by manually formulating the IP problem using the equations in section 3.1.6 and comparing it with the above IP problem.

The 34 variables in the figure represent the different variables found in the IP problem of section 3.1.6. Note that in the sanitization IP problem of section 3.1.6, the variables of the sanitization IP problem are composed of (n*m) number of x_{ij} variables, and similarly (n*p) zir, (p) Ur, (p) Rr, (o) s1k, (sensSupSum) s2ik, (p) s3r, (nonSensSupSumn*p) s4ir, (p) s5r and (n*m) number of s6ij variables. So, the 34 variables shown in Figure 4.4 represent (3*3) number of xij variables, and similarly (3*1) zir, (1) Ur, (1) Rr, (1) s1k, (2) s2ik, (1) s3r, (2) s4ir, (1) s5r and (3*3) number of s6ij variables.



```
/* Objective function */
min: C1 C2 C5 C6 C7 C8 C9 C10 C11 C12 C13;

/* Constraints */
+C1 +C2 +C7 +C8 -C15 = 1;
+C1 +C2 +C16 = 1;
+C7 +C8 +C17 = 1;
+C11 +C12 -C14 +C18 = 0;
+C5 +C6 -3 C11 +C19 = 0;
+C8 +C9 -3 C12 +C20 = 0;
-2 C13 +C14 +C21 = 0;
+C1 +C22 = 1;
+C2 +C23 = 1;
+C3 +C24 = 1;
+C4 +C25 = 1;
+C5 +C26 = 1;
+C6 +C27 = 1;
+C7 +C28 = 1;
+C8 +C29 = 1;
+C9 +C30 = 1;
+C10 +C31 = 1;
+C11 +C32 = 1;
+C12 +C33 = 1;
+C13 +C34 = 1;

/* Integer definitions */
int C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C21,C22,C23,C24,C25,C26,C27,C28,C29,C30,C31,C32,C33,C34;
```

Figure 4.4. The formulated IP problem for sanitization of the dataset

By referring to the equations in the sanitization IP problem of section 3.1.6 we can see that, o number of constraints come from equation (3.9), and similarly sensSupSum constraints from (3.10), p from (3.11), nonSensSupSum from (3.12), p from (3.x), (n*m) from (3.14), (n*p) from (3.15) and p constraints from equation (3.16). So, out of the 20 constraints in the figure above, 1 constraints comes from equation (3.9) and similarly 2 constraints from (3.10), 1 from (3.11), 2 from (3.12), 1 from (3.13), (3*3=9) from (3.14), (3*1=3) from (3.15) and 1 constraints from equation (3.16).

Also note that although we demonstrated the working of the sanitization IP formulation using a small IP problem of 34 variables and 20 constraints, its correctness was also verified for IP problems of over 2000 variables and over 1000 constraints because sanitizations done using such large sanitization IP problems were also observed to remove the sensitive item sets successfully.

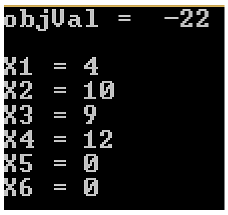
4.2. Solving of Linear Programming (LP) Problems

As discussed in section 3, the algorithm that was used for solving the formulated sanitization IP problems was the branch and bound algorithm. And in order to compute bounds in the branch and bound algorithm the revised simplex algorithm was used, since bounds are obtained by solving Linear Programming (LP) problems (see section 2.7.2 for more information about the simplex method).So, our implementation of the revised simplex algorithm was tested with different sample LP problems to verify if it solves LP problems correctly. For example, we will show below the result obtained when solving a sample Linear Programming problem using our implementation of the revised simplex method.

The standard form of the following Linear Programming problem was given to our implementation of revised simplex algorithm.

$$\begin{aligned} &\min 2x_1 - 3x_2 \\ &\text{s.t. } x_2 \leq 1 \\ &\quad x_1 + x_2 \leq 2 \\ &\quad -0.5x_1 + x_2 \geq 8 \\ &\quad -x_1 + x_2 \geq 6 \end{aligned}$$

And the solution that was obtained is shown in the figure below



```
objVal =  -22

X1  = 4
X2  = 10
X3  = 9
X4  = 12
X5  = 0
X6  = 0
```

Figure 4.5. Screenshot of the solution obtained when solving the LP problem

, where $x_1 = 4$, $x_2 = 10$ with an objective function value of -22. x_3, x_4, x_5 and x_6 were slack and surplus variables which were used to convert the inequalities in the constraints of the LP problem to equalities. We can verify that any other combination of x_1 and x_2 values won't give solution with lesser objective function value while meeting all the four constraints.

But note that the above LP problem was only used for demonstration purposes and it wasn't the only one which was used to verify the correctness of our implementation of the revised simplex method. For instance, linear Programming problems containing over 2000 variables and 1000 constraints were also correctly solved by our implementation. The verifications were done by comparing the result of our implementation with the result of open source linear programming solver libraries like lpsolve (http://sourceforge.net/projects/lpsolve/).

One may ask why didn't use already existing libraries like lpsolve in our implementation. This is because in our project we use two platforms: CPU and GPU depending on the size of the problem (i.e. depending on the number of active nodes in the branch and bound tree). While it is possible to use already available libraries when the program runs on CPU, it is not possible to use similar libraries when the program runs on GPU. This is because the OpenCL library which was used to write programs for the GPU doesn't support the use of libraries, which need dlls(dynamically linked libraries).

4.3. Solving of Integer Programming (IP) Problems

In order to solve the sanitization IP problems that were formulated the algorithm that was used was the branch and bound algorithm. Like the revised simplex algorithm, our implementation of the branch and bound algorithm was also tested with different sample IP problems to verify if it solves IP problems correctly. For example, we will show below the result obtained when solving a sample Integer Programming problem using our implementation of the branch and bound method which in turn uses the revised simplex method to compute bounds.

The standard form of the following Integer Programming problem was given to our implementation of the branch and bound method.

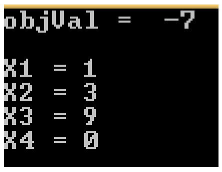
$$\min z = 2x_1 - 3x_2$$

$$\text{s.t.} \qquad -10x_1 + 2x_2 \le 5$$

$$3x_1 + 2x_2 \le 9$$

$$x_1, x_2 \ge 0, x_1, x_2 \in N$$

And the solution that was obtained is shown in the figure below



```
objVal =  -7

X1  = 1
X2  = 3
X3  = 9
X4  = 0
```

Figure 4.6. Screenshot of the solution obtained when solving the IP problem

, where $x_1 = 1$, $x_2 = 3$ with an objective function value of -7. x_3 and x_4 were slack variables which were used to convert the inequalities in the constraints of the IP problem to equalities. We can verify that any other integral combination of x_1 and x_2 values won't give solution with lesser objective function value while meeting all the two constraints.

Like in the LP case, while there are Open Source libraries that can solve IP problems, we can't use such libraries in our project because our program not only runs on CPU but it also runs on GPU. And the OpenCL library which we used to write our GPU code doesn't support the incorporation of external libraries because every code that is written for the GPU is required to be compiled in order to run on the GPU which is not possible for libraries.

