Cost Optimization of Scrap When Making Steel with an Electric Arc Furnace

Wang Weifeng

Master of Engineering

Department of Mechanical Engineering

McGill University

Montreal,Quebec

2012-02-03

A thesis submitted to McGill University in partial fullfillment of the requirements of the degree of Master of Engineering

ACKNOWLEDGEMENTS

First and foremost, I would like to express my heartfelt thanks to my supervisor Prof. Vincent Thomson for his constant support and guidance over the course of my Master Degree. He is one of the most patient people I have known at McGill, and it was truly a pleasure working with him. I am also grateful to my friend and colleague Dr. Onur Hisarciklilar for the time he spent helping me in various aspects of my research. His passion for continous improvement is infectious and it has reignited my own interest in the subject. Thank you Dr. Hisarciklilar for your help with programming using Visual Basic for Applications. I would like to thank my colleagues in the lab and engineers at ArcelorMittal who were dedicated to this project. My McGill colleagues, Kailash Bathy Voeyar Math and Prof. Jung Eui Hong; the engineers at ArcelorMittal, Pascal Gaudreau and Simon Chevalier. I would like to thank my friends and family for being there for me through the good times and the bad times. I would not have made it this far without your support and encouragement. Finally, I would like to thank The Mathematics of Information Technology and Complex Systems (MITACS) program and ArcelorMittal for their financial support during this project.

ABSTRACT

In steel production, an electric arc furnace (EAF) is most commonly used to melt raw material in order to produce liquid steel. Scrap is the main raw material which differs in regard to the content of iron and of some chemical elements. The price of scrap depends on these attributes. In order to obtain the desired quality and quantity, each melting bath unit of steel has either its own material constraints or the constraints for electric arc furnace such as the capacity of EAF. In addition, the availability and transportation of scrap are also restricted because they need space. The research in this thesis is to create an optimization model which minimizes the cost of raw material and charges the EAF efficiently while meeting the constraints of the scrap recipe and scrap transportation system. This problem is a combinational optimization problem and the model is developed based on linear programming theory. The running speed of the model is reasonably guaranteed by properly designing the combinatorial structure with branch and bound rules and heuristics. Finally, a software is created by representing the model in the spreadsheet, which can be used in real, everyday production. Simulation results show significant improvement compared to the strategy applied today at ArcelorMittal(Contrecoeur, Quebec): the cost of scrap steel is reduced by 2 to 6% and the time of charging buckets is 2 to 10 minutes faster.

ABRÉGÉ

Dans la production d'acier, four à arc électrique (FEA) est la technologie la plus couramment utilisée pour faire fondre les matières premières afin de obtenir de l'acier liquide. La ferraille est la matière première principale dont les types se diffèrent selon le contenu de fer et d'autres éléments chimiques. Le prix de la ferraille dépend de ces attributs. Afin d'obtenir la qualitè et la quantitè souhaitées, chaque unité bain de fusion de l'acier a subi à ses propres contraintes matérielles ou des contraintes liées au four à arc électrique, telles que la capacité du FEA. En outre, la disponibilité et la capacité à transporter de la ferraille sont également limitèes, en raison d'espace limité. L'objectif dans cette thèse est de créer un modèle d'optimisation qui minimise le coût des matières premières et charge le FEA efficacement afin de satisfaire des contraintes de la recette de ferraille et de transport de ferraille. Le modèle est développé sur la base de théorie de la programmation linéaire. La vitesse de l'exécution du modèle est raisonnablement garantie par une bonne conception de la structure combinatoire avec les règles de 'branch and bound' et heuristiques. Enfin, un logiciel qui applique le modèle est créé. Celui-ci peut être utilisé dans la production réelle quotidienne. Les résultats des simulations montrent une amélioration significative par rapport aux pratiques actuelles de planification de production appliquée aujourd'hui dans ArcelorMittal (Contrecoeur, Quebec): le coût de la ferraille est réduite de 2 à 6 pour cent et le temps de godets de charge est de 2 à 10 minutes plus vite.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Steel is one of the world's most important materials. Steel markets in Canada are increasingly influenced by global markets where over one-third of global steel production is traded internationally. According to Statistics Canada, Canada produces 13 million tonnes of steel per year, [1].

Steelmaking in Canada takes two basic forms: the integrated process and the Electric Arc Furnace (EAF) process.

i) Integrated Process (BOF). This process has three raw materials: iron ore, coke and limestone from which molten iron is produced in a blast furnace. Molten iron and smaller amounts of scrap steel are transformed into liquid steel in a Basic Oxygen Furnace (BOF) process. Finally, liquid steel is cast into slabs and billets for further processing.

ii) Electric Arc Furnace (EAF). The main raw material is scrap steel, which is reheated, melted into liquid steel and cast into new steel bars for further processing. EAF is widely used in steelmaking. Currently 1/3 of world production uses EAF [3]. In the EAF, recycled steel is melted and transformed into high quality liquid steel by using a high-powered electric arc. The basic set-up of an electric arc furnace is shown in Figure 1-1.

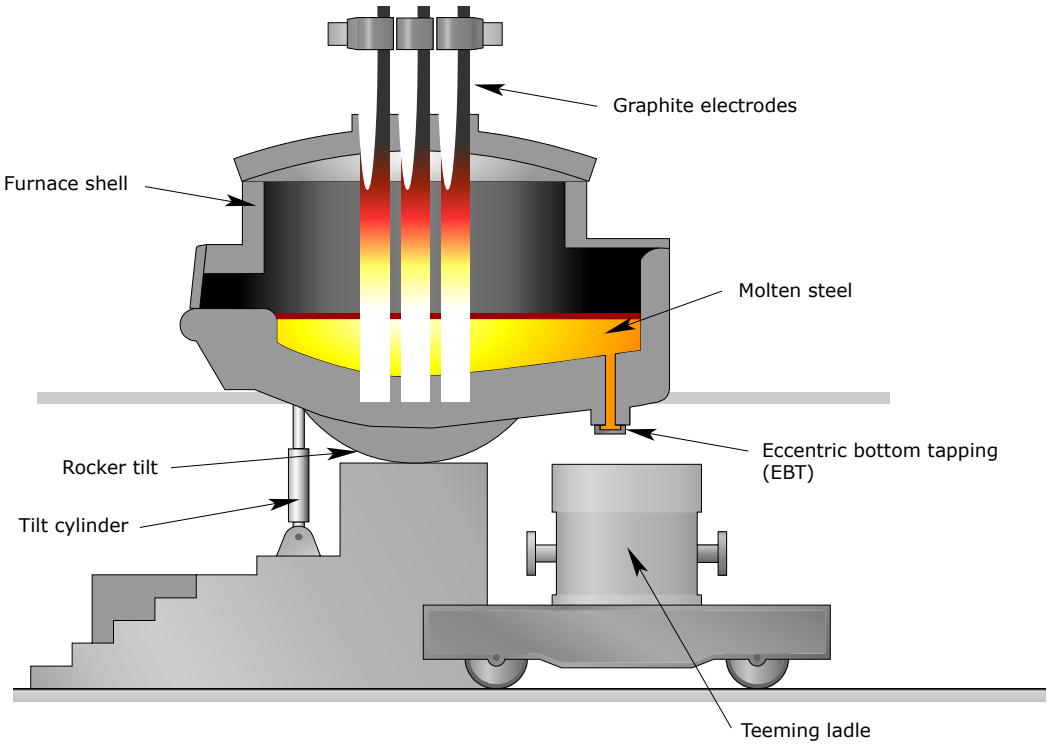

Figure 1–1: Electric Arc Furnace [4]

Steel plants are located wherever it is economically feasible to bring together large quantities of raw materials. The big steel plants in Canada have been built at locations along the Great Lakes-St Lawrence Seaway system. ArcelorMittal, the steel company we study in this project, is located in Contrecoeur, Quebec, close to the Saint Lawrence river with railroad connections.

1.1 Problem Statement

In general, steel production can be divided into several steps [10]:

- Raw material processing
- Process metallurgy

- Casting and solidification
- Hot and cold forming
- Mechanical processing

In steel production, the cost of raw materials represents 85 % of the total cost of crude steel production [12]. There are many types of scrap with different properties, such as chemical element contents, yield, density and price. Appendix A gives a list of different types of scrap. A recipe using scrap steels is prepared in order to fulfill the requirements for producing the desired steel types at a low-cost. Therefore, many steel companies or technology service firms are trying to obtain an optimal recipe from different types of raw materials in order to reduce production cost.

All the steps in steel production must be well coordinated to guarantee production consistency and smooth operation. During the phase of raw material processing, there are many operations that include scrap transportation and EAF loading. Good coordination of these operations are needed in order to speed up the process.

In this thesis, we address the first two steps in steel production: raw material processing and process metallurgy. The aim of this project is to build optimization models to obtain (1) a daily-production-based recipe of scrap with a focus on cost reduction, and (2) better coordination between scrap charging operations to increase production rate.

1.2 Thesis Objective

We have two main objectives in this work: (1) optimize the recipe to reduce cost, and (2) optimize scrap transportation and loading operations to increase production rate.

The current production cannot be adjusted quickly enough with respect to cost, demand and production capacity. To satisfy the first objective of this thesis, an intelligent model is constructed, which can dynamically update the recipe according to the flexible market price information for scrap steel and optimize production cost. In addition to reducing cost, the model should run within a reasonable period of time in order to serve as a useful tool in daily production. It also needs to be able to integrate new types of scrap, as the facility's scrap portfolio expands.

Badly coordinated scrap transportation and loading operations can cause interruption or delay in production and can impact production cost. Therefore, the second objective is to optimize the scrap charging process so that buckets of scrap for the EAF are charged as quickly as possible. It is therefore required to coordinate operations in the scrap supply system, including railcar layout and bucket loading for fast and flawless operation.

1.3 Thesis Organization

This thesis deals with the development and validation of a dynamic optimization model for steel production in an EAF. Chapter 2 introduces the background information about scrap types and charging operations during steel production. Chapter 3 gives a literature review on optimization techniques and existing models

for optimization of EAF operations. Next, in Chapter 4, a scrap cost optimization model for EAF steel production is developed using linear programming. Branch and bound rules, along with heuristics and other techniques, are used to improve model speed. Chapter 5 and 6 present the scrap distribution model, which is composed of a car loading model and a bucket charging model. Chapter 5 illustrates how the car loading model works and mimics the car loading operations in the scrap yard, an area near the entrance to the EAF. Chapter 6 details the car distribution in the scrap hall. This model consists of three algorithms: cars layout algorithm, bucket layer algorithm and combination algorithm. Then, Chapter 7 shows the industrial application of our model with case studies at ArcelorMittal. Finally, in Chapter 8, concluding remarks are made and suggestions are given for future improvements to the model.

CHAPTER 2
Background Information

A simple model of the steel production process at ArcelorMittal is shown in Figure 2-1. The following paragraphs detail recipe building and scrap charging operations which occur at the beginning of the process to put the correct scrap into the EAF to produce the correct steel for a customer order.

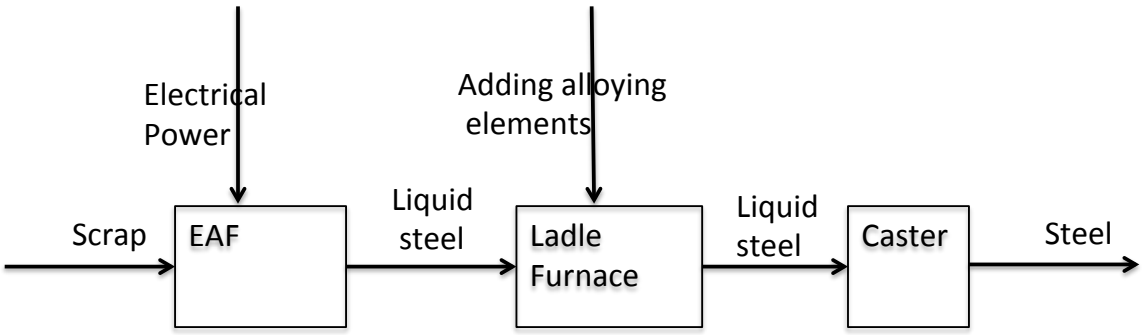

Figure 2–1: Steel production process

2.1 Scrap Types and Recipe

Although the quality of solid steel is usually based on mechanical properties such as tensile strength and corrosion resistance, the quality of liquid steel is is usually based on its chemical composition. The target chemical composition defines the desired limits of some critical chemical elements in the liquid steel. These elements include copper, chromium, nickel, sulphur, etc.

Recycled scrap steel is the main raw material used in an Electric Arc Furnace. In Canada, the steel recycling rate stands in excess of 60% and 7 million metric tons of steel are recycled in 2010 [1].

High quality scrap steel has a low level of impurities and its size can be modified by mechanical means. Several types of scrap are available in the scrap market, and are classified according to several properties:

- Chemical composition of the steel
- Level of impurity elements, e.g., Cu, Cr, Sn, S
- Casting and solidification
- Shape and physical property, e.g., hardness
- Melt yield, i. e., the percentage of the liquid steel produced from scrap

A few types of scrap are described below. These scrap steels have standard names to facilitate trading in the market. The names used here follow the recommendations of the US-based Institute of Scrap Recycling Industries (ISRI). At ArcelorMittal, there is an internal code for each type of scrap, which is given in the brackets.

- Shred(104, 204, 404, 504)

  Pieces of light steel arising (arising means steel works, forgers, machinists). Be free from dirt, non-ferrous metals and foreign material with no excessive moisture, including cast iron, turnings and borings. Must follow specifications: minimum density of 0.8 tonnes per m^3, average content of Cu of 0.35 %

- Busheling(105, 126, 305)

Clean scrap steel, for example, sheet clippings, stampings, etc. Not including old auto body and fender stock. Free from coated metal, vitreous enameled and electrically modified sheet. Cu content of 0.12%

- Plate and structure (111)

  Consists of cut structural and plate arisings 6 mm thick with sizes not exceeding $15m \times 0.6m \times 0.6m$. May include properly prepared wagon material less than 6 mm thick. The wagon is a usually four-wheeled vehicle for transporting bulky commodities and drawn originally by animals. Excludes tubes and hollow sections

- ♯ 1 (101)

  Predominantly 6 mm thick, less than $1.5m \times 0.6m \times 0.6m$ in size. May include tube and hollow section, wire rope, properly prepared material from heavy commercial vehicles including wheels, but excluding body and wheels from light vehicles.