4.4. Sanitization of Datasets

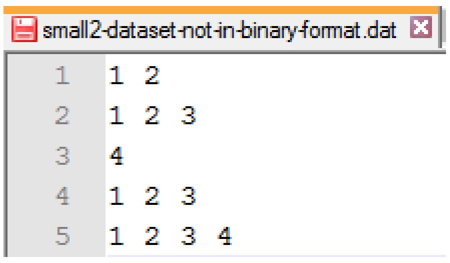
The result of experiments done on different sample datasets using our proposed sanitization algorithm show that it is possible to remove sensitive item-sets from input datasets with minimal impact on non-sensitive item-sets. Since our proposed sanitization algorithm is an exact algorithm (Integer Programming based) its sanitization output is always guaranteed to remove all sensitive item sets while also meeting constraints that ensure minimal impact on the non-sensitive itemsets.

For instance, the results of sanitization of two sample datasets using our proposed sanitization algorithm are shown in the following two experiments and the observations from the experiments are also discussed.

4.4.1. Experiment 1

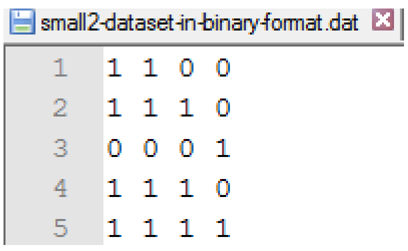
Sanitization Inputs

In this experiment, the dataset used was a sample dataset with 5 transactions and 4 unique items. This dataset is shown in the figure below both in market-basket-format and binary format.


small2-dataset-not-in-binary-format.dat

```
1 2
1 2 3
4
1 2 3
1 2 3 4
```

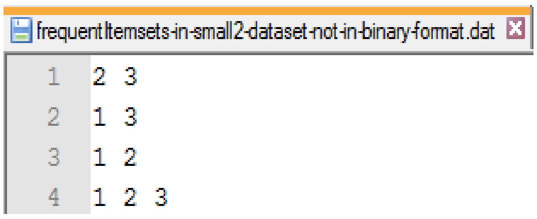
or


small2-dataset-in-binary-format.dat

```
1 1 0 0
1 1 1 0
0 0 0 1
1 1 1 0
1 1 1 1
```

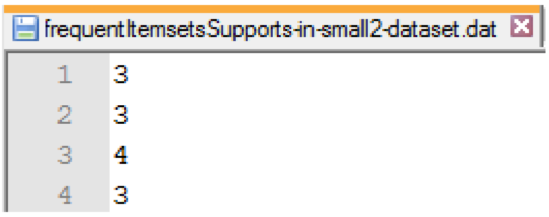
Figure 4.7. Sample dataset called small2

When the above dataset was mined using the implementation of the Apriori algorithm using a minimum support threshold (ψ) of 2, the following non-singleton frequent itemsets were obtained.



frequentItemsets-in-small2-dataset-not-in-binary-format.dat

```
1  2 3
2  1 3
3  1 2
4  1 2 3
```



frequentItemsetsSupports-in-small2-dataset.dat

```
1  3
2  3
3  4
4  3
```

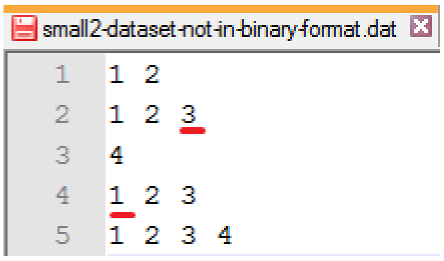
Figure 4.8. Frequent itemsets in small2 itemsets

Figure 4.9. Support of the frequent

In this experiment, out of the four frequent itemsets shown above, the itemset {1, 3} which is found at row 2 of figure 4.8 was chosen as sensitive.

Sanitization Outputs

A sanitization IP problem was then formulated by our sanitization program according to the equations shown in section 3.1.6. The formulated IP problem had 99 variables (including slack or surplus variables) and 58 constraints. This IP problem was then solved by our program using a branch and bound algorithm and its solution gave a 5x4 matrix of x_{ij} values, which were then used to determine which items of the input dataset should be removed to sanitize the input dataset. The 5x4 matrix of x_{ij} values obtained by the solution of the formulated IP problem indicated that the items shown underlined in the figure below had to be removed in order to sanitize the input dataset.

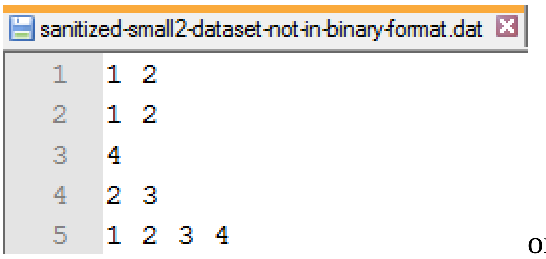


small2-dataset-not-in-binary-format.dat

1 1 2
2 1 2 $\underline{3}$
3 4
4 $\underline{1}$ 2 3
5 1 2 3 4

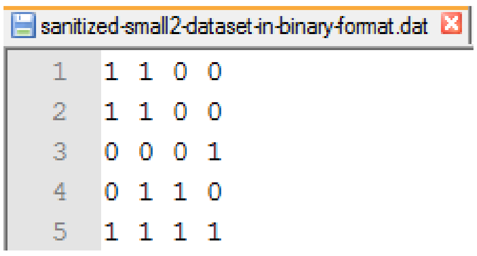
Figure 4.10. Items identified for removal to sanitize small2

The sanitized dataset obtained by removing the above underlined items from the input dataset is shown in Figure 4.11 both in market-basket format and binary format.And When the sanitized dataset was mined using the implementation of the Apriori algorithm and using the same minimum support threshold (ψ) of 2, the non-singleton frequent itemsets shown in Figure 4.12 were obtained.


sanitized-small2-dataset-not-in-binary-format.dat

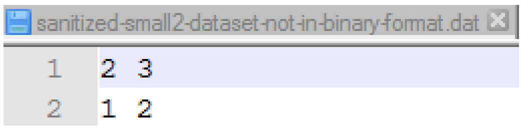
```
1 2
1 2
4
2 3
1 2 3 4
```

or


sanitized-small2-dataset-in-binary-format.dat

```
1 1 0 0
1 1 0 0
0 0 0 1
0 1 1 0
1 1 1 1
```

Figure 4.11. The sanitized version of small2


sanitized-small2-dataset-not-in-binary-format.dat

```
2 3
1 2
```

Figure 4.12. Frequent itemsets in the sanitized version of small2

In conclusion, we can observe from Experiment 1 that the chosen sensitive itemset {1, 3} was successfully hidden from the sanitized dataset. I.e. the sensitive itemset was not frequent when the sanitized dataset was mined at the same minimum support threshold (ψ) of 2. Moreover, the sanitized dataset was obtained from the input dataset while the following side-effects of sanitization were kept minimal (see section 3.1) i) number of items removed from the input dataset ii) the number of non-sensitive itemsets removed from transactions, and iii) the number of non-sensitive itemsets that were made infrequent.

The following table summarizes the performance of the implementation of the proposed sanitization algorithm for the inputs used in Experiment 1.