- ♯ 2 (112, 326)

  Predominantly 3 mm thick less than $1m \times 0.6m \times 0.6m$ in size. May include properly prepared material from dismantled vehicles including light vehicle wheels. Excludes vehicle body and appliances

Appendix A gives information about the types of scrap used at ArcelorMittal, including chemical element content (such as Cu, Cr, Sn, S), melt yield and cost.

A scrap recipe is the composition of scrap to be melted in an EAF, which fulfills (1) chemical requirements to achieve the final product and (2) mechanical requirements coming from the EAF design and operations. As the basis of steel

production, a recipe is a mixture of scrap in different quantities, is built to fulfill these requirements. An example recipe is shown in Table 2-1. As can be seen in Table 2-1, the content of a recipe includes the following information: steel grades, the amount of needed scrap, the estimated copper content, cost and yield. Among them, the amount of needed scrap is used to organize production activities, including loading cars and buckets with scrap, charging EAF, helping purchasers acquire scrap from external companies, and production analysis. The copper content is used to distinguish one recipe from another. An estimation of cost is also given to be used as part of the total cost of production. At ArcelorMittal (Contrecoeur, Quebec), the production coordinator manually produces a recipe according to his experience.

2.2 Scrap Charging Operations

When a recipe is determined, scrap from inside the scrap yard is loaded into railcars and transported to the scrap hall. Inside the scrap yard, scrap is separately piled. These piles are beside a railroad, which allows the railcars to gather them and transport them to the scrap hall. Once inside the scrap hall next to the EAF, scrap is put into buckets using a magnetic crane; the scrap is poured from the buckets into the EAF. The proximity of each scrap in the railcars to the buckets determines the speed of the loading process. This can be controlled through two parameters: the way the railcars are loaded, and the way buckets are loaded (bucket layering).

Table 2–1: Example recipes

Scrap	0.50-0.80% Cu max	0.35-0.40% Cu max	0.3%Cu max	0.25%Cu max
DRI	0	0	0	0
Busheling	0	30000	60000	40000
Plate and Structural	0	20000	30000	70000
Shred	60000	60000	70000	70000
♯ 1	70000	60000	30000	20000
♯ 2	30000	20000	0	0
Tire wire	0	0	20000	20000
Turnings	30000	20000	0	0
Tin Cans	20000	0	0	0
Fonte	0	0	0	0
External Skulls	30000	20000	0	20000
Internal Skulls	0	0	10000	0
Billets	0	0	10000	0
Rolled Revert	0	10000	10000	0
Charges (lbs)	240000	240000	240000	240000
Cu (%)	0.32	0.30	0.24	0.21
Melt Yield(%)	88.71	90.92	92.94	92.31
Cost ($/T)	324.92	341.46	335.79	357.29

2.2.1 Railcar Loading

Scrap is supplied from the scrap yard by railcars. A commonly used railcar dimension is shown in Figure 2-2. In the hall, railcars are kept on the track. Railcar loading refers to the required number of railcars for each type of scrap and how they are assembled on the track. Figure 2-3 shows the layout of the scrap hall. As seen from Figure 2-3, railcars are divided into two fleets and usually each track has six cars in the hall. Some scrap, such as external and internal skulls (a skull is a layer of solidified metal or dross on the wall of a pouring vessel often when the metal has been poured.), are not loaded into cars. Instead, they are located beside the two tracks. Aside from skulls, 2-3 other types of scrap must be fully loaded into the cars. A crane loads the buckets which are located in two terminals of the hall.

Figure 2–2: A railcar containing scrap.

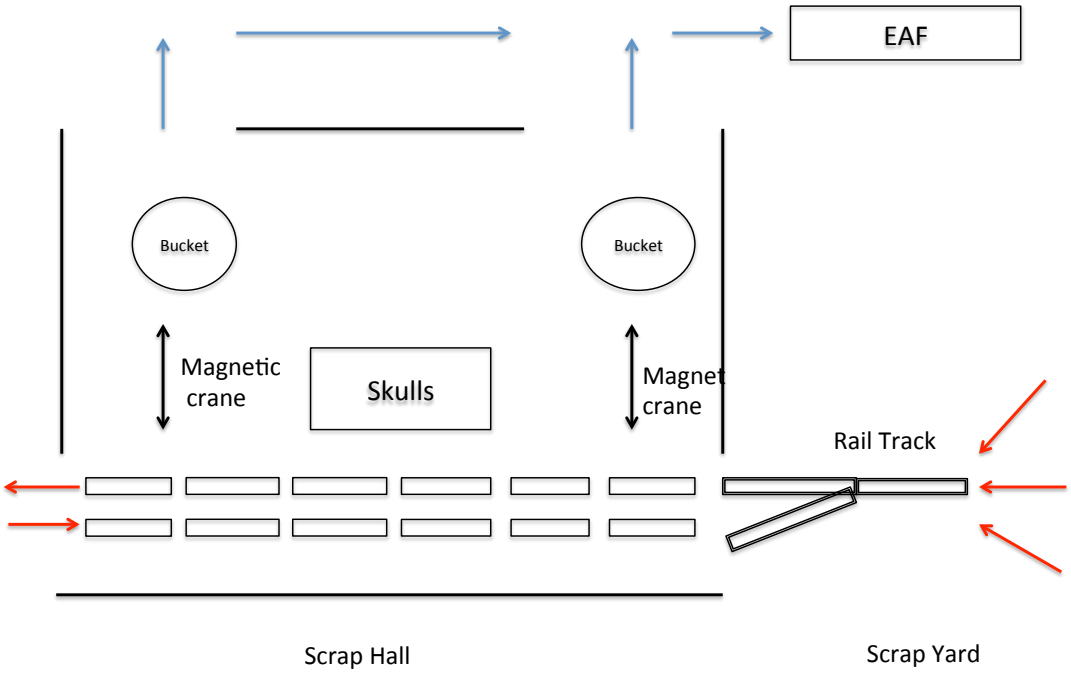

Figure 2–3: Scrap hall layout

According to the production recipe, all the railcars are firstly filled in the scrap yard. There are two types of railcars with volumes of 3510 and 3360 ft^3, respectively. After being fully loaded, scrap is chosen from the railcars in the scrap hall. The scrap dispatcher checks railcars in the scrap hall to decide when to call the transportation system to take empty railcars out and fetch full railcars. The scrap dispatcher system and transportation system cooperate to obey the same constraints in order to guarantee that the train arrives at the hall on time to make the whole process continuous.

2.2.2 Bucket Loading

Scrap is loaded into the EAF using buckets, and usually 2 to 3 buckets of scrap are melted per cycle. Each bucket is loaded with a few layers of different scrap (usually 6 to 8 layers). After charging each bucket, three graphite electrodes are lowered towards the scrap, the electrical power is switched on and the scrap is melted. Then, the next bucket of scrap is loaded into the furnace. This process is repeated until the required amount of melted steel is obtained. When the liquid steel pool has obtained the required composition and temperature, the power is switched off and the furnace is tapped.

Electrode breakage occurs occasionally in the EAF, mainly when the electrodes are lowered down and hit hard steel. The cost associated with electrode breakage is usually very high and interruption in steel production occurs until the time the broken electrodes are replaced with new ones. Therefore, electrode breakage must be avoided. Electrode breakage usually results from a mechanical load on the electrode from the scrap inside the EAF. As the electrode penetrates the scrap pile in the furnace, if the electrode directly hits the hard scrap, the electrode might break. At ArcelorMittal, scrap on top of the pile is chosen to be very soft and easy to penetrate.

An example of bucket layering is shown in Figure 2-4. A bucket is divided into three layers. The scrap in each layer should obey some constraints. For example, scrap in the top layer in the bucket should be soft and thick enough to avoid breaking electrodes. When penetrating the top layer in the EAF, electrodes should not hit hard scrap. So, the thickness of this layer should be at least greater

than the length of electrode in the furnace. This kind of scrap is also loaded in the bottom layer in the bucket in order to make the loading easy and protect the furnace. At ArcelorMittal, scrap in these two layers is usually filled with shredded materials. In the middle layer, other types of scrap are loaded. The way of layering in the bucket is decided by the scrap dispatcher system according to the production recipe.

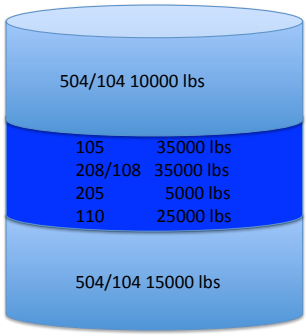

Figure 2–4: Scrap layering in the bucket.

A crane with a magnet end effector is used to load and unload metal from railcars at the EAF. A crane is shown in Figure 2-5.

Figure 2–5: The magnet used to carry scrap.

CHAPTER 3
Literature Review

3.1 General Concepts

3.1.1 Optimization

In mathematics, computational science, or management science, optimization refers to the selection of a best element from some set of available alternatives [13]. An optimization problem refers to maximizing or minimizing a real function by systematically selecting input values from within a defined set and calculating the value of the function. Optimization plays an important role in decision science and in the analysis of physical systems.

Optimization algorithms are usually iterative. At the beginning, the optimal values are initiated with a guess, an estimate of the variables. Then, a sequence of estimations improve the optimal values. At the end of several iterations, a final solution is reached.

An optimization problem can be represented as follows:

Given: a function $f : \mathbf{A} \to \mathbf{R}$ from some set $\mathbf{A}$ to real numbers

Sought: an element x_0 in $\mathbf{A}$ such that $f(x_0) \leq f(x)$ for all $\mathbf{x}$ in $\mathbf{A}$(minimization) or such that $f(x_0) \geq f(x)$ for all $\mathbf{x}$ in $\mathbf{A}$ (maximization).

Such a formulation is called an optimization problem or a mathematical programming problem. The function f is called an objective function, cost function (usually for a minimization problem) or utility function(usually for

a maximization problem). **A** is some subset of the Euclidean space $\mathbf{R}^n$, often specified by a set of constraints, equalities or inequalities that the members of **A** have to satisfy. The domain **A** of f is called the search space , and the elements of set **A** are called feasible solutions.

We also mathematically represent this optimization problem:

x is the vector of *variables*

f is the objective function, a function of x that we want to maximize or minimize

c is the vector of *constraints* that outputs must satisfy, including both equality and inequality.

Minimize

$$f(x)$$

Subject to

$$c(x) = 0$$

$$c(x) \geq 0$$

$$c(x) \leq 0$$

A feasible solution is a member of a set of possible solutions to a given problem. A feasible solution does not have to be a likely solution to the problem. It is simply in the set that satisfies all constraints.

An optimal solution for a minimization problem is a point at which the objective function is smaller in the entire feasible region, which is also called a global optimal solution with respect to local optima. Global solutions are highly

desirable, but they are hard to identify and even harder to locate in complex problems. In these cases, the fastest optimization algorithm only looks for a local solution, a point at which the objective function is smaller than all other feasible points in its neighbourhood. This does not always guarantee that the global optimal solution is found. This generates a trade-off between operation speed and the quality of solution.

Linear Programming Linear programming is a specific case of mathematical optimization with a linear objective function and linear constraints, which may include both equalities and inequalities. The feasible region is a convex polyhedron, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality.

Linear programs are usually stated in the following canonical form

Minimize

$$c^T x$$

subject to

$$Ax = b, x \geq 0$$

$$Ax \geq b$$

Comparing the above mathematical formulation with the general one, [using equation numbers] the objective function and constraints are linear, which indicates linear programming is a subset of mathematical optimization.

Linear programming is used in business and economics, but can also be utilized for some engineering problems. Industries, such as transportation, energy,

telecommunication, and manufacturing, use linear programming models. Linear programming has proved useful in modelling various types of problems in planning, routing, scheduling, assignment and design.

Integer Programming

In certain optimization problems the variables do not make sense unless they take on integer values. Suppose that in the transportation problem, factories produce tractors. In this example, the variable x represents an integer value rather than a real number. This strategy is to solve the problem with real variables ignoring the integrality requirement and then round all the components to the nearest integer, which by no means guarantees to give solutions that are close to optimal. Problems of this type should be dealt by discrete optimization. As for the mathematical formulation, the following constraint is added to the existing constraints

$x_i \in \mathbf{Z}, for\ all\ i$

where $\mathbf{Z}$ is the set of all integers. This problem is known as an integer programming problem. Some models include variables varying continuously and others can take on only integer values. These are referred as *mixed integer programming* problems.

Combinatorial Optimization

Combinatorial optimization [5], [15], [16] is a lively field of applied mathematics, combining techniques from combinatorics, linear programming, and the theory of algorithms, to solve optimization problems over discrete structures. As

several studies show, most combinatorial optimization problems belong to the *NP*-hard class for which efficient algorithms often do not exist. This class of problems solvable in polynomial time is usually denoted by P. As for the class NP, they are a class of problems that might be larger. NP does not mean "not polynomial time", but instead stands for "non-deterministic polynomial time".

To increase the solvability of the problem, different approaches of mathematically representing the problem can reformulate the problem before obtaining an optimal solution to a large integer programming in a reasonable amount of computing time. In this aspect, such problems work well by increasing the number of integer variables, the number of constraints, or both. Unlike linear programming, the feasible region of the combinatorial problem is not a convex set, which makes finding an optimal solution to combinatorial optimization problems a difficult task.

Different problems were researched to build the model such as transportation problem, assignment problem, and scheduling problem. Details can be refered to [20]. There exists many different approaches for solving integer programming problems, such as enumerative techniques and cutting planes.

Enumerative Approaches: This approach enumerates all the finitely many possibilities, which is the simplest approach to solving a pure integer programming problem. The most commonly used enumerative approach is branch and bound, which is extensively employed in our model and is illustrated in the following section.