Table 4.1. Performance of our sanitization algorithm for Experiment 1

Sensitive Itemsets Not Removed	None
Number of items removed during sanitization	2 which accounts for 15.4% of all items found in the original transactions.
Number of transactions affected by sanitization	2 which accounts for 40% of the transactions
Number of non-sesnsitive itemsets hidden as side effect	1 which account for 33.3% of the non-sensitive itemsets.
Sanitization time(in milli seconds)	18 with lpsolve library

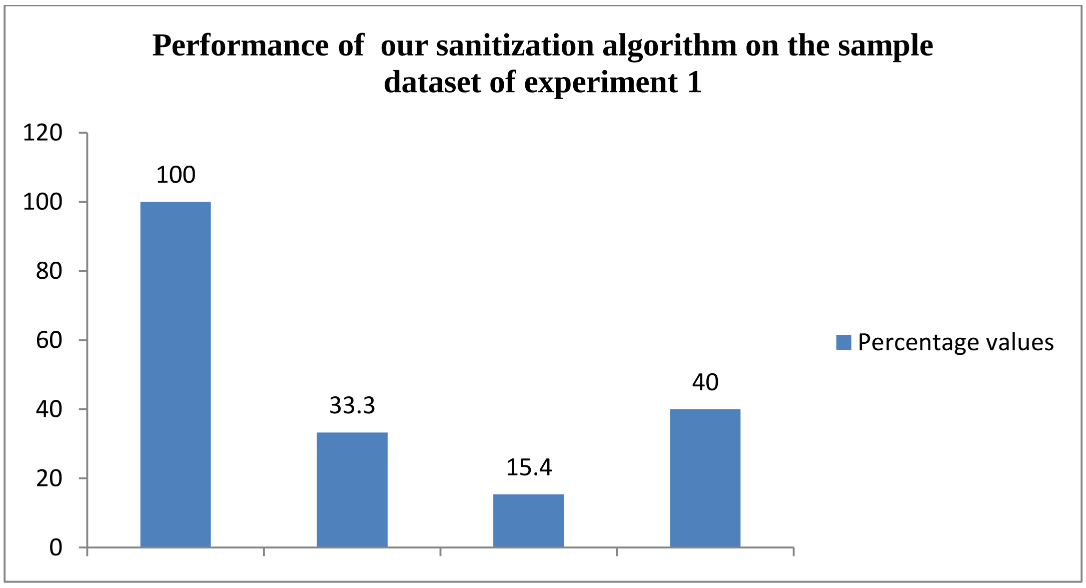


Figure 4.13. Performance of our sanitization algorithm for Experiment 1

4.4.2. Experiment 2

Sanitization Inputs

In this experiment, the dataset used was a randomly generated sample dataset with 100 transactions and 20 unique items. This dataset is shown in the figure below in market-basket-format.

```
3 5 7 8 9 11 14 15 16 17 19 20
1 2 4 5 6 7 11 12 13 14 15 16 20
1 7 13 17 19 20
1 2 3 4 8 9 11 13 14 15 18 19 20
2 4 6 9 11 13 14 15 16 17 19 20
1 4 5 6 7 13 14 15 17 18
3 6 8 9 10 11 13 15 20
1 2 3 6 8 9 10 11 12 13 15 17 18
1 4 5 7 8 10 12 16
3 4 6 7 10 11 12 13 15 18 19
3 6 7 9 12 14 15 16
1 4 5 6 7 9 10 13 14 17 18 19
4 5 6 12 16 17 18 19 20
2 3 7 8 11 12 14 15 16 19 20
1 4 5 6 7 8 9 10 11 12 14 16 17
6 7 9 10 11 15 16 17 18
1 6 7 8 9 11 12 13 19
1 2 3 4 5 10 11 13 14 16
1 3 4 7 8 9 10 11 15 17 19
1 2 6 7 9 11 13 17
1 4 5 7 8 9 12 13 14 15 17 19 20
3 5 8 11 12 13 14 16 18 19
1 5 6 7 8 10
2 3 6 10 13 14 15 18
1 2 7 9 13 15 18 20
1 9 10 12 13 15 16 17 20
1 2 5 10 11 14 15 19 20
2 6 7 8 9 11 12 13 14 15 17 18 19 20
2 6 9 12 15 16 17 20
1 2 5 6 7 8 9 10 13 14 15 16 18 20
1 2 3 4 6 7 8 10 11 13 14 16 17 18
1 5 8 12 13 17 18 19 20
1 3 4 5 8 9 12 14 15 19 20
```

Figure 4.14. A sample dataset for experiment 2

(cont. on next page)

```
3 5 9 11 13 14 16 17 18 19
3 5 7 11 13 16 19 20
2 3 5 6 8 14 15 16 17 19
5 6 7 8 9 10 12 15 17
4 6 7 13 17 18 19
3 6 7 8 9 11 12 15 18
1 2 3 4 6 7 9 11 14 16 18 20
3 4 5 6 8 10 11 13 16 17 18 19 20
4 5 13 16
2 4 6 7 8 10 11 12 13 18 19
2 3 4 7 9 12 17 18
4 5 9 11 14 17 20
1 3 4 6 9 11 12 14 15 17 19 20
3 4 6 7 8 10 12 13 14 15 16
2 3 9 12 13 14 16 18 19
3 7 9 10 12 13 16 17 18 20
2 4 6 8 11 16 17 19 20
1 4 6 8 14 16 18 19 20
1 2 7 9 10 13 14 16 18
2 4 5 7 9 12 14 16 18 20
1 3 7 8 9 10 11 13 17 20
2 3 8 10 11 12 13 16 17 19
1 5 7 10 11 13 15
1 2 5 8 10 12 15 16 17 18 20
1 2 3 6 7 9 10 12 13 14 15 17 18
7 11 15 19
1 7 12 13 15 16 20
2 3 4 6 10 11 14 17 19 20
2 4 5 6 7 9 12 15 16 20
1 3 4 7 10 13 15 16 17 18 20
1 5 7 9 10 14 15 17 18 19 20
2 5 6 9 10 11 14 16
1 3 4 5 8 9 10 11 14 15 18 19
1 2 3 4 5 6 7 8 9 11 13 16 18
2 3 4 5 6 12 14 15 16 17 18
1 3 4 5 6 9 10 12 15 19 20
2 4 5 7 8 9 10 13 14 18 19
2 4 7 9 10 14 15 17 19
2 3 4 6 7 9 10 11 12 14 17 18
1 4 6 7 8 9 10 11 12 13 20
3 4 5 9 10 11 13 16 17 18 19 20
2 3 4 6 8 10 11 12 18 19 20
1 2 5 7 12 13 14 16 20
2 3 5 6 10 11 12 13 15 18 20
1 2 4 5 6 7 8 12 15 16 19 20
1 3 5 6 8 10 11 13 14 15 19
1 2 3 5 7 9 11 13 15 16 17 19 20
```

Figure 4.14. (cont)

(cont. on next page)

1 2 6 8 10 11 12 13 14 15 16 18 20
2 3 4 8 13 14 15 16 17 18
2 3 5 6 7 13 14 15 17 19
1 4 7 8 9 10 11 13 15 16 17 20
1 2 3 4 8 9 10 11 12 15 16 17 19 20
3 6 8 10 11 12 14 15 16 17 18
1 2 3 5 7 10 11 12 18 20
1 2 3 5 6 7 9 10 11 12 14 15 17 19 20
1 2 4 5 6 9 10 11 12 13 14 16 17 18
1 2 4 6 9 10 11 14 15 19
1 2 3 5 7 8 9 10 12 13 14 15 17 19
1 2 5 9 10 11 12 14 15 17 18
3 4 9 10 14 16 17 18
3 4 5 13 14 16 18
1 3 4 6 8 9 10 11 13 14 17
4 5 7 13 15 16 17 19 20
1 5 6 8 9 10 11 15 16 19
3 4 6 7 9 16
1 4 9 11 13 14 16 17
2 3 5 6 10 15 16 17 18

Figure 4.14. (cont)

When the above dataset was mined using the implementation of the Apriori algorithm using a minimum support threshold (ψ) of 30, the following 26 non-singleton frequent itemsets were obtained

6 11
7 13
6 10
10 15
3 10
9 17
1 15
1 13
11 13
14 15
9 14
9 10
7 9
1 9

Figure 4.15. The frequent itemsets in the dataset of experiment 2

(cont. on next page)

3 11
2 14
6 15
10 11
1 11
7 15
15 17
1 10
9 15
1 7
9 11
10 17

Figure 4.15. (cont.)

In this experiment, out of the four frequent itemsets shown above, the itemset {6, 11},{14 15},{7,15} which are found at rows 1,10 and 20 of figure 4.15 were chosen as sensitive.

Sanitization Outputs

A sanitization IP problem was then formulated by our sanitization program according to the equations shown in section 3.1.6. The formulated IP problem had 9538 variables (including slack or surplus variables) and 5192 constraints. This IP problem was then solved by our program using a branch and bound algorithm and its solution gave a 100x20 matrix of x_{ij} values, which were then used to determine which items of the input dataset should be removed to sanitize the input dataset. Once the sanitized dataset is obtained, it was mined again to determine its frequent itemsets and to check whether the sensitive itemsets were hidden and the following 23 frequent itemsets were obtained by mining the sanitized dataset where none of the 3 sensitive itemses were freuqent.

7 13
6 10
10 15
3 10
9 17
1 15
1 13

Figure 4.16. The frequent itemsets in the sanitized version of the dataset of experiment2

(cont. on next page)

```
11 13
9 14
9 10
7 9
1 9
3 11
2 14
6 15
10 11
1 11
15 17
1 10
9 15
1 7
9 11
10 17
```

Figure 4.16. (cont.)

Table 4.2 and Figure 4.17 summarize the performance of the implementation of the proposed sanitization algorithm for the inputs used in Experiment 1.

Table 4.2.Performance of our sanitization algorithm for Experiment 1

Sensitive Itemsets Not Removed	None
Number of items removed during sanitization	4
Number of transactions affected by sanitization	4
Number of non-sesnsitive itemsets hidden as side effect	0
Sanitization time(in milli seconds)	4340 with lpsolve library

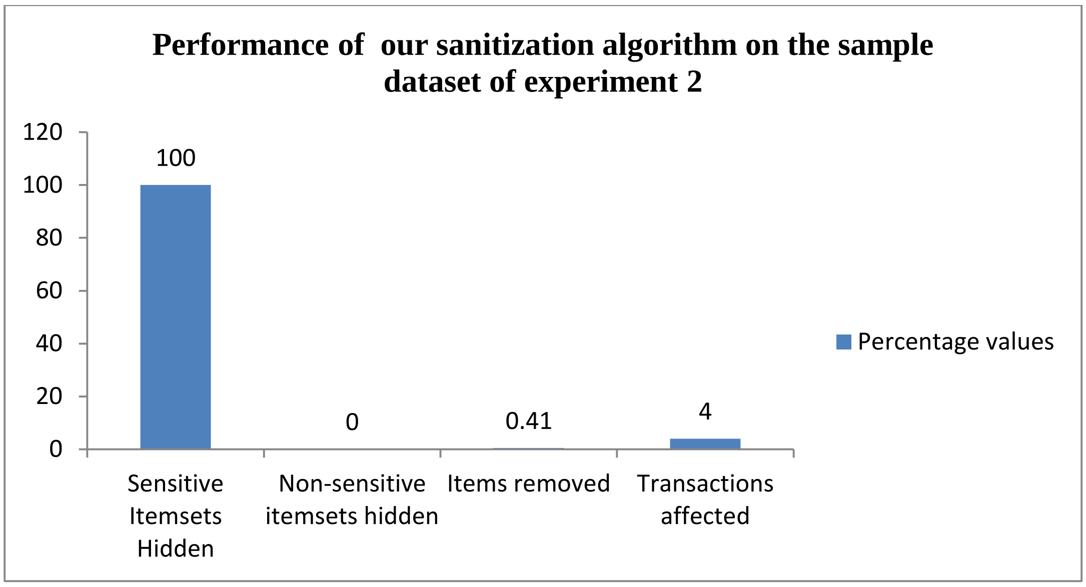


Figure 4.17. Performance of our sanitization algorithm for Experiment 2

Comparing the output of experiment 2 with the outputs of experiment 1 may suggest that as the sanitization problem (the inputs datasets) gets larger the performance of our sanitization algorithm increases in terms of accuracy but this has to be confirmed by carrying out experiments with even larger datasets.

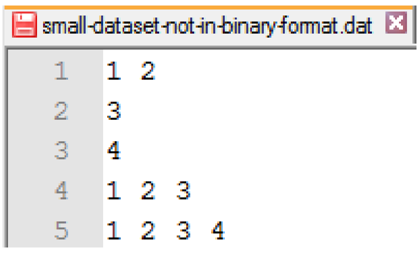
4.5. Implementation on a CPU-GPU platform

Writing a code that only runs on CPU differs from writing code that runs on both CPU and GPU and which works by switching platforms depending on the problem size. While the experiments and results shown before in sections 4.1 through 4.4 were done using a program that was optimized for a CPU platform by using dynamic memory allocation and sparse matrices, it is difficult to create such efficient program for a program that will run on both CPU and GPU platforms. This is because in order to represent IP problems we use a c structure and the way this structure is written must be the same on both the CPU and the GPU codes of the program, when we use OpenCL to write codes for GPU. Unfortunately, we can’t use c structures whose array members’ sizes vary dynamically in the GPU code while this is possible on the CPU code [38]. Thus we are forced not to use arrays with varying sizes as member of structures in the CPU in order to make the CPU structure the same as the GPU’s , which means the

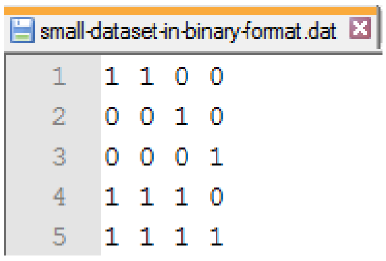
GPU becomes a bottleneck in the way we write the c-structure for representing an IP problem.

Here we will compare the performances that are obtained when doing all the computations on CPU, on GPU and on the combination of CPU and GPU using the versions of our codes that don't use sparse matrix representations in accordance to the limitation that the GPU creates as discussed above. The sanitization inputs used for all the three cases are as shown below.