Cutting Plane algorithms based on polyhedral combinatorics: This approach makes significant computational advances in exact optimization. When polyhedral

theory, developed over the last twenty-five years, is applied to numerical problem solving, the size and complexity of the problems are considerably increased. A polyhedron is the intersection of finitely many spaces, which is the set of feasible solutions to linear programming with efficient algorithms and the solutions can be reduced to linear optimization problems over polydrydra as well. The idea underlying polyhedral theory is to use an alternative convexification of the feasible points and extreme rays of the problem instead of the constraint set of an integer programming problem.

3.1.2 Branch and Bound Rules

Combinatorial optimization problems share the following properties:

- They are optimization problems and have a finite but usually very large number of feasible solutions
- The majority of the problems has no method that is solvable in polynomial time.

All of these problems are *NP*-hard. For example, vehicle routing, crew scheduling, and production planning. These problems are often solved with a great deal of computation. Thus very efficient algorithms are required. Branch and Bound (B&B) is by far the most extensively used tool for solving large scale *NP*-hard combinatorial optimization problems [11]. The method consists of a systematic enumeration of all the feasible solutions in which large subsets of fruitless candidates are discarded by using bounds on the variable being optimized. Solutions are obtained through a search of the solution space. Initially there is only one set of solutions. The unprocessed subspace is represented as nodes in a dynamically

generated search tree, which is explored by each iteration of a B&B algorithm. Each iteration has three principal tools: node selection strategy, bound calculation, and branching (moving to a new section in the solution space). In order to fathom nodes as early as possible, a good initial feasible solution is also very important. This problem is illustrated in the heuristics section. Now, we introduce how to develop a bounding function, branch rules and a node selection strategy.

Bounding Function

A bounding function provides upper and lower bounds for the minimum value of f(x) within a given subset. A bound is called strong if it gives a value close to the optimal value for the bounded subproblem, and weak if the value is far from the optimum. A bounding function is the most important component in a B&B method. The more time spent on calculating the bound, the better the bound value is. Normally speaking, a bounding function should be as strong as possible while keeping the size of the search tree as small as possible.

Take the minimization problem as an example. There are mainly two ways of performing a bound calculation. The first one is to use relaxation, i.e., leave out some constraints in the problem so that the subset is enlarged. The optimal solution to the relaxed problem is used as a lower bound if it does not satisfy all the constraints in the original problem or as a new incumbent otherwise. The second approach is to modify the objective function ensuring that for all feasible solutions the modified function has values less than or equal to the original function while maintaining the feasible solution to the problem. A lower bound can be obtained by solving the modified problem.

Selection Strategy

This strategy is used to select the live solution subspace to be investigated in the current iteration, which reflects a trade-off between keeping the number of explored nodes in the search tree low and staying within the memory capacity of the computer.

The first commonly used strategy is called the best first search strategy. In this approach, one always selects the solution space with lowest bound in all the subproblems. If a subproblem has a bounding function value smaller than the optimal solution of the problem being explored, this problem is critical. Only a critical node is explored; otherwise, it is discarded.

A computer memory problem arises when using the best first search strategy if the number of critical problems is large. In this situation, a breath first search strategy is used, in which the algorithm firstly processes all nodes at one level of the search tree before any node at a higher level. When the number of nodes at each level of the search tree grows exponentially with the level, this strategy is not realistic for large problems.

The third strategy is called depth first search. In this case, a live node with the largest level in the search tree is chosen for exploration. The advantage for programming is the use of recursion to search the tree which allows storage of the information about the current subproblem in an incremental way. Contrarily, the incumbent is far from the optimal solution, the computations are very demanding.

According to the specific case, one can combine two of these strategies. This type of combination are detailed later.

Branching

A branching rule is applied to subdivide the subspace into more subspaces to be investigated in subsequent iterations if a subspace after investigation cannot be discarded. The most common way is to assign values to variables. If the subspace is partitioned into two, the branch of this type is called dichotomic, otherwise polytomic branching.

3.1.3 Heuristics

An heuristic is a technique used to speed up finding a satisfactory solution to a problem when an exhaustive search is not possible. Heuristics result in simplifying a complex problem and produce a good solution, [15]. Thus, heuristics now have two very important purposes: to provide good solutions to problems for which current algorithms are incapable of proving optimality within reasonable time and to help improve the fathoming efficiency of an exact algorithm.

Local search is introduced into the research on heuristics, whereby one obtains a feasible solution and then iteratively improves that solution by moving locally. Constructive algorithms attempt simple strategies to pick the best single move without any look ahead to consider the impact of both rounding up and rounding down a given variable in an linear programming solution. Other more complicated moves are also proposed such as the Lin-Kerningham algorithm for the traveling salesman problem. Many of these techniques are based on analogies from the natural world, such as properties of materials, natural selection, neural processing, or properties of learning found in animals.

In branch-and-bound algorithms, an heuristic is used to both construct the initial feasible solution and branch in the search tree. To generate a good initial feasible solution, heuristics such as Simulated Annealing, Genetic Algorithms, and Tabu Search are most popular.

Simulated annealing algorithms are inspired by the properties from statistical mechanics where an annealing process requires the slow cooling of metals to improve their strength. The analogy is that one slowly converges to a feasible solution by inserting a randomization component. Similarly, genetic or evolutionary algorithms are based on properties of natural mutation. As for the analogy, every feasible solution to the combinatorial optimization problem is equivalent to a DNA string and each such string is given a value. Two individuals mate according to their objective function value and they create a new solution whose attributes are a combination of attributes of each parent. However, an offspring can contain a mutation as well, i.e., an attribute that neither parent has.

In the search tree, a heuristic selects branches more likely to produce solutions than other branches. For example, when the gap between the upper and lower bounds become smaller than a certain threshold, branching is stopped and this node is discarded at once. In this case, the solution is good enough for practical purposes and can reduce computation effort. An heuristic of this type is applicable in the case where the cost function is the result of statistical estimations; thus, it is not known very precisely or lies within a range of values with a certain probability.

3.2 Scrap Optimization Model

Most of steel production factories have their own scrap optimization software, and some call them a scrap optimizer. The complexity of scrap optimizers varies from plant to plant. The simplest one is a couple of recipes based on experience, such as the one being used by ArcelorMittal, Contreceour. Engineers construct scrap recipes for different steel qualities based on past experience. The recipes depend on production requirements and scrap steel properties, such as chemical composition, yield and volume and take into account scrap market prices and availability. When building the recipe, engineers also consider logistics limitations to avoid problems such as delaying production and electrode breakage.

A more complex case is to construct a theoretical model based on known chemical and physical principles like thermodynamics, mechanics, electrical, fluid dynamics and reaction kinetics. MacRosty and Swartz [9] presented a detailed model of the melting process and chemical reactions in an EAF based on fundamental principles and included a degree of empiricism where the real mechanisms were difficult to model or information was not sufficient. Unlike considering the melting process in a model, other factors, like electrode consumption, oxygen or energy, were introduced into the model by Snell [14] to optimize the total cost.

Scrap optimization can be further complicated by using "dynamic" scrap recipes. The parameters in the optimizer, such as cost, chemical composition and density, are updated on a regular basis. Most scrap optimization models are dynamic to some extent.

An alternative is to build an empirical model based on statistical evaluation of process measurements from normal production or experimental tests. In theoretical models, it is key to know the properties of raw materials and reaction mechanisms since any deviations from assumed properties result in computational bias. Compared with theoretical models, empirical models are more advantageous to avoid large deviations because the models are based on relative statistical relationships between measured variables rather than physical properties. Sandberg [10] made a multivariate statistical model, which evaluated how production data that were stored in the process logging system could be used to improve operating practices and optimize energy and scrap utilization. This model predicted the final chemical analysis of the steel, energy consumption, metallic yield and some estimations of scrap properties. However, if the raw material quality fluctuated, the ability to model the process was limited and systematic variations made the model unreliable.

The chemical composition of scrap is highly uncertain which can cause a scrap mix to fail to satisfy the composition requirements of the final product. Rang and Lahdelma [19] developed a fuzzy, constrained linear programming model and represented uncertainty based on fuzzy set theory. Thus, failure risk can be hedged by the proper combination of the aspiration levels and confidence factors for representing the fuzzy numbers.

In recent times, artificial intelligence techniques has been employed to optimize scrap cost, such as artificial neural networks [6] [18], and an ant colony algorithm [12]. A neural network model is often developed to control the EAF

and to optimize material cost and melting cost. Wilson [6] developed NEURAL's ScrapMaster system to minimize the cost of raw materials. This scrap optimization system included the ability to account for variability in scrap properties and the ability to provide on-line updates of scrap property estimates based on previous melt-in chemistry measurements. Liu [12] gave an effective model for the problem of minimizing the cost of raw material with high uncertainty. In his work, he mainly considered the alloying materials and based his model on the properties of alloying elements, harmful elements and corporate standards. An heuristic algorithm, ant colony algorithm, was proposed. Results showed that the proposed algorithm performed better than linear programming solvers with regard to solution quality and comprehensive performances.

3.3 Scrap Distribution Model

In addition to scrap cost optimization, there is usually a process optimization system for scrap supply. The objective of the process optimization system is to increase productivity. Bernatzki [8] developed a scrap optimization model by considering both the scrap transportation system and the EAF. However, in his model, dilution with scrap was used to reduce excessive temperature produced by blowing oxygen onto hot metal. A scrap dispatcher was considered when modelling the whole system. So, scrap supply capacity was also treated as a constraint. However, only a single-train model was developed, whereas a multi-train model was desirable when considering a general global planning situation. Branch and bound rules were widely used in the scrap cost optimization in Bernatzki's work [8]; he implemented a mixed integer programming (MIP) based branch and bound

algorithm. Techniques based on polyhedral optimization were employed to improve the model and reduce solution time. These techniques included variable fixing and eliminating, coefficient reduction, cutting planes and priority order.

Variable fixing is a process where the values of variables can be easily determined in advance and are replaced by constant values. In Bernatzki's model, all the railcars containing more than k tons of scrap were never needed. Bounding rules were also made tighter with coefficient reduction. For example, the maximum total amount of scrap on a single track was reduced from 600 to 400 tons so that some non-integer solutions were infeasible.

In the combinatorial structure of a problem, if two or more variables are equivalent, one of them can be reserved and others are deleted. This technique is called variable elimination. In Bernatzki's scrap distribution model, some cars were considered equivalent and the effective number of railcars were reduced.

Bucket loading is a simple task although it has now become essential to save time in charging a furnace. Not much literature was found about this issue. A project at the University of Pretoria conducted similar research in this field [7]. They established a standard operating procedure based on linear programming regarding the loading of scrap into buckets and also made simulation on the decision whether to allocate additional buckets to specific melt lines. They used the commercial software, Lingo, to solve the model.

Management Science Associates Inc. developed a scrap optimizer, MSA's Blending Optimization Software Suit and Scrap Yard Systems, including scrap

supply and bucket layer loading optimization [2]. Their model increased consistency in the loaded materials and reduced bucket loading times. Finally, the system showed the location of under crane materials and bucket loading by layers.

CHAPTER 4
Scrap Optimization Model in an Electric Arc Furnace

The main focus of this chapter is the modelling of the scrap cost optimization problem. The scrap optimization problem is a combinatorial NP-hard problem. Therefore, branch-and-bound rules are used to reduce the calculation workload. One of the goals in developing a numerical model of scrap optimization is to generate an optimal dynamical production recipe based on dynamic scrap prices and availability. To this end, a substantial effort was made to mathematically model the cost and all the constraints involved in this problem. Since the constraints are linear, this model was based on linear programming. Of particular importance are those constraints related to the choice of branch and bound rules.

There are two main challenges in scrap charge cost optimization:

- The various scrap steels are generally classified into different categories; for example, there are 14 categories at ArcelorMittal, Conctrecoeur. The materials included in a category can be very heterogeneous; in other words, material properties inside a category can deviate significantly, sometimes more than between categories. As a consequence, chemical composition analysis for each type of scrap is very difficult, and that for mixed combinations of scrap is not easy as well.

- The selection of the increment for each class of scrap is a trade-off between computational complexity and cost optimization. A small increment increases the calculation exponentially, while a large one deteriorates cost optimization. Therefore, decisions have to be made depending on the production requirement.

In Section 4.1, a mathematical model is presented along with a discussion on the objective function and constraints. Chemical elements in the liquid steel and scrap costs are estimated in the model. In order to obtain the desired quality and quantity, each melt bath unit of steel has either its own material constraints or constraints set by the Electric Arc Furnace such as the capacity of the EAF. Section 4.2 is devoted to determining an approach to solve this problem by demonstrating how branch and bound rules are used. In this section, a lot of effort is made to choose a selection strategy as well as branching and bounding rules, and the performance of each rule is compared. Finally, Section 4.3 deals with techniques and heuristics application in the model in order to increase the computing speed of the optimization algorithm. This section also discusses the tradeoff between the performance of the algorithm and the optimal solution.

4.1 Mathematical Model Setup

This section details the model that mathematically represents the scrap cost optimization problem. We start by stating the objective function in Subsection 4.1.1 and follow by constraint analysis, including raw material constraints in Subsection 4.1.2, EAF capacity in Subsection 4.1.3, and inventory and scrap availability in Subsection 4.1.3.

4.1.1 Scrap Cost Model

In the model, our objective function is the cost function of demanded scrap steel. One point which should be highlighted is the consideration of yield in the cost function. Besides, this cost is calculated per tonne. The cost function is expressed as follows:

$$Cost = \frac{1}{W} \sum \frac{P_i * Q_i}{Y_i}$$

where:

W: target tap weight of liquid steel;

P_i: the price of scrap i;

Q_i: the quantity of scrap i;

Y_i: the yield of the scrap, where each type of scrap has a yield of liquid steel per tonne of scrap.

When modelling cost, the price and amount of scrap are naturally treated as part of the cost, but the yield is also considered as a parameter, which reflects the connection between scrap and liquid steel. The yield is an estimated property of scrap by scrap providers, which can be understood as the pureness of the scrap, and is highly relevant to the target tap weight. Mathematically speaking, yield is roughly the percentage of liquid steel over scrap steel. The yield value determines the total amount of scrap used in production. The total price of scrap is equal to the product of amount of each scrap type and its corresponding price.