Common Sanitization Inputs for the Three Cases

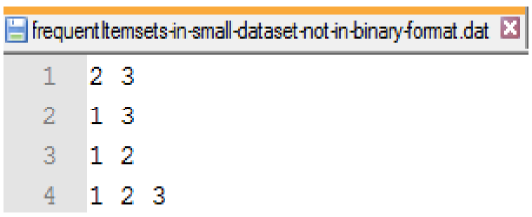

small-dataset-not-in-binary-format.dat

```
1 2
3
4
1 2 3
1 2 3 4
```


small-dataset-in-binary-format.dat

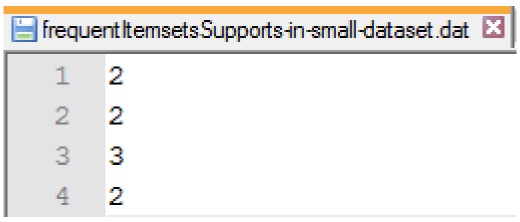
```
1 1 0 0
0 0 1 0
0 0 0 1
1 1 1 0
1 1 1 1
```

Figure 4.18. A sample dataset used to compare the performance of Implementation of the sanitization algorithm on different platforms(CPU,GPU,CPU-GPU) called small


frequentItemsets-in-small-dataset-not-in-binary-format.dat

```
2 3
1 3
1 2
1 2 3
```

Figure 4.19. Frequent itemsets in small


frequentItemsetsSupports-in-small-dataset.dat

```
2
2
3
2
```

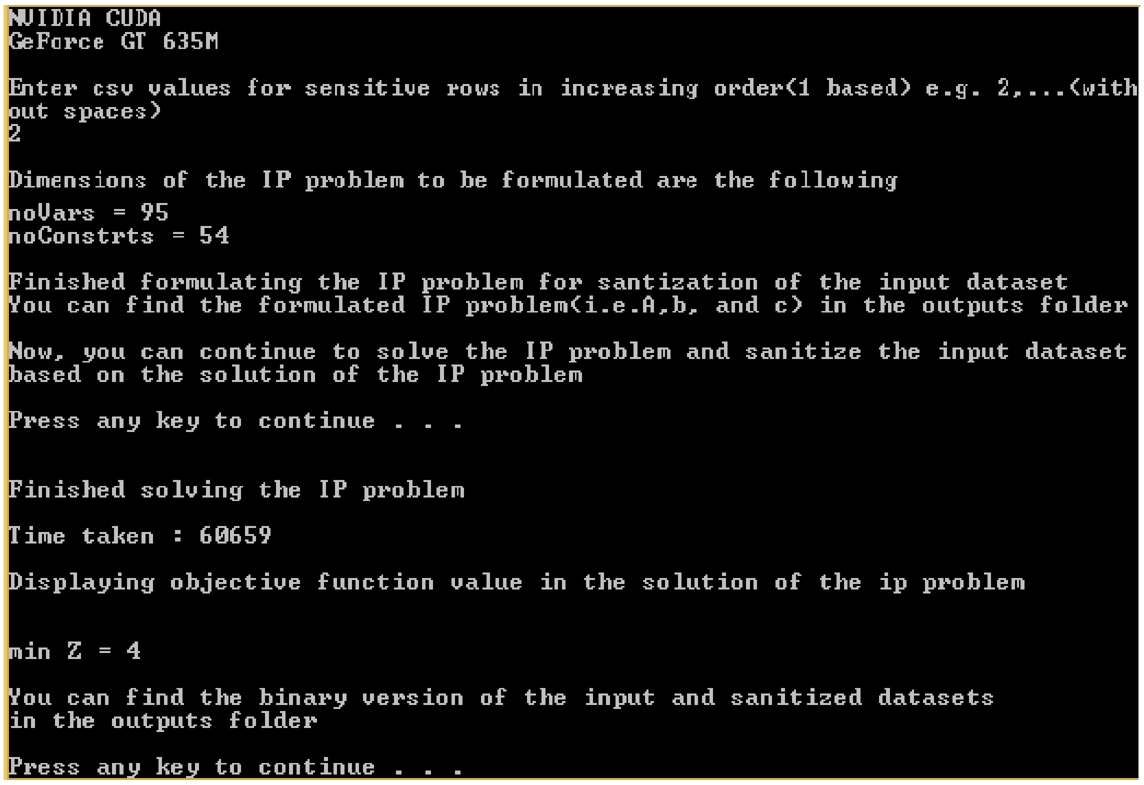
Figure 4.20. Support of the frequent itemsets

Where {1,3} was chosen as sensitive itemset.

Sanitization Ouptus

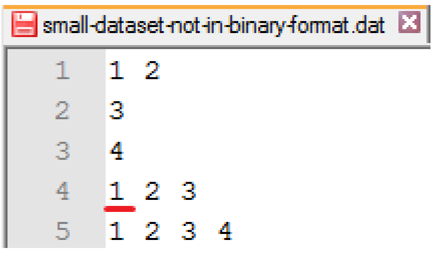
The CPU case

The following outputs were obtained when doing all the computations on CPU.



```
NVIDIA CUDA
GeForce GT 635M

Enter csv values for sensitive rows in increasing order(1 based) e.g. 2,...(with
out spaces)
2

Dimensions of the IP problem to be formulated are the following
noVars = 95
noConstrts = 54

Finished formulating the IP problem for santization of the input dataset
You can find the formulated IP problem(i.e.A,b, and c) in the outputs folder

Now, you can continue to solve the IP problem and sanitize the input dataset
based on the solution of the IP problem

Press any key to continue . . .


Finished solving the IP problem

Time taken : 60659

Displaying objective function value in the solution of the ip problem


min Z = 4

You can find the binary version of the input and sanitized datasets
in the outputs folder

Press any key to continue . . .
```

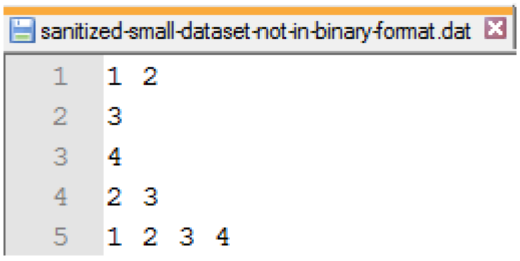
Figure 4.21. Screen shot of command window while trying sanitize sanitize the dataset shown above using CPU only



small-dataset-not-in-binary-format.dat

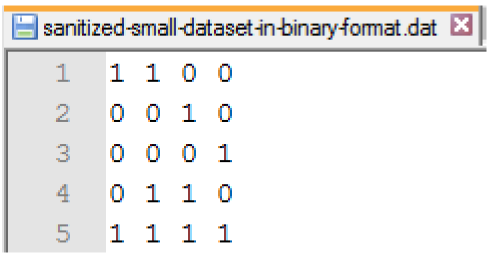
```
1 2
3
4
1 2 3
1 2 3 4
```

Figure 4.22. The item identified for removal by our sanitization algorithm



sanitized-small-dataset-not-in-binary-format.dat

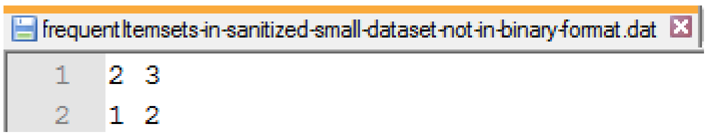
```
1 2
3
4
2 3
1 2 3 4
```



sanitized-small-dataset-in-binary-format.dat

```
1 1 0 0
0 0 1 0
0 0 0 1
0 1 1 0
1 1 1 1
```

Figure 4.23. The sanitized version of the dataset



frequentItemsets-in-sanitized-small-dataset-not-in-binary-format.dat

```
2 3
1 2
```