In our project, hot metal left after tapping liquid steel from the EAF is ignored due to the small and varying amount although many optimization models

in other plants consider this factor. This amount of liquid steel could influence the chemical composition of subsequent tapped liquid steel. However, this impact is very limited.

4.1.2 Raw Materials

Target liquid steel is produced by melting scrap steel in the EAF. Here, we ignore the uncertainty of losses due to burning of chemical elements during the melting process. However, properties of the liquid steel are highly influenced by the input scrap steel. Not only does a charge optimization model usually have constraints for chemical balance of the steel, but also scrap chemical properties are also constrained. The chemical balance constraints refer to the sum in the final liquid steel of various chemical elements in the different scrap steels.

Chemical Element Content

ArcelorMittal produces steel by casting about 20 melting bath units per day. Four types of steel are produced that are classified by maximum copper content. Copper content is the most significant chemical element that we should consider in our model. Table 4-1 gives the steel grades and copper content.

Table 4–1: Steel types

Steel type	Copper Content
♯ 1	0.50-0.80%
♯ 2	0.35-0.40%
♯ 3	0.30%
♯ 4	0.25%

Besides copper, chromium, sulfur and tin are also the chemical elements determining the quality of liquid steel. Because a few types of scrap contain a high

percentage of these chemical elements, they become more significant only when these kinds of scrap are used as raw material. The final content of chromium and sulfur should be under 0.2%. Some types of skulls contain a high percentage of chromium or sulfur, and these skulls should be limited in each heat in order to avoid exceeding the upper bound of chromium and sulfur content. Scrap steels, such as tin cans and some types of external skulls, contain a high percentage of tin, and if they are used as raw material, the content of tin becomes significant. In most cases, there is an upper bound for chemical element content in a recipe.

The content of chemical elements in steel is estimated by the following general equation

$$E = \frac{\sum Q_i * C_i}{\sum Q_i}$$

where:

E represents chemical element content in the final combination of scrap steels;

C_i is the content of chemical element in scrap i.

Mathematically speaking, this constraint can be represented as

$$E \leq E_{max}$$

where E_{max} is the maximum chemical element content in tapped liquid steel.

Chemical Properties

The reason why we consider scrap chemical properties as constraints in the model is that these properties can influence the amount of scrap used in each heat. We consider two aspects of scrap chemical properties. Some scrap has a

specific function when they are melted in the EAF. This is because of the chemical structure of the scrap. In the model, we consider the difficulty of melting scrap steel. Different types of scrap are assigned lower or upper bounds for the amount of steel to be in a recipe. Some examples illustrate this type of constraint in the following section. In general, these constraints are mathematically interpreted as

$$q_i \leq U$$

or

$$q_i \geq L$$

First of all, there should be some minimum amounts of ♯1 scrap steel per heat because the use of this type of scrap can help melt other scrap steels faster. Secondly, in order to melt tire wire and turnings, a lot of energy is required. So, an upper bound per heat is assigned to these two scrap steels. Thirdly, skulls are difficult to melt; so, the total amount of skulls, including external and internal skulls, has an upper limit. Table 4-2 shows these lower or upper bounds for different scrap steels.

Table 4–2: Scrap steels with quantity limitations

Scrap type	Upper(U) or Lower(L) limit (lbs)
♯ 1	3000 (L)
Tire and Turnings	5000 (U)
Skulls	2500 (U)

Other Constraints

Another very important constraint is the number of scrap steels used in each heat. In the scrap hall, two tracks of railcars are assembled with a maximum of

12 railcars, 6 on each track. The amount of steel in the railcars produces 6 to 8 heats, and then, the sets of railcars are replaced with another 2 sets of railcars. To guarantee normal production, the number of types of scrap should have a limit due to the difficulty of loading railcars with many types of scrap and replacing empty railcars in sufficient time to be able to load buckets for the EAF. In fact, a maximum number of scrap steels per heat is given by the experience of operators. At ArcelorMittal, this upper bound is usually eight.

4.1.3 EAF Capacity

As the main facility for producing steel, the EAF constricts steel production. The constraints of the EAF are generated from two main sources: production capacity and the layering structure of scrap steels in the bucket.

An EAF has limited production capacity because of the targeted weight of scrap per heat and furnace volume. The targeted weight of liquid steel constrains the total amount of scrap. Some types of scrap used in production have low density; as a consequence, they can easily occupy a lot of room in the furnace. The targeted amount of liquid steel to be produced, the scrap density and the volume of the EAF constrains the total amount of scrap that can be put into an EAF. In real operations, scrap is loaded into the EAF in two or three stages. After melting the first charge, the power of the furnace is turned off and another bucket of scrap is put into the furnace. Then, the power is turned on and melting resumed. In the recipe, the total amount of melted scrap should meet the target weight. In some cases, the sum of two charges of scrap cannot produce the required amount of liquid steel, and a third charge is needed to produce enough liquid steel. The

amount of this charge depends on the specific case and is decided by an operator in real time.

Scrap volume is another significant factor determined by the optimization model. To guarantee that the steel production is as smooth as possible, each charge should have a maximum volume to avoid difficulty in closing the furnace. The EAF used at ArcelorMittal has a maximum volume of about 5000 ft^3. Table 4-3 shows an estimation of density for each type of scrap. An estimation of volume for each scrap is given by the following equation

$$Volume_i = \frac{Mass_i}{Density_i}$$

which is mathematically interpreted as a constraint as follows

$$\sum Volume_i \leq 5000$$

Moreover, some types of scrap steel are composed of large pieces, such as skulls. In order to reduce any difficulty during charging, we need to limit the amount of certain types of scrap per heat, such as skulls.

In order to prevent electrodes from breaking and to make scrap loading easy, scrap is loaded into the EAF in layers. Scrap with low density and soft physical properties is put into the top and bottom layer. The thickness of these layers should guarantee that electrodes do not penetrate the top layer of scrap and touch harder scrap steel; this generates a lower bound for the amount of shred. In ArcelorMittal, this minimum amount of shred per heat is 4000 lbs.

Table 4–3: Scrap density

Scrap Type	Density(lbs/ft^3)
DRI	100
Busheling	43.3
Plate and Structural	42.9
Shred	69.5
♯ 1	34.5
♯ 2	20.4
Tire wire	75
Turnings	102.7
Tin Cans	28.3
Fonte	150
External Skulls	200
Internal Skulls	200
Billets	250
Rolled Revert	175

4.1.4 Inventory and Scrap Availability

There are about 30 types of scrap with various prices and properties in the inventory at ArcelorMittal. However, they are not always available every day. Therefore, the consumption of scrap is restricted by the availability of inventory.

This constraint can be interpreted in two ways. Some scrap has low inventory, which is used in a small amount per heat. So, the quantity of this type of scrap has an upper bound. This constraint can be mathematically represented as:

$$q_i \leq Max_i$$

where Max_i is the upper bound of the amount of scrap used for production.

On the other hand, some scrap is expensive but is readily available. Thus, a minimum amount of these types of scrap steels are used per heat. This constraint

can be mathematically represented as:

$$q_i \geq Min_i$$

where Min_i is the lower bound of the amount of scrap used for production.

To summarize Section 4.1, the scrap charge optimization problem for minimizing raw material costs is represented as follows

Minimize

$$\frac{1}{W} \sum_{i=1}^{n} \frac{P_i * Q_i}{Y_i}$$

Subject to

$$\frac{\sum Q_i * C_i}{\sum Q_i} \leq E_{max}$$

$$\sum_{i=1}^{n} \frac{Q_i}{D_i} \leq 5000$$

$$Q_i \leq U$$

$$Q_i \geq L$$

$$n \leq 8$$

The mathematical model shows that optimization can be done with linear programming. Also, all the variables are discrete numbers so modelling is done with integer programming. This combinatorial problem has a large number of feasible solutions, which takes a lot of computational effort. This thesis presents an efficient branch-and-bound algorithm used to solve the problem. Section 4.2 illustrates the design of this algorithm.

4.2 Solving the Linear Programming Model

This section discusses how to use a branch-and-bound approach in our optimization model presented in Section 4.1. A major issue in applying branch-and-bound in our scrap optimization model is that the solution space is quite large, i.e., an almost infinite number of subdivisions may be checked to reach the optimal solution. We have 14 types of scrap as input and even more if we consider subtypes of scrap. In addition, the model can be divided into many solution space subdivisions depending on the smallest scrap increment being considered. The challenge here is to identify the combination of scrap inputs and properly select the branching technique and bounding rules to reduce the calculation load. Towards this end, we use branching rules by connecting the possible increment in terms of the amount of scrap with the capacity of the crane, and by setting bounding rules at each node with the strongest constraint.

We shall now present a formal statement of a prototype branch-and-bound algorithm for the scrap optimization problem. The words in italic letters constitute the critical operations of the algorithm and are discussed in subsequent subsections. The notations used in this part are shown as follows.

Notation: P List of subproblems;

k Iteration number;

M Increment;

S^k Subset corresponding to iteration k;

u^k Upper bound obtained at iteration k;

l^k Lower bound obtained at iteration k;

X^k Feasible solution to subproblem k;

U Upper bound on the global optimum;

L Lower bound on the global optimum;

X^* Candidate globally optimal solution.

The algorithm

Initialization: Process the problem by setting the initial data (S^0, U^0, L^0). Add the problem min[f(X)]. Set k=0.

Iteration k:

Step k.1: If $P = 0$, terminate; otherwise, select a subproblem k from the list P in the set of subproblems. The amount of scrap of one type in each combination is different from one another by a multiple of increment M.

Step k.2: *Bound* the upper and lower solution of subproblem k. Determine a feasible solution X^k from P and compute the upper bound u^k and lower bound l^k.

Step k.2.a: Set $L \leftarrow \min\ l^k$.

Step k.2.b: If $u^k < U$, then $X^* \leftarrow X^k$ and $U \leftarrow u^k$.

Step k.2.c: Investigate the subproblem. If $u^k \geq U$, then goto Step K.2 and select another subproblem.

Step k.3: *Branch*, partitioning S^k into S^{k1} and S^{k2} by adding one type of scrap to the current problem. Set $k \leftarrow k + 1$ and goto **Step k.1**. Then change the amount of one type of scrap by adding an increment of scrap from minimum to maximum.

The above statements are also shown in Figure 4-1. The list of subproblems in Level 1(Scrap 1) is partitioned by adding Scrap 2 and we get another level

of subproblems as shown in Level 2 (Scrap 2). The selection strategy is also illustrated in Figure 4-1. In each set of subproblem branched from the upper level, each combination is selected by adding a multiple of increment of scrap for each type of scrap. In the level of Scrap 2, **Step k.2.c** is shown. For example, the combination (2000, 4000) in Figure 4-1 is bounded and stops branching in the next step.

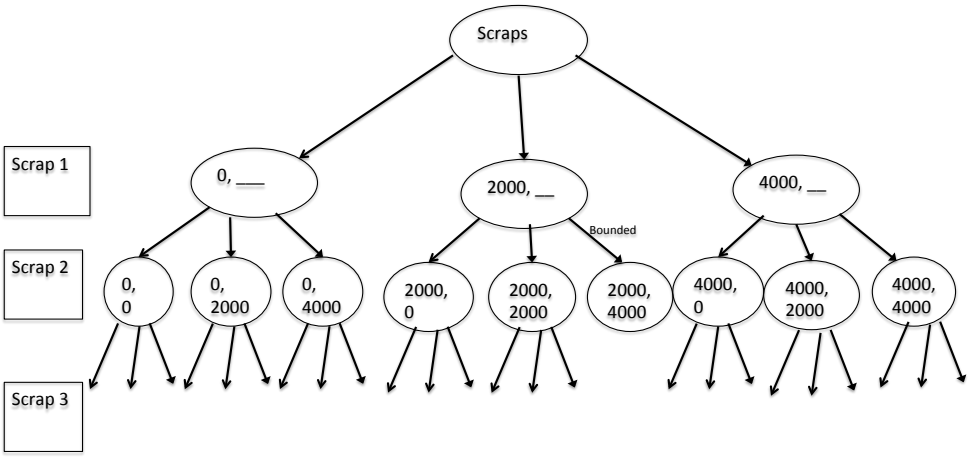

Figure 4–1: Illustration of the Branch-and-Bound rules

4.2.1 Initial Data

Although not explicitly mentioned often, another key issue in the solution of a large combinatorial optimization problem by branch-and-bound rules is the construction of a good initial feasible solution. In this problem, as the objective function, the total cost of scrap is considered as incumbent, which is obtained based on the formula

$$Cost = \frac{1}{W} \sum \frac{P_i * Q_i}{Y_i}$$

The cost of each scrap is calculated by setting $P_i = W_i$. Then, the scrap with the highest cost is considered as the initial objective scrap, whose cost is selected as an element in the initial set U^0. Other elements in the initial set are input data, such as the chemical element contents, and minimum and maximum amount of scrap used per heat.

4.2.2 Selection Strategy

The strategy for selecting the subproblem to further study is usually a tradeoff between keeping the calculation as simple as possible, and obtaining the optimal solution with as low a cost as possible. In **Step k.1**, a subproblem needs to be selected from the list of problems P. Then, this subproblem is considered for bounding and further branching. The selection operation is very critical for a branch-and-bound algorithm. In our model, a combination of best and breadth first search strategy is employed.

The subproblem with a bounding function value smaller than the optimal solution of the problem is explored first. This strategy is called best first search. At the same time, a breadth first search strategy is also used to improve the performance of the algorithm. In the latter strategy, all the nodes at this level of the search tree are processed before exploring any node at a higher level.

4.2.3 Increment

The increment M is a parameter that determines the scrap combination in one level of the search tree, distinguishing different nodes at one level. The choice of increment determines the difficulty of calculation and depends on a few factors.

The first factor is the ease of control for scrap types in the recipe for operators, which gives an upper bound to the increment. If the increment exceeds an upper bound, some issues arise. For example, if only a few types of scrap are used, some parameters, such as the chemical element content, volume and cost, change a lot with an increase of one increment, resulting in possibly missing a potential optimal solution in the range of the increment. Therefore, a large increment makes cost optimization difficult. On the contrary, a small increment makes the algorithm computationally expensive because the search tree is extremely large. For example, if 14 types of scrap are used as an input datum and the increment reduces from 4000 to 2000, the number of computations increased by a factor of up to 2^{14}. Therefore, a lower limit should be taken for the increment. Generally speaking, the smaller the increment is, the lower the optimization of the scrap cost. Here there is a trade-off between difficulty and the optimal solution.

4.2.4 Branching Rules

The branching rules in a branch-and-bound algorithm can be seen as the subdivision of a part of the search space through the addition of constraints, in our case in the form of assigning a new type of scrap to the problem. The number of types of scrap is determined by the inventory. Often there are 14 types of scrap used to produce steel, but certain types of scrap have many subtypes, which increases the complexity of the problem. In the model we consider each subtype individually. These subtype vary with either certain chemical element content or the price due to different purchasing sources. For example, there are 7 types of

external skulls with different prices and chemical element content. Table 4-4 shows the common subtypes of scrap and their differences.

Table 4–4: Scrap steels with subtypes

Scrap Code	Scrap Type	Price ($)	Cu (%)	Cr (%)	S (%)	Sn (%)
105	Busheling	359	0.12	0.04	0.01	0.008
126	Busheling	310	0.12	0.04	0.01	0.008
305	Busheling	373	0.12	0.04	0.015	0.008
209	External Skulls	196	0.02	0.09	0.08	0.006
109	External Skulls	196	0.02	0.1	0.5	0.02
208	External Skulls	201	0.05	0.06	0.07	0.006
211	External Skulls	196	0.21	0.11	0.002	0.015
508	External Skulls	196	0.05	0.06	0.07	0.006
609	External Skulls	180	0.04	0.04	0.09	0.004
709	External Skulls	150	0.12	0.95	0.006	0.006
204	Shred	401	0.2	0.071	0.02	0.013
304	Shred	390	0.4	0.1	0.02	0.2
104	Shred	413	0.3	0.11	0.04	0.008
404	Shred	401	0.35	0.041	0.04	0.008

At one level of the search tree, subproblems are further partitioned by adding an increment of one type of scrap within the range of the allowed amount of scrap per heat. This branching rule can be controlled by changing the minimum and maximum amount of scrap allowed per heat and the increment.

4.2.5 Bounding Function

As the most key component of any branch-and-bound algorithm, a high quality bounding function can make the algorithm efficient. A good choice of branching and selection of strategies cannot compensate for low quality bounding functions to reduce the amount of computational effort. In our model, a bounding function is selected from all the constraints. Besides, the objective function, cost,

is also considered as a bounding function. A good bounding function should be strong enough and in our case a combination of a couple of bounding functions is also employed to increase the strength of bounding. To decide the strength of each constraint, a simulation is set up.

Simulation

In this simulation, 6 types of scrap are taken into consideration and a set of bounding functions are constructed based on constraints such as cost, volume, weight, copper content and chromium content. In each simulation, one of these constraints is selected as a bounding function using the same algorithm. All of them must be selected one by one and simulation time is used as an indicator to compare the strength of a bounding function. To guarantee the integrity of the simulation, the above mentioned sets of simulations are repeated for 5 trials with different input data and the average time for each bounding rule is used to determine the strength of constraints. Table 4-5 presents simulation results. As seen from Table 4-5, the cost and weight of scrap are shown to be the two strongest bounding rules among all the constraints, which are used in our model. However, we also consider other rules such as copper content even though the strength of this constraint is not as high as cost and weight. The reason for this is stated as follows.

- Our model is used in a very dynamic environment which could include some special scrap steel with high copper content
- Copper is a very important chemical element used to determine the quality of liquid steel

Table 4–5: Bounding rule strength test

Bounding Function	Average Time of Running Algorithm (seconds)
No bounding function	260
Cost	62
Copper Content	110
Chromium Content	233
Volume	124
Weight	70

4.3 Accelerating the Model Solution Process

The detailed description of the optimization model and the branch-and-bound algorithm in Sections 3.1 and 3.2 lays the ground work for finding the optimal solution. Solving scrap optimization problems in a dynamic setting introduces another level of complexity, as the response time becomes a crucial issue when increasing the number of scrap steels. Even though we can eventually find the optimal solution, the above stated algorithm is too computationally expensive to be applied in a dynamic setting. For example, the algorithm spends as long as one hour in order to finish all the branches by considering 10 types of scrap, and even worse, it takes two hours when considering 14 types of scrap. Therefore, some model improvements technologies are introduced to the above model in order to reduce the time of finding an optimal solution. However, the implementation of various techniques with this kind of function in such an environment represents a considerable challenge.

In the following paragraph, some typical techniques developed for our dynamic scrap cost optimization model are described in detail. In fact, there are two main classes: the first one includes techniques that improve branch rules and introduces

extra constraints such as priority order and predicting bounding, and the second one is the use of heuristics.

Priority Order

One of the most efficient methods for improving solution performance is to assign a branching priority for variables. Variables refer to scrap in the model. To decide which variable should have a higher priority, results of the simulation in subsection 3.2.4 are used as a reference. Since results show that the cost is the strongest constraint, it is considered as the priority rule for each variable. Scrap steels with higher priorities, namely higher costs, are selected and branched in the branch-and-bound tree before scrap with lower priorities.

The consideration of scrap in the dynamic problem is not done randomly, but rather in a sequential fashion. When designing the algorithm, the selected scrap steels, are firstly ranked based on their cost, and the most expensive scrap is considered as the first problem set from which a feasible solution is generated. Then, as a branching rule, another type of scrap is added into the original problem set, thus creating partitions of sets of subproblems. This added type of scrap is the second most expensive. Then this procedure is repeated until all the selected types of scrap are used. The flowchart of the branch-and-bound algorithm in Figure 4-1 is modified with this technique as shown in Figure 4-2.

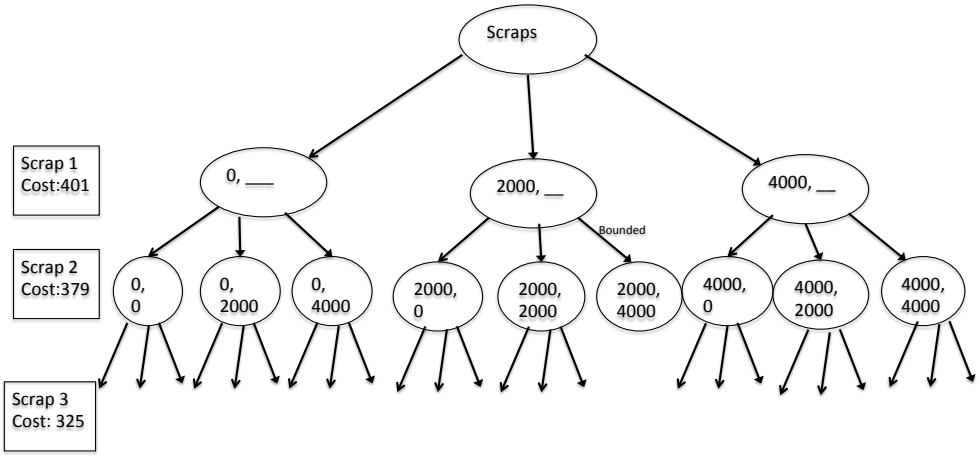

Figure 4–2: Illustration of the Branch-and-Bound rules with priority order of cost.

Consequently, this technology can improve the performance of our model in general. The biggest advantage of this technique is that the scrap cost, a bounding function in the algorithm, can be updated with a small cost in the early stage of searching. More branches are bounded early by this bounding function such that the time is saved by avoiding going through these branches. However, in some cases, high computational effort is still required to finish the entire algorithm because scrap steels with low cost are considered at the end, and become less sensitive to the cost bounding rules. As a result, the cost bounding rule originally considered as the strongest, was considered weakest later. The performance greatly depends on the input data. Therefore, the model is not stable enough to get the optimal solution for a long-term view.

Predicting Bounding. Another efficient method for improving solution performance is to reduce the feasible solutions by adding some constraints so as

to save time. In the branch-and-bound search tree, each node is explored by a rule to decide whether to branch further. This rule, as an additional constraint, predicts the cost of most possible feasible solution and checks whether this solution is better than the incumbent. If it is better, the algorithm continues branching at this node. Otherwise, this node is discarded without further branching.

In the application, the weight of scrap is chosen as the variable to make an additional constraint, and the upper bound is the target input scrap weight. Compared with other parameters, weight is the most significant and very effective. Cost becomes weaker in the late stages of branching when using the priority order technique, whereas weight remains the same during the entire process because the increment is preset at the beginning of the search. Volume is a strong constraint only for scrap with a low density such as ♯2; it cannot produce a constant prediction effect. Other parameters, such as copper and chromium content, are not strong as shown in the simulation in Subsection 4.2.4. The additional constraint can be mathematically represented as follows

$$Q_{current} + \sum_{i=m}^{n} Q_i^{max} \leq Q_{target}$$

where

m is the number of used scrap steels;

n is the number of all types of scrap steels;

m-n is the number of unused scrap steels;

$Q_{current}$ is the current amount of used scrap, that is, $Q_{current} = \sum_{i=1}^{m} Q_i$;

Q_i^{max} is the maximum amount of scrap used in each heat;

Q_{target} is the target input scrap weight.

In the model, a branch is discarded without further partitioning and bounding if the above constraint cannot be satisfied, which means that the current scrap amount is so low that it cannot reach the required target scrap weight even though the maximum amount of the scrap of all the unused scrap steel is added. This situation is predicted at each node. Thus, some unreasonable subproblems are discarded based on the prediction. This logic is shown in Figure 4-3.

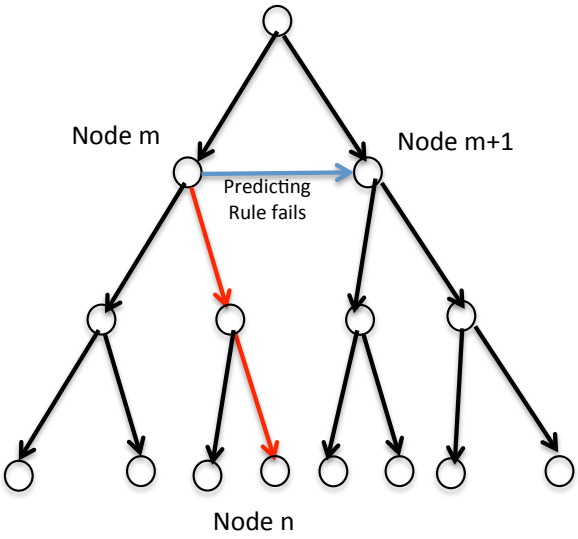

Figure 4–3: Predicting rule applied in the search tree

As seen from the above figure, the predicting constraint is applied at **Node m**. Then, the total weight is calculated along the branch in red with the largest weight of scrap until the last node, **Node n**, which is located at the bottom level of the search tree. If the predicting rule fails, **Node m** is discarded and **Node m+1** is the next one checked by this rule.

This technique does not have any chance of discarding an optimal solution, but it can improve the algorithm to a limited degree. This is because the number of unreasonable subproblems is small in comparison to the possible global feasible solutions. In most cases, this additional constraint stays in an idle condition, in which the node cannot be discarded. Therefore, the performance of this technique is less significant than that of priority order.

Heuristics

The scrap cost optimization problem is dynamic because the company continuously purchases new types of scrap from many sources with changing price and the scrap inventory is updated monthly, weekly, and even daily. A stable and efficient model is required to guarantee a normal production rate.

As a rule of thumb, an heuristic is learnt by experience and cannot guarantee that an optimal solution is ever found, but it can improve the efficiency or effectiveness of our optimization algorithm. Many heuristics are used in the model and the possibility of finding an optimal solution is also shown to be very high. The introduction of heuristics makes our model more stable, which satisfies the requirements of the dynamic industrial production problem even though the price and availability of scrap changes with a high frequency during long term production. Note that the adverse effects associated with the use of heuristics are that the global optimal solution cannot be found all the time.

The first type of heuristic introduced to the model is neighbourhood search heuristics. Such heuristics are proposed as a means to effectively and efficiently tackling this dynamic problem and optimizing the scrap input. This result

is achieved through a powerful neighbourhood structure based on choice of increment. The logic of such heuristics is shown as follows:

Figure 4–4: Heuristics logic

As seen from the above logic, a neighbourhood structure is constructed by resetting the increment in the neighbourhood of the optimal solution obtained from the first running of the algorithm, in which the increment is so high that an optimal solution can be quickly obtained. Then, a subset, as a neighbourhood structure, is selected from the neighbourhood of the optimal solution obtained from the first run. The range of this neighbourhood is either the new and smaller increment or the maximum amount of the selected scrap. Then, the increment M is reset to a smaller value and the algorithm is rerun for that subset. Another optimal solution is obtained from the second time run, which could have either a smaller cost than the first solution or at least equal to that. As a consequence, the algorithm is faster than that without this type of heuristic even though the branch-and-bound algorithm is run twice. According to the requirement for the time efficiency and optimal solution, the algorithm can be run a couple of times.

Another type of heuristic is used in the optimization algorithm. In **Step k.2** of the branch-and-bound algorithm, ninety percent of the upper bound is used as the heuristic upper bound to bound each branch. The result is that each bound is early bounded so that the algorithm can run fast. The optimal solution could

be a local optimal solution instead of a global optimal solution. Compared with the predicting bounding technique, this heuristic shares such common features as discarding some branches to save time, but risks missing the optimal solution. Instead, the predicting bounding technique does not discard the optimal solution. This heuristic has a better performance than predicting bounding technique in terms of algorithm running speed.