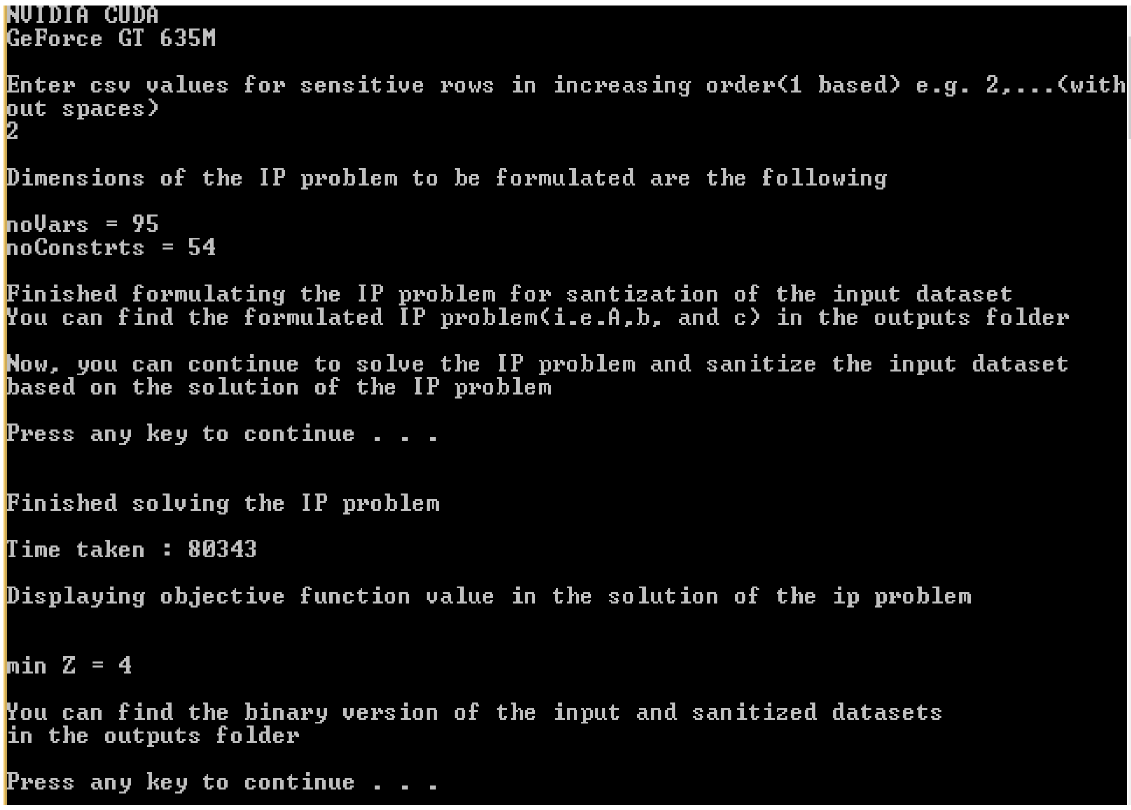
Figure 4.24. Frequent itmestes in the sanitized version of 'small' dataset

Table 4.3. Performance of the implementation of the sanitization algorithm using CPU platform only

Sensitive Itemsets Not Removed	None
Number of items removed during sanitization	1 which accounts for 9.1% of all items found in the original transactions.
Number of transactions affected by sanitization	1 which accounts for 20% of the transactions
Number of non-sesnsitive itemsets hidden as side effect	1 which account for 33.3% of the non-sensitive itemsets.
Sanitization time(in milli seconds)	60659 with our cpu-gpu code, with no sparse matrix representation

The GPU-case

The same outputs as the CPU case were obtained when doing all the computations on GPU as shown below with the exception of the running time.



```
NVIDIA CUDA
GeForce GT 635M

Enter csv values for sensitive rows in increasing order(1 based) e.g. 2,...(with
out spaces)
2

Dimensions of the IP problem to be formulated are the following

noVars = 95
noConstrts = 54

Finished formulating the IP problem for santization of the input dataset
You can find the formulated IP problem(i.e.A,b, and c) in the outputs folder

Now, you can continue to solve the IP problem and sanitize the input dataset
based on the solution of the IP problem

Press any key to continue . . .


Finished solving the IP problem

Time taken : 80343

Displaying objective function value in the solution of the ip problem


min Z = 4

You can find the binary version of the input and sanitized datasets
in the outputs folder

Press any key to continue . . .
```

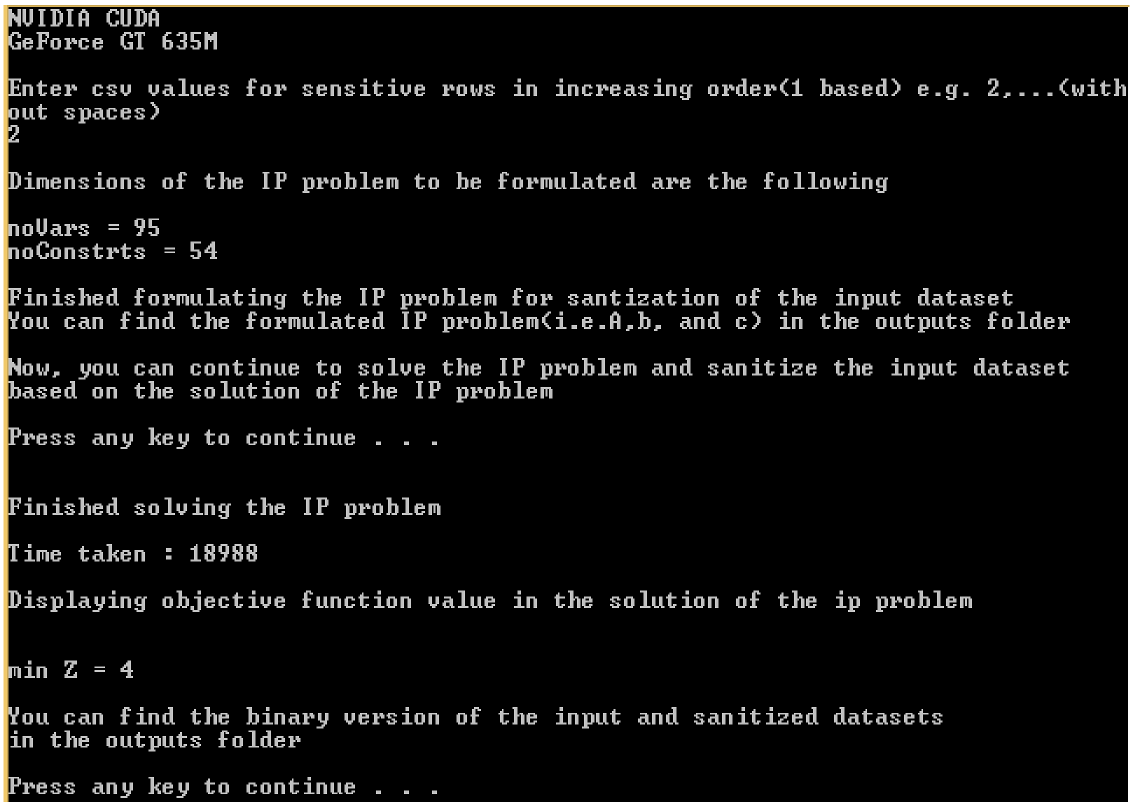
Figure 4.25. Screen shot of the command window during sanitization of the dataset using GPU only

Table 4.4. Performance of the implementation of the sanitization algorithm using GPU platform only

Sensitive Itemsets Not Removed	None
Number of items removed during sanitization	1 which accounts for 9.1% of all items found in the original transactions.
Number of transactions affected by sanitization	1 which accounts for 20% of the transactions
Number of non-sesnsitive itemsets hidden as side effect	1 which account for 33.3% of the non-sensitive itemsets.
Sanitization time(in milli seconds)	80343 with our cpu-gpu code, with no sparse matrix representation