To compare the performance of two types of heuristics, many sets of simulations are constructed. In the simulation to control variables, 6 types of scrap steels are employed so that the model can reach optima very fast. 10 sets of simulations are run, each of which uses the same branching and bounding rules and has different requirements for the following aspects: allowed amount of scrap (minimum and maximum), price of scrap, tapped liquid steel weight and constraints. What is more, in each set of simulations, there are three algorithms executed with the same input data: the one without heuristics, the one using the first heuristic, and the one using the second heuristic. Consequently, two indicators for each algorithm are used to compare the effectiveness of two heuristics. The optimal solutions obtained from each algorithm, as the first effectiveness indicator, are recorded, and the costs of the optimal solutions are compared, which is represented in a percentage calculated by dividing the number of simulations with the same optimal solution by the total number of conducted simulations. The times needed to run the algorithms, as the other effectiveness indicator, are recorded. Then, for each type of algorithm, the average time is calculated. Table 4-6 shows the results of the two indicators comparing the effectiveness of the two stated heuristics. As

seen from the Table 4-6, the heuristics significantly improve the performance of the branch-and-bound algorithm with a lower degree of deviation from the global optimal solutions. Because the first heuristic has a higher probability of obtaining the optimal solution, it was used in our scrap optimization model.

Table 4–6: Heuristics performance comparison

Heuristics	Optimal Solution (%)	Performance (seconds)
No heuristics	100%	250
Heuristic 1	96%	82
Heuristic 2	92 %	75

CHAPTER 5
Scrap Car Loading Model

This chapter describes the modelling of the scrap car loading operation, which is part of the scrap distribution system. The scrap distribution system is used to charge the EAF, the process is comprised of four steps. These steps are

- loading cars with scrap in the scrap yard;
- assembling needed cars onto the tracks to be transported into scrap hall;
- loading buckets with scrap from cars in the scrap hall;
- charging EAF with scrap in the buckets.

The scrap car loading operation is the first step and described in this chapter and the other three steps are described in the next chapter.

The assembled track-car system provides the logistics for the whole production system. Any failure to utilize the full capacity of cars carrying scrap is a missed opportunity to produce liquid steel in the EAF, which can directly be translated into cost. The scrap steel transfer from yard to EAF is currently falling short of its performance capacity due to a number of factors. This, along with bucket loading problem, has led to a delay of production and a difficulty in charging the EAF, which should in fact be the constraining factor or bottleneck of the system. To increase the production rate and improve the utilization of the scrap hall, a car loading model has been developed . The general logic is shown in Figure 5-1.

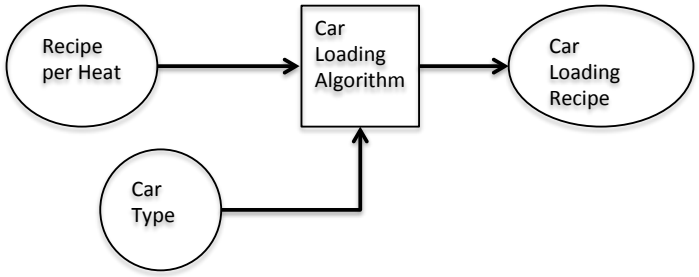

Figure 5–1: General rule for car loading model

The first section of this chapter gives a description of the mathematical model for the problem. Section 5.2 illustrates the structure of the optimization algorithm. In the end, Section 5.3 gives an analysis for the results of the model.

5.1 Mathematical Representation

This section mathematically represents the optimization model, including objective function and constraints. The study in this section is required in order to represent the production problem in the scrap supply to the EAF.

5.1.1 Scrap Transportation Capacity Model

First of all, we should choose an objective function for the optimization model. Our objective function is to maximize the amount of steel produced per transportation cycle. This can be expressed as the number of EAF 'heats' and is mathematically represented as

Maximize

$$H$$

5.1.2 Scrap Constraints

The first constraint we consider comes from scrap properties. The recipe usually contains 6 - 8 types of scrap. Skulls cannot be loaded into cars due to bulky volume and are instead piled next to tracks. Other scrap is loaded into cars in the scrap yard and transported into the scrap hall. A pile of skulls is shown in Figure 5-2.

Figure 5–2: Skull next to cars in the scrap hall.

5.1.3 Car Dimension

Car dimension is another important factor to be considered in the model. Cars used at ArcelorMittal include two types with different dimensions (Table 5-1). Each car is fully charged with scrap and the number of cars for each type of scrap

is determined by the recipe. This constraint is given by

$$H * V_i \leq V_c * C_i$$

where i indicates the scrap used in the recipe;

V_i is the volume of scrap in the recipe per heat;

V_c is the car volume.

Table 5–1: Car dimension

Type of car	Length(ft)	Width(ft)	Height(ft)	Volume(ft^3)
♯ 1	52	7	9	3276
♯ 2	65	6	9	3510

5.1.4 Scrap Hall Capacity

Although the scrap recipe establishes the number of scrap types used in the production, the scrap hall layout limits the maximum of the total number of cars, which is 12. To fully utilize the capacity of scrap hall, the number of heats per car turnover should be maximized. Hence, these 12 cars should be fully filled with scrap, the scrap should fulfill the required heats without the operators needing to fetch scrap from scrap yard due to a lack of certain types of scrap. The scrap hall capacity constraint is represented as

$$\sum C_i = 12$$

where C_i is the number of cars containing scrap i.

Finally, the mathematical model is summarized by

Maximize

$$H$$

subject to

$$H * V_i \leq V_c * C_i$$

$$\sum C_i = 12$$

5.2 Optimization Algorithm

We develop an algorithm by using the above stated optimization model to maximize the number of heats based on the capacity of scrap hall.

As a basic component of the scrap distribution model, a formal statement of the car loading algorithm is presented. The notation is shown as follows.

Notation:

R Set of amount of scrap listed in the recipe per heat;

S Subset of R including external and internal skulls;

G Subset of R excluding external and internal skulls;

V Set of volume of scrap per heat excluding external and internal skulls;

V^H Set of volume of scrap for multiple heats;

V_i Element i in V^H;

V_c The volume of selected car;

V_m The volume of scrap with maximum volume per heat;

H The number of heats;

C The number of cars loaded with scrap;

C_i The number of cars loaded specifically with scrap i;

k Iteration number.

Algorithm

Step 1: Read data from the scrap recipe and assign them to R. Extract skulls from R and assign them to S. Others in R are assigned to G.

Step 2: Initialization. Calculate the volume of scrap in set G and assign them to V. Select the maximum value in G and assign it to V_m. Determine the initial value of H: $H = \frac{V_c}{V_m}$, in which H is an integer and the division is rounded toward negative infinity to the nearest integer. C is initialized with the number of elements in G. V^H is assigned by multiplying H and V.

Step 3: Iteration k:

Step 3.k.1: Check if any element V_i in V^H is smaller than V_c. If $V_i \leq V_c$, update some variables: $H = H + 1$ and $V^H = V^H + V$. Otherwise, $C = C + 1$, $C_i = C_i + 1$ and $V_i = V_i - V_c$.

Step 3.k.2: Check if $C \leq 12$. If $C \leq 12$, go to **Step 3.k.1**. Otherwise, stop iteration, and $H = H$ - 1 and $C_i = C_i - 1$.

Step 4: Output C_i and H. Total amount of external and internal skulls is equal to the product of S and H.

5.2.1 Initialization

In this optimization model, the input data is extracted from a spreadsheet containing scrap properties and the scrap recipe. The following considerations should be noted.

(1) If the scrap recipe includes any type of skulls, either internal or external, the skulls must be assigned to variables other than other types of scrap. This is an interpretation of scrap constraints in Section 5.1.2 for the algorithm.

(2) The initial value of our objective function **Heat** is determined by the scrap with the maximum volume in the scrap recipe so the algorithm should have the ability to find this scrap and calculating the initial value of heat

$$H = \frac{V_c}{V_m}$$

H should be an integer. Thus the division should be rounded toward negative infinity to the nearest integer. The reason for this choice relies on the fact that the scrap with maximum volume can fill the car more easily than others by increasing the number of heat. Therefore, the initial heat is obtained by fully charging one car.

(3) The initial value of the number of cars is equal to the number of scrap steels in the recipe excluding any type of skulls. The idea behind this value is that each type of scrap is put into one car and so the number of cars is the sum of scrap in the recipe excluding skulls.

5.2.2 Iterations

These iterations are based on increasing two variables: the number of heats and the number of cars. The number of heats serves as the rule of repeating iteration, while the number of cars is a stop rule, which means that the iteration continues until 12 cars are fully charged with scrap.

In each iteration, constraints from Section 5.1.3 are used as rules to coordinate the increase of the number of heats and the number of filled cars. Therefore, these two variables are increased in relation to each other. The relationship between them are illustrated in **Step 3**.

An alternative for **Step 3.k.1** is to check the maximum value of scrap volume in V^H and then compare it with the volume of the railcar. If this value is smaller than or equal to the car volume, update some variables: $H = H + 1$ and $V^H = V^H + V$. Otherwise, $C = C + 1$, $C_i = C_i + 1$ and $V_i = V_i - V_c$. In the end, the performance of two approaches is equivalent.

5.3 Result Analysis

Simulation results show that this car loading algorithm is efficient and simple to operate. In most cases, the algorithm takes less than 5 seconds, which meets our expectation. As an example shown in Table 5-1, the so-called car loading recipe, is obtained from the car loading model. From this table, operators can load each type of required scrap into cars with the specified number. Compared with the method presently being used at ArcelorMittal, this model can predict the production and avoid omitting some amounts of scrap during operation, which properly solves a common problem in current production.

The maximum number of heats, H, reflects the extent of utilization of the scrap hall capacity. In the above example, H is 8. Results are compared with their historical production data, showing that about two thirds of results from our optimization model are larger than the number of heats produced in the past and

Table 5–2: Car loading recipe

Scraps	♯ of cars or amount of skulls (lb)
Tire wire	2
Busheling	2
Shred	2
♯ 1	2
♯ 2	4
Internal Skulls	168,000
External Skulls	140,000

five percent are less than the actual production. Therefore, the car loading model is robust and optimizes the steel production process.

CHAPTER 6
Bucket Charging Model

This chapter mainly focuses on development of the bucket charging model. This model is used to represent the scrap charging operation in the scrap hall. First of all, we should understand the structure of the scrap hall and all the operations conducted in it. Figure 6-1 gives the layout of the scrap hall. As shown in the figure, the layout of the scrap hall presents the location of each component described in the charging operation steps. Every fleet of cars enters the hall from the right side and leaves the hall from the left side.

A few challenges confront us in modelling the scrap distribution system:

- The location of each car in the scrap hall is not fixed because the operator can relocate cars by moving cars during the bucket charging operation. It is difficult to mathematically model the time of bucket charging which is an intended objective to be achieved through optimization. So the development of the objective function is challenging;
- Scrap in the bucket should be in layers, which is determined by the properties of the scrap steel and the charging operation of the EAF;
- The capacity of a magnetic crane is difficult to model. Each type of scrap has a different density and the amount of scrap picked up by the crane each time is a variable, which is determined by a few factors, such as the physical

properties of the scrap and the performance of the magnet for each type of steel.

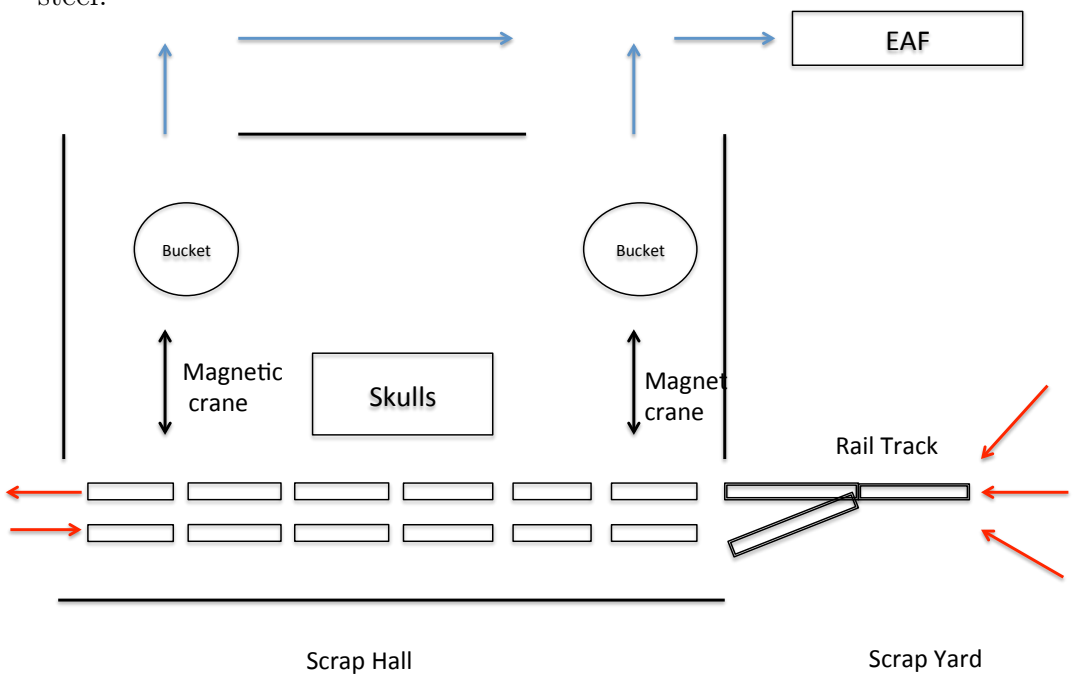

Figure 6–1: Scrap hall layout

To start this chapter properly, let's recall the process of charging the EAF.

- loading cars with scrap in the scrap yard;
- assembling needed cars onto the tracks to be transported into scrap hall;
- loading buckets with scrap from cars in the scrap hall;
- charging EAF with scrap from the buckets.

The bucket charging system is fundamental to the whole production system. To increase production rate and avoid delays in production, an efficient model should be built to estimate the production properties and control production

process. The model developed in this chapter is based on linear programming and branch and bound rules. Before we model the scrap distribution system, the scrap recipe should be generated from the model in Chapter 4, which is the input data for the system. Figure 6-2 shows the logic of the scrap distribution model. As shown in the logic,

- car loading algorithm maximizes the number of heats to be produced in each car turnover through setting the number of cars to be filled by each type of scrap in the recipe. It is illustrated in the previous chapter.
- bucket layer algorithm finds the optimal way to fill each of the two buckets based on the recipe to minimize the EAF loading time. This algorithm is illustrated in Subsection 6.2.
- car layout algorithm determines the optimal alternation of scrap to the cars to minimize the loading time for the two buckets. As part of the bucket charging model, this algorithm is detailed in Subsection 6.3.
- combination algorithm is a bridge that connects the car layout algorithm and bucket layer algorithm. It calculates the total charging time based on car layout and bucket layer algorithm solutions. This, along with an heuristic to improve algorithm execution time, is shown in Section 6.4.

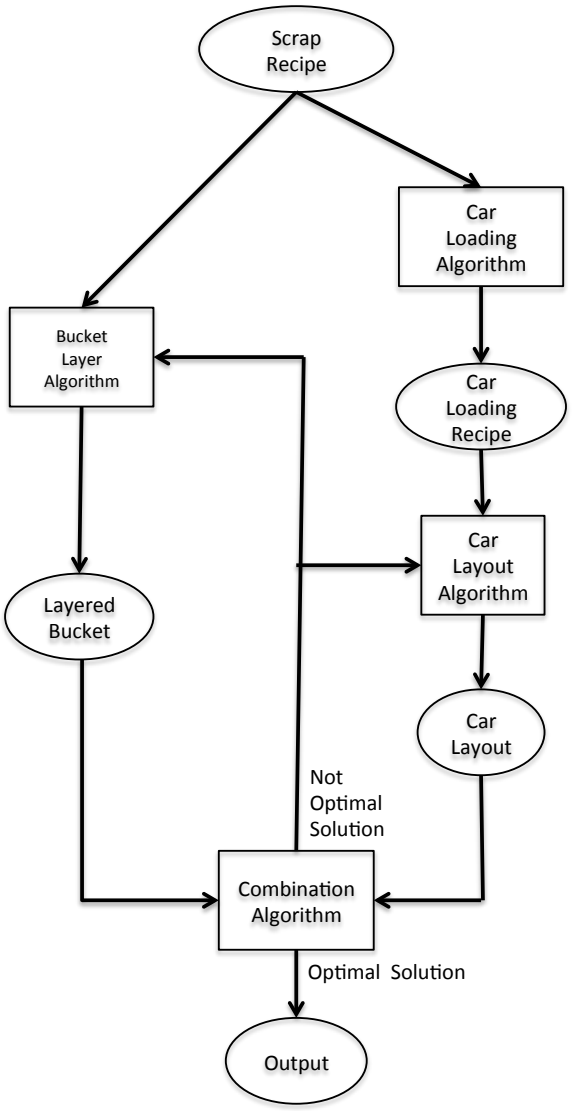

Figure 6–2: The flowchart of bucket layer optimization

6.1 Mathematical Representation

In this section, we will present a mathematical representation of the bucket charging model. The objective of this model is to reduce the idle time of an EAF.

6.1.1 Scrap Charging Time Model

The objective function of the scrap charging optimization model is the time spent charging both buckets to produce one heat of steel. The time model is mathematically represented by

Minimize

$$\sum_{i=1}^{2}\sum_{j=1}^{13}\lambda_{ij}t_{ij}$$

where:

2 refers to the two buckets needed per heat;

13 is the sum of 12 cars and one pile of skulls;

λ_{ij} is a binary parameter indicating whether we use scrap in car j. If λ_{ij} is 0, the scrap in car j is not loaded in the bucket i; if λ_{ij} is 1, the scrap in car j is loaded in the bucket i;

t_{ij} is the time to carry scrap from car j to the target bucket i.

In the model, the time is calculated for loading both buckets, which is represented in the time function. When we load each bucket, the time of charging each layer is accumulated and finally treated as the objective function. The introduction of variable λ models the utilization of each type of scrap per heat. In this project, the time of charging the EAF from buckets is not included in the time model since we assume that the time of charging the EAF does not depend upon the content of the buckets.

To support our assumption, some data about time taken in scrap charging operations are taken from ArcelorMittal, including the time of loading buckets (t_1), the time of charging an EAF (t_2), the time between first and second charge (t_3),

and the time taken to complete one whole heat (t_4). Table 6-1 shows a summary of these time.

Table 6–1: A summary of different time per heat

Class of time	Amount of time (minutes)
t_1	20-25
t_2	2
t_3	15
t_4	60-80

6.1.2 Constraints

The general method for loading a bucket is filling the bucket with scrap taken from a railcar by using a crane. Constraints involved in this operation are described and modelled below.

Car Location

The location of cars in the scrap hall is one constraint in the category of scrap source for the bucket. In order to fill the bucket as quickly as possible, the scrap must be close to the bucket. Nonetheless, buckets are located on opposite sides of the scrap hall, thus two rows of railcars compensate somewhat for this inconvienence.

Scrap

Scrap properties generate constraints for the optimization model. After filling the bucket, operators charge the EAF with its contents. Some constraints of the EAF described in Chapter 4 are used in this model.

Certain soft and thin scrap, such as shred, should be loaded on the top and bottom layers in both buckets in order to avoid breaking electrodes in the EAF

and to protect EAF refractory from scrap when releasing scrap from the bucket into the EAF. This constraint can be represented as follows

$$m_{shred} \geq 2000$$

This constraint is applicable to both buckets.

Skulls are difficult to melt so they should be near the beginning of the melt. In the first bucket, a constraint should be included

$$m_{skulls} = m_{recipe}$$

where m_{recipe} is the amount of skulls used in the recipe, including internal and external skulls. Skulls should be loaded in the middle layer of the bucket.

Crane Capacity

The capacity of a crane for each type of scrap is one important factor influencing the time needed to load both buckets. The crane uses a magnet and the amount of scrap a crane can carry depends on the type of scrap. The capacity of a crane is experimentally determined. Table 6-2 gives an estimated amount of scrap that each crane can hold during each load. A crane can move either horizontally or vertically, from any point to another in the hall.

Bucket Capacity

Bucket capacity limits the charging operation so the scrap volume and weight should not exceed the maximum capacity of bucket. The volume should have an upper limit to avoid difficulty in charging the EAF. The weight of scrap in each bucket has a maximum value because it is impossible to move an over-loaded

Table 6–2: Average weight of scrap per load

Scrap Type	Average weight(lbs)
101	3573
104	6636
105	2638
108	5948
109	7689
110	2942
111	3781
112	3202
123	4052
201	9037
202	7011
205	2585
208	4579
404	6304
504	7415

bucket. These two constraints are given by

$$\sum_{i=1}^{n} \frac{m_i}{d_i} \leq 2500$$

$$\sum_{i=1}^{n} m_i = 120,000$$

where

2500 is the volume of the bucket, the unit is ft^3;

120,000 is the bucket capacity in terms of scrap weight in lbs;

n is the number of scrap steels used in the bucket;

m_i is the weight of scrap i in the bucket;

d_i is the density of scrap i in the bucket.

Again, we summarize the above mathematical model as follows

Minimize

$$\sum_{i=1}^{2}\sum_{j=1}^{13}\lambda_{ij}t_{ij}$$

Subject to

Constraints in the first bucket are

$$m_{shred} \geq 2000$$

$$m_{skulls} = m_{recipe}$$

$$\sum_{i=1}^{n}\frac{m_i}{d_i} \leq 2500$$

$$\sum_{i=1}^{n} m_i = 120,000$$

Constraints in the second bucket are

$$m_{shred} \geq 2000$$

$$\sum_{i=1}^{n} m_i = 120,000$$

6.2 Bucket Layering Algorithm

The objective of the bucket layer model is to generate the feasible solutions for filling the bucket from the scrap recipe. This is one of the three algorithms that together determine the quickest way of charging the EAF. Towards this end,

all the possible layers in the bucket must be found and used as candidates of an optimal solution. The bucket layer model uses a scrap recipe as an input and generates a set of bucket filling combinations (layers of scrap), each of which is a feasible solution and an input to the combination algorithm.

To simplify the model, we apply the bucket layering algorithm only to the first bucket and deduce the filling of the second bucket from what is left to fulfill the scrap recipe.

6.2.1 Algorithm Structure

First of all, we present a formal statement of the algorithm. The following step-by-step algorithm utilizes the branch and bound rules and search tree to execute the model.

Step 1: *Initialization of Variables*

Read data from the scrap recipe. Assign the amount of scrap and volume of scrap to subsets **T** and **V**.

Step 2: *Add New Variable on New Level*

Adding one new level to the search tree means taking one more type of scrap into account. At each level, more branches are created by adding a multiple of the amount M in the next level of scrap. This is the branch rule for this algorithm. This step proceeds until the number of levels **L** reaches the number of scrap types in the recipe.

Step 3: *Test Feasibility*

Each time a new variable is added. Then, calculate the second bucket layer combination. Check the constraints in the mathematical model.

a. If at least one of the constraints is not satisfied, this node is discarded and will not be further partitioned.

b. If all the constraints are fulfilled, proceed to **Step 2**.

Step 4: *Save Candidate Set and Backtrack to Previous Level*

a. When L is equal to the number of scrap types and all the constraints are met, the whole branch of the tree is added to the candidate set F_{layer}.

b. Return to the previous level and explore the branch adjacent to the candidate branch. Return to **Step 2**.

Step 5: *Finish*

The set of candidate F_{layer} is achieved after all the branches of the tree are explored. F_{layer} is used as an input of the combination algorithm.

The search tree of the algorithm is illustrated in Figure 6-3.

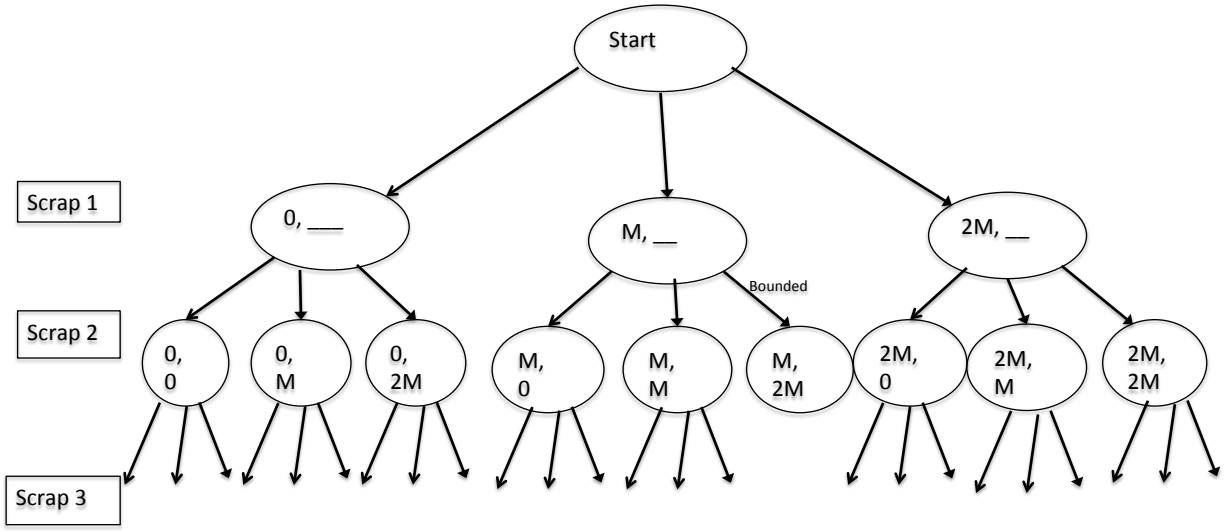

Figure 6–3: Search tree of bucket layer algorithm

As we mentioned at the beginning of this section, branching rules and bounding function are two core components in this combinatorial optimization algorithm. The following paragraphs specifically illustrate the way we model them.

6.2.2 Initial Data

Other than classical branch and bound rules application, this model does not have any incumbent. That is because this algorithm functions as an intermediate step for the global bucket layer optimization model. The output of this model is a set of feasible solutions instead of an optimal solution. Other data such as the scrap type and amount are initialized from the scrap recipe.

6.2.3 Branching Rules

There are two branching rules: scrap type in the scrap recipe and the amount of scrap increasing with a multiple of increment, which is represented as M as indicated in the search tree.

At each node, when one more type of scrap is added to the bucket, the node is partitioned into the next level, which is an interpretation of **Step 2** in algorithm structure. This branching rule is quite similar to the one in the scrap optimization model. Unlike that in scrap optimization model, the number of levels is limited by the number of scrap types in the scrap recipe. This property reduces the computational effort of our algorithm.

As another branching rule, each increment of scrap added in one level generates an additional branch. This increment is not the same as the one in Chapter 4. The size of the increment does influence the quality of the objective function. The feasibility of the model depends on the increment. Because all the

scrap types in the recipe must be charged into the bucket. An increment should be chosen that guarantees that the total amount of scrap in each bucket is 120,000 lbs and the minimum of shred in each bucket is reached. In Chapter 7, criteria for selecting a value for M and the impact of M on the model are detailed with some application cases.

6.2.4 Bounding Function

Bounding functions directly determine algorithm execution speed because the size of search tree can be reduced by choosing strong bounding rules. From the constraints in Section 6.1, the weight and volume of scrap in the first bucket are used as bounding rules. Our consideration is based on the following aspects:

- Weight and volume can be easily modelled and sensitivity to a stable solution determined. After employing branching rules at each node, the weight and volume of each variable are increased by a minimum value resulting from the first branching rule or an increment resulting from the second branching rule. Any situation in which bounding rules are exceeded can be detected and the search stopped. If we choose other constraints as bounding rules, such as shred , it is applied only when the variable is shred.
- Weight and volume is a good combination of bounding function. Density is a critical factor determining the significance of weight and volume. Volume becomes significant for scrap with low density, whereas weight contributes more to scrap with a high density. The advantage of employing these two parameters is to globally increase the strength of bounding rules no matter which scrap is selected.

Besides data initialization, the branch and bounding rule, selection strategy will not be further illustrated because a combination of best and breath first search strategy is employed in this model, which is the same as the scrap optimization model in Chapter 4.

6.2.5 Reduce the Number of Feasible Solutions

The above algorithm produces a good result and the performance is acceptable. Our study tries to employ more techniques to improve the above algorithm. Based on [15] and [17], an investigation of processing tools is illustrated. In this section, a so-called variable fixing is employed to make the model more efficient. The following paragraph illustrates the application of this technique in the model.