The CPU-GPU Case

And again the same outputs were obtained as in the CPU and GPU cases with the exception of the running time when doing the computations on CPU-GPU platform using a Threshold of 5 which makes computations to be done on CPU when the number of active nodes in the priority queue is below 5 and computations to be done on GPU when the number of active nodes in the priority queue is at least 5. The use of a CPU-GPU platform showed a remarkable speedup over the corresponding computations done using CPU and GPU alone.



```
NVIDIA CUDA
GeForce GT 635M

Enter csv values for sensitive rows in increasing order(1 based) e.g. 2,...(with
out spaces)
2

Dimensions of the IP problem to be formulated are the following

noVars = 95
noConstrts = 54

Finished formulating the IP problem for santization of the input dataset
You can find the formulated IP problem(i.e.A,b, and c) in the outputs folder

Now, you can continue to solve the IP problem and sanitize the input dataset
based on the solution of the IP problem

Press any key to continue . . .


Finished solving the IP problem

Time taken : 18988

Displaying objective function value in the solution of the ip problem


min Z = 4

You can find the binary version of the input and sanitized datasets
in the outputs folder

Press any key to continue . . .
```

Figure 4.26. Screen shot of the command window during sanitization of the dataset using CPU-GPU platoform

Table 4.5. Performance of the implementation of the sanitization algorithm using CPU-GPU platform combination

Sensitive Itemsets Not Removed	None
Number of items removed during sanitization	1
Number of transactions affected by sanitization	1
Number of non-sesnsitive itemsets hidden as side effect	1 which account for 33.3% of the non-sensitive itemsets.
Sanitization time(in milli seconds)	18988 with our cpu-gpu code, with no sparse matrix representation

The following figure summarizes the running times that were required by using different platforms (i.e. cpu only, gpu only and cpu-gpu combination) when sanitizing a sample dataset whose sanitization IP problem contains 95 variables and 54 constraints.

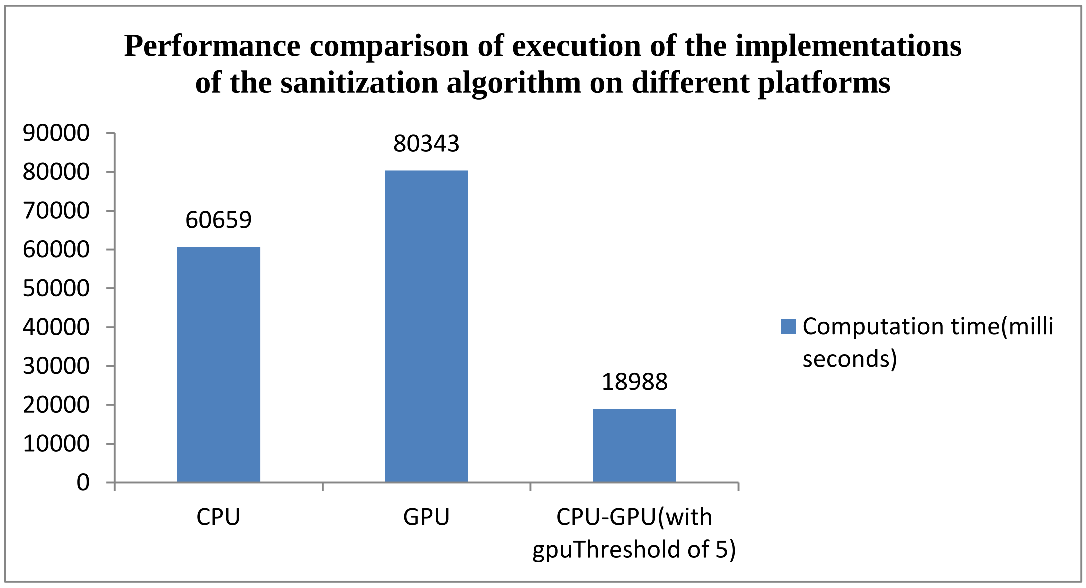


Figure 4.27. Performance comparison of implementing the sanitization algorithm on different platforms using the same sample dataset

CHAPTER 5

DISCUSSION AND CONCLUSION

The goal of our thesis was coming up with a new integer programming based itemset hiding algorithm and parallelization of its implementation with the use of GPU since solving of integer programming problems is NP hard. The algorithm that we proposed is exact which means it removes all sensitive itemsets with the least impact on the non-sensitive itemsets. The correctness of its outputs was verified by using it to sanitize some sample datasets.

Since sanitization using our approach requires solving of an integer programming (IP) problem, the branch and bound method was used to solve a sanitization IP problem. The performance of the branch and bound method largely depends on how efficiently the bound is computed since it involves large number of bound computations. The algorithm that we used for bound computation was the revised simplex algorithm.

Since solving of IP problems is NP hard, we proposed a CPU-GPU architecture to parallelize solving of the sanitization integer programming problem. The codes on the CPU side were written in c++ using visual studio IDE while the codes on the GPU side were writing using OpenCL. The platforms used were a computer with 8GHz of memory and with Intel® Core™ i5-3230M CPU @ 2.60GHz. The GPU used was NVIDIA GeForce GT 635M which has 90 cores @ 675 MHz and with 2048 MB of memory

In the CPU-GPU architecture, when the number of active nodes in the branch and bound tree exceeds a certain threshold, several nodes are transferred to the GPU to perform branching on them and then bound computation on their children. And when the number of active nodes in the branch and bound tree is below the specified threshold, computations are done on CPU since the speed up that could be obtained by using GPU will be offset by the time it takes to transfer nodes between CPU and GPU. Implementation of our sanitization algorithm on a CPU-GPU platform shows that the CPU-GPU architecture provides speedup over using CPU or GPU alone.

Limitations and Future Work

At the time of working on this thesis there was no dynamic memory allocation support in OpenCL which prevented us from using sparse matrix representations [38]. And without sparse matrix representations computations will be slow and the GPU's memory will quickly get full and the program will run out of memory. This limitation can be eliminated if support for dynamic memory allocation (like malloc of c) is incorporated in future releases of OpenCL. In an article published on March 2014, the authors of [38] have designed a memory manager called KMA that provides generic malloc() and free() APIs. So, this limitation will probably be overcome in the near future by using such APIs.

While our implementation of the proposed sanitization algorithm shows that it can successfully remove sensitive itemsets while meeting all constraints specified in the proposed IP based sanitization algorithm, our implementation of an integer programming problem solver was found to be not as fast as open source implementations. Our investigation of why this difference aroused showed that we need to use a number of tricks to speed up the revised simplex algorithm that is used for bound computations. One technique we used to speed up our implementation was the use of sparse representation for large matrices and this resulted in a significant speedup. Other techniques that can be used to speed up our implementation are suggested below.

Since degeneracy (see section 2.7.2.7) was observed to frequently occur in the linear programming problems that are solved by the simplex method during bound computations, we tried applying Charnes perturbation method to reduce the effect of degeneracy in the simplex method[16]. But, it didn't result in performance improvement as expected. So, if other degeneracy handling algorithms are used, a better speed up of the simplex method might be obtained. In addition, the following technique that can speed up the simplex method was found from our literature review. Since the implementation of the revised simplex method involves LU factorization of the basis AB and its transpose AB' after each iteration when solving for $\mathrm{AB} * \mathrm{u} = \mathrm{CB}$ and $\mathrm{AB}' * \mathrm{aBar} = \mathrm{AN}_{\mathrm{pivotColumnIndex}}$ as shown in Table 3.13, it would be time consuming if the LU factorization of AB and AB' is done after each iteration. Thus, updating of the basis AB and its transpose after each iteration rather re-factorizing them can increase the performance of simplex implementation. So, in the future the Bartels-Golub-Reid update [39] or similar basis update techniques can be tried to speed up the implementation of the revised simplex method.

REFERENCES

[1] Hand, D. ; Mannila H. ; and Smyth P. ; 2001; “Principles of Data Mining“ ; book published by The MIT Press, chapters 1.1, 1.8 and 5.1

[2] Bramer, M; 2013; “Principles of Data Mining“; Springer, 2nd ed, 2013 edition

[3] Maimon , O. ; Rokach L; 2010; “Data Mining and Knowledge Discovery Handbook”, second edition,Springer, chapter 15

[4] Agrawal R. and Srikant R. 1994 ;“Fast Algorithms for Mining Association Rules in Large Databases”;In International Conference on Very Large Data Bases, pages 487–499.

[5] Borgelt C. ; 2005; “An Implementation of the FP-growth Algorithm” ; In International Workshop on Open Source Data Mining, pages 1–5.

[6] Borgelt C. ; 2003 ; “Efficient Implementations of Apriori and Eclat.” ; In Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2003, Melbourne, FL)

[7] Agrawal R. and Srikant R. ; 2000 ; “Privacy Preserving Data Mining” ; In the proceedings of the 2000 ACM S1GMDD international conference on management of data

[8] Evfimievski A., Gradison T.; 2009; “Privacy Preserving Data Mining”; *Handbook of Research on Innovations in Database Technologies and Applications : Current and Future Trends* (pp. 527-536) , USA ;

[9] Gkoulalas-Divanis A. , Verykios V.S ; 2010 ; “Association Rule Hiding for Data Mining”; Springer New York Dordrecht Heidelberg London;

[10] Sathiyapriya , K. ; Sadasivam G. S. ; 2013; ” A Survey On Privacy Preserving Association Rule Mining” ; International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.3, No.2,

[11] Jadav K. B.; Vania J.; Patel D. R.; 2013 ; ” A Survey on Association Rule Hiding Methods”;In International Journal of Computer Applications, Volume 82 , No 13

[12] Genova K. ; Guliashki V. ; 2011; “Linear Integer Programming Methods and Approaches – a Survey”, Cybernetics and Information Technologies, BAS, Vol. 11, No 1, pp. 3-25, ISSN 1311-970

[13] Chandru V. ; 2010; “Integer Programming”; In the book *Algorithms and theory of computation handbook* , Chapman & Hall/CRC, pp. 31-1 to 31-47, ISBN: 978-1-58488-822-2

[14] Ibaraki, T; 1976; “Integer programming formulation of combinatorial optimization problems”; In the journal Discrete Mathematics 16, pp. 39-52

[15] Yang, X-S. ; 2008 ; “Introduction to Mathematical Optimization : From Linear Programming to Metaheuristics”; a book published by Cambridge International Science Publishing, pp 67-72 and pp 1-6;

[16] Sinha, S. M. ; 2006; “Mathematical Programming : Theory and Methods”; a book published by Elsevier Science & Technology,pp 133-138 and pp 165-172

[17] Nocedal, Jorge Wright, Stephen J.; 2006 ; “Numerical Optimization”; published by Springer, pp 372-390

[18] Jeannin-Girardon, A; Ballet, P; Rodin,Vincent ; 2013; “A Software Architecture for Multi-Cellular System Simulations on Graphics Processing Units”; in the journal *Acta Biotheoretica* , Volume 61, Issue 3, pp 317-327

[19] Banzhaf, W; Harding, S; Langdon, W.B. , Wilson, G; 2009;” Accelerating Genetic Programming through Graphics Processing Units”;in the book *Genetic Programming Theory and Practice VI, Genetic and Evolutionary Computation* , pp 1-19

[20] Owens, J. D. ; Luebke D. ; Govindaraju, N.; Harris M. ; Krüger J.; Lefohn A.E. ; Purcell T. J.; 2007;” A Survey of General-Purpose Computation on Graphics Hardware”; in the journal *computer graphics forum,* Volume 26, Issue 1,pp 80-113

[21] Owens J. D., ; Houston Mike,; Luebke D.; Green S.; Stone J. E., ; Phillips J. C.; 2008; “GPU Computing”; *Proceedings of the IEEE,* Vol. 96, No. 5, May 2008, pp 879-899

[22] Nickolls, J. and Dally, W.J.; 2010; ” The GPU Computing Era”; in the journal *Micro, IEEE, pp 56-69*

[23] Hallmans, D. ; Asberg, M. ; Nolte, T; 2012; ”Towards using the Graphics Processing Unit (GPU) for embedded systems” ; *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA),* 2012,pp 1-4

[24] Khronos OpenCL Working Group; 2012 ; "The OpenCL specification, version 1.2."; Technical report, Khronos Group, 2012

[25] Munshi A.; Gaster B.; Mattson T. G.;Ginsburg D.; 2011;"OpenCL programming guide"; Pearson Education, Jul 7, 2011

[26] Fang J.; Varbanescu A.L. ; Sips H . ; 2011 ; "A Comprehensive Performance Comparison of CUDA and OpenCL" ; In proceedings of the 2011 International Conference on Parallel Processing (IEEE Computer Society Washington, DC, USA), pp. 216-225

[27] Achterberg T. ; Koch T.; Martin A. ; 2005; " Branching rules revisited";In Operations Research Letters, Volume 33 Issue 1,pp. 42-54

[28] Bertsimas D. ; Tsitsiklis J. ; 1993 ; "Simulated Annealing"; in statistical science, vol. 8, no. 1, pp 10-15

[29] Fernando R.; Kilgard M. J.; 2003 ; " The CG Tutorial : The Definitive Guide to Programmable Real-Time Graphics", Addison-Wesley Professional, ISBN 0 - 321-19496-9

[30] Tarditi D.; Puri S. ; Oglesby J; 2006; "Accelerator : using data parallelism to program GPUs for general-purpose uses"; In ASPLOS XII Proceedings of the 12th international conference on Architectural support for programming languages and operating systems,pp. 325-335

[31] Oneppo M.; 2007; " HLSL shader model 4.0"; In SIGGRAPH '07 ACM SIGGRAPH 2007 courses, pp. 112-152

[32] Robere R.; 2012; "Interior Point Methods and Linear Programming"; University of Toronto

[33] Gondizo J.; Terlaky T.; 1996; "A computational view of interior point methods"; in the book *Advances in linear and integer programming*, pp. 103-144

[34] Boukedjar A.; Lalami M. E.; El-Baz D.;2012; " Parallel Branch and Bound on a CPU-GPU System"; 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2012

[35] Chakroun I., Melab N. ; 2012; "An Adaptative Multi-GPU based Branch-and-Bound. A Case Study: the Flow-Shop Scheduling Problem" ; 4th IEEE International Conference on High Performance Computing and Communications, HPCC 2012

[36] Lalami M. E. ; Boyer V. ; El-Baz D.; 2011 ; “Efficient Implementation of the Simplex Method on a CPU-GPU System”; In the 11th Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum

[37] Bieling J.; Peschlow P. ; Martini P.; 2010; ” An Efficient GPU Implementation of the Revised Simplex Method”; In the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)

[38] Spliet R.; Howes L.; Gaster B. R.; Varbanescu A. L. ; 2014; “KMA: A Dynamic Memory Manager for OpenCL”; In proceedings of Workshop on General Purpose Processing Using GPUs; ACM New York

[39] Reid J. K. ; 1982; ”A sparsity-exploiting variant of the Bartels—Golub decomposition for linear programming bases”; In Mathematical Programming, 1982, Volume 24, Issue 1, pp 55-69