After taking a look at the combinatorial structure, the size of the integer programming model can be reduced by replacing some variables by constant values. Consider the first bucket, shred should be placed in the top and bottom layer with a minimum value of 10000 *lbs*; skulls should only be located in the first bucket and we assume the middle layer is reserved for all the skulls. This technique reduces the size of feasible solution generated from layered bucket optimization.

Finally, the algorithm is very efficient and the running time is in an acceptable range. The above method is executed with a recursive algorithm, which is advantageous when the level of search tree is not constant. Nonetheless, a loop function cannot satisfy our need. In this case, the number of levels depends on the number of scrap types in the scrap recipe. The condition under which the recursive procedure calls itself is the branch rules which means there are two conditions for

calling itself. Since there are two good conditions, there is no risk of executing in an infinite loop when running a recursive procedure. The condition of terminating the recursive algorithm is the success of finding a feasible solution, which is stated in **Step 4**.

6.3 Car Layout Algorithm

As shown in Figure 6-1, buckets on the terminals of the scrap hall are loaded with a crane by taking scrap from railcars on the tracks. Time is increased when taking scrap from railcars far from a bucket. To increase the efficiency of loading buckets, the layout of cars in the scrap hall should be properly determined. In other words, the sequence of cars assembled on the tracks should be optimized. Currently, there is no formal consideration to determine car layout at the Contrecoeur facility, the distribution of cars in the scrap hall is decided by the experience of operators.

The layout of cars in the scrap hall is part of the scrap distribution system. Engineers at ArcelorMittal have to decide the layout of the 12 cars according to the daily scrap recipe and operators assemble cars on the track based on the layout of cars. An ideal layout for railcars makes a crane take scrap from a car as close to the target bucket as possible in order to save loading time.

6.3.1 Algorithm Structure

In this model we do not have an objective function and the car layout algorithm is also a preparation step to obtain the global optimal solution of the bucket charging optimization model. The car layout algorithm should have a good performance when efficiently determining a set of possible car layouts which

contains all the scrap in the recipe and this set is called a candidate car layout, which is another input to the combination algorithm which is illustrated in the next section. To reduce the computation load of determining car layout in the scrap hall, branch and bound rules are applied.

The following enumerative algorithm utilizes the branch and bound rules and search tree to solve the problem defined above.

Step 1: *Initialization of Variables*

Read data from car loading recipe. Assign the types of scrap and numbers of cars to subsets **T** and **N**.

Step 2: *Add New Variable on New Level*

Adding one new level to the present level means considering one more car. Each node in one level refers to one type of scrap. This node can be partitioned into as many nodes as the number of types of scrap **T** by adding one car to the next level. This step terminates when the number of levels **L** reaches 12.

Step 3: *Test Feasibility*

a. If the number of cars containing scrap T_i is equal to N_i, this node cannot be branched further with scrap T_i. Then proceed to **Step 2** with adding other types of scrap.

b. If the number of cars containing scrap T_i is smaller than N_i, proceed to **Step 2** with considering all types of scrap.

Step 4: *Save Candidate Set and Backtrack to Previous Level*

a. When L=12, then the whole branch of the tree is assigned to the candidate set F_{cars} if the subset **N** is satisfied. The coloured branches in the tree structure are two elements in F_{cars}.

b. Return to previous level and explore the branch adjacent to the candidate branch. Return to **Step 2**.

Step 5: *Finish*

The set of candidate F_{cars} is achieved after all the branches of the tree are explored.

In order to understand the above statements, a search tree is introduced. Figure 6-4 presents the tree structure of branch and bound algorithm in car layout algorithm. More details about the meaning of the search tree are given in Section 6.3.2 and 6.3.3.

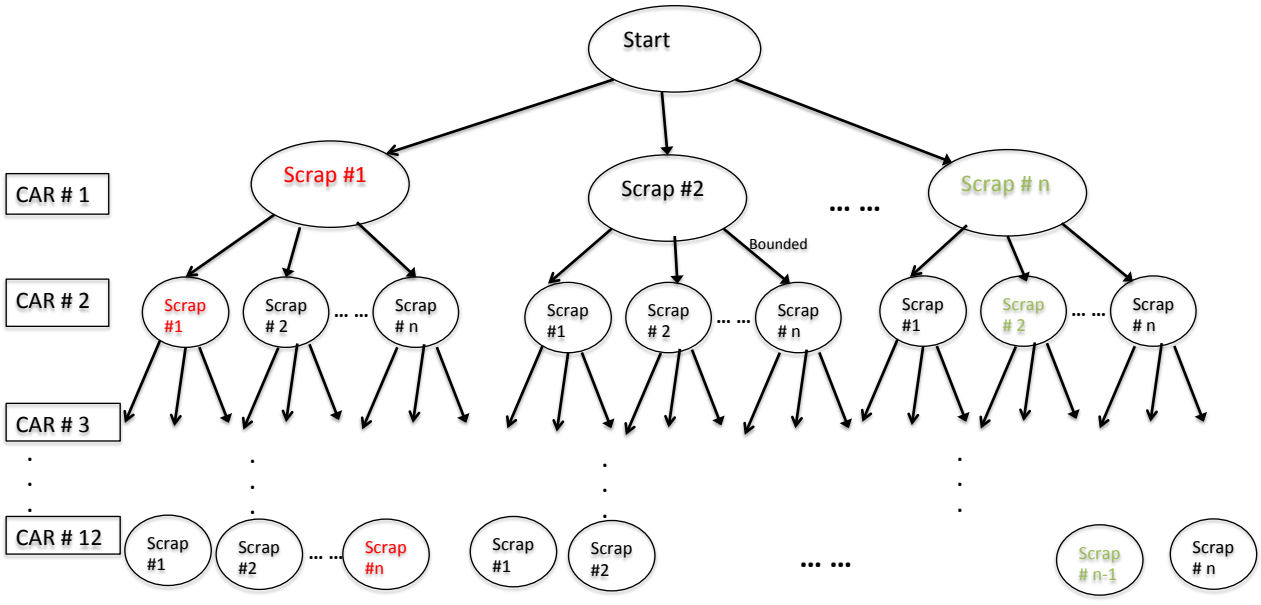

Figure 6–4: Search tree of car layout algorithm

6.3.2 Branching Rules

As repeated in the previously described branch and bound algorithm, it is of importance for the current algorithm to select proper branching rules so as to explore the whole solution tree. Similar to the bucket layer algorithm, the branch rules consist of two types:

(1) add one more car. This branching rule is interpreted by **Step 2** in the above stated enumerative algorithm. In the search tree, each level of tree represents one car. As seen from the tree structure, there are 12 levels because there are 12 cars in the scrap hall. At each level, the number of nodes are equal to the number of types of scrap. The search tree is very large when, say, 5 scrap species are combined. Such cases are typical.

(2) change the type of scrap in each car. Each node is partitioned into subbranches by changing the type of scrap in each car as shown in the search tree. Unlike the increment used in the scrap optimization model and bucket layer algorithm, this horizontal branching rule cannot influence the complexity of the search tree. After the scrap recipe and car loading recipe are determined, its complexity is determined as well.

6.3.3 Bounding Function

Since there is no incumbent in this branch-and-bound algorithm, there is no upper bound that can be updated and bound at each branch during solution exploration. At the same time, the volume of each car is fixed which depends on

the car dimension. Thus, any constraint in Section 6.1 cannot be used to develop a bounding function. Consequently, building a bounding function is challenging.

After investigating the search tree structure, it is an efficient tactic to develop the bounding function based on the number of cars for each type of scrap in the recipe. Even though, the general rule of bounding function is followed by such a parameter. Some challenges still exist, including the approach to model this rule and the strength of bounding function.

The number of cars for each type of scrap is recorded in the car loading recipe and depends on the specific production case. Unlike bounding functions in other optimization models, this parameter is a set of variables. The first difficulty is that such a set of variables is not fixed and case-based. Another difficulty is that only one type of variable is checked in each exploration of a node in the search tree. An algorithm should be developed to determine which variable in the bounding function is qualified to bound this branch. If we consider the whole search tree, this kind of calculation requires much effort. So, this algorithm must be efficient.

To solve these difficulties, an enumerative algorithm is developed, which is illustrated in **Step 3**. This sub-algorithm is run at each node of the search tree and if the variable exceeds the corresponding upper bound, this node is discarded. Specifically speaking, if the number of cars for the currently-investigated scrap, which is accumulated in the upper level of the node being searched, exceeds the number of cars in the car loading recipe, the algorithm stops branching, neither adding more cars nor changing the type of scrap in the car.

As investigation (Table 6-3) shows, this bounding rule is strong enough. The simulation test in the first row runs the algorithm by applying bounding rules and the second one runs the algorithm without bounding rules. The time for running algorithms and the size of the feasible solution for car layouts are two indicators to test the strength of the bounding rule. Results show that the bounding rule efficiently improves the performance of the algorithm and reduces the size of the set of car layouts, which proves that the bounding rule is very strong.

Now we try to explain the principles of this bounding rule. Taking a close look at the search tree, a lot of branches are unreasonable and terminated at the early stage of exploration so much time is saved and the set of car layouts only includes the reasonable candidates; so, the size is significantly reduced. For example, in the search tree all branches on the left of the branch in red are unexplored, which graphically illustrates our analysis.

Table 6–3: Bounding rule verification

Test	Time of running algorithm (s)	Size of feasible solution of car layouts
w	24	8×10^{8}
w/o	325	1.8×10^{13}

6.3.4 Heuristic Rules to Reduce the Number of Feasible Car Layouts

As simulation results show, the size of the car layout candidate set is very large, for example, several billions. Not only does it take time to reach the full set of car layout candidates, but also there is difficulty with the combination algorithm as an input set. Besides the above stated branch and bound rules, we introduce some extra constraints to discard more unfeasible candidates in the set F_{cars}. How

far from the optimal solution the discarded unfeasible candidates are depends on the strength of constraints.

In the ideal situation, bucket layers should be built such that the needed scrap is close to the target bucket. There are two buckets and each type of scrap can be in more than one car. To make sure that a crane can access the closest cars with needed scrap when filling both buckets, the cars in the scrap hall are divided into two groups: the left one, close to Bucket 1; and the right one, close to Bucket 2. Ideally speaking, each group should have the same type of scrap and the same number of cars. However, this strongly depends on the scrap recipe. In the model, the number of cars for one type of scrap is divided by 2 and rounded up to the closest integer. The obtained number is the number of cars in the first group. The rest are assigned to the second group. This technique efficiently reduces the solution set to some millions of elements. The refined cars layout candidate set, F_{cars}, is used as an input to combination algorithm along with the bucket layer candidate set, F_{layer}.

The model is programmed with a recursive procedure which calls itself under the condition of adding one car or that one scrap substitutes another in the same car. The recursive procedure is terminated when a feasible solution meeting all the constraints is found.

6.4 Combination Algorithm

After obtaining the bucket layer candidate set F_{layer} and car layout candidate set F_{cars}, an algorithm should be developed to use them as inputs to determine the optimal solution for the optimization model described in Section 6.1. In this

algorithm, we use trial and error, which means that each bucket layer in F_{layer} and each cars layout in F_{cars} are explored and the time for loading Bucket 1 and Bucket 2 are calculated and the combination that gives the fastest loading time is selected as the optimal solution.

In order to calculate the time of loading both buckets, the time a crane spends fetching scrap from cars is estimated based on observation. The time to take scrap from a car is also not fixed. It differs according to full and partial load size. So, an average time is considered in the model. Figure 6-5 and 6-6 give the time that a crane spends carrying scrap from railcars to the first and second bucket respectively.

Summary of Bucket #1 Loading Time

	1	3	5	7	9	11
Track # 1	**25**	**20**	**30**	**35**	**40**	**50**
Track # 2	**33**	**27**	**35**	**40**	**47**	**65**
	2	4	6	8	10	12

Figure 6–5: Summary of times for filling the first bucket from 12 railcars

Summary of Bucket #2 Loading Time

	1	3	5	7	9	11
Track # 1	**57**	**48**	**37**	**32**	**20**	**25**
Track # 2	**66**	**50**	**40**	**37**	**27**	**33**
	2	4	6	8	10	12

Figure 6–6: Summary of times for filling the second bucket from 12 railcars

6.4.1 Algorithm Structure

The iteration algorithm is illustrated step by step as follows:

Step 1: *Initialization of Variables*

The incumbent T^* is given an initial value.

Step 2: *Select Bucket Layering*

A subset F^i_{layer} is selected from F_{layer}.

Step 3: *Calculate Times of Fetching Scrap From Railcars*

A layer l_i is taken from subset F^i_{layer}. Number of movements required to fetch scrap from cars $n_i = \frac{l_i}{C_i}$, where C_i is the average capacity of a crane carrying scrap i.

Step 4: *Select Cars*

A subset F^i_{cars} is selected from F_{cars}. In F^i_{cars}, among the cars filled with scrap the closest car to the target bucket is selected. The average time from this car to the bucket is t_i and the total time $T_i = t_i * n_i$. Time $\mathbf{T}$ needed to load both buckets is the sum of T_i for layers in both buckets.

Step 5: *Test Optimality*

If $\mathbf{T} \leq T^*$, subsets F^i_{cars} and F^i_{layer} are sets as the current local optimal solution. Proceed to repeat **Step 1-5** until finished exploring all the elements in sets F_{cars} and F_{layer}.

Step 6: *Run All Other Feasible Solutions*

Next bucket layering combination is taken into consideration. **Step 1-5** are repeated.

Step 7: *Finish*

The optimal solutions are subsets F^i_{cars} and F^i_{layer} with lowest T^* among the local optimal solutions.

The computation load of this algorithm directly depends on the size of F_{cars} and F_{layer}. Therefore, some techniques and heuristics are employed to reduce their

size, which were introduced in the previous section. In the combination algorithm, some methods are introduced to increase the speed.

6.4.2 Accelerating an Algorithm with Heuristics

Since the combination algorithm is a bridge connecting the bucket layer algorithm and car layout algorithm, some heuristics can be used in the combination algorithm, which makes them more connected and logical. The two group method in the car layout algorithm is an outstanding approach that both efficiently uses the feasible solution of car layout and reduces computational time. Figure 6-7 illustrates the basic idea of the heuristics we used in the model.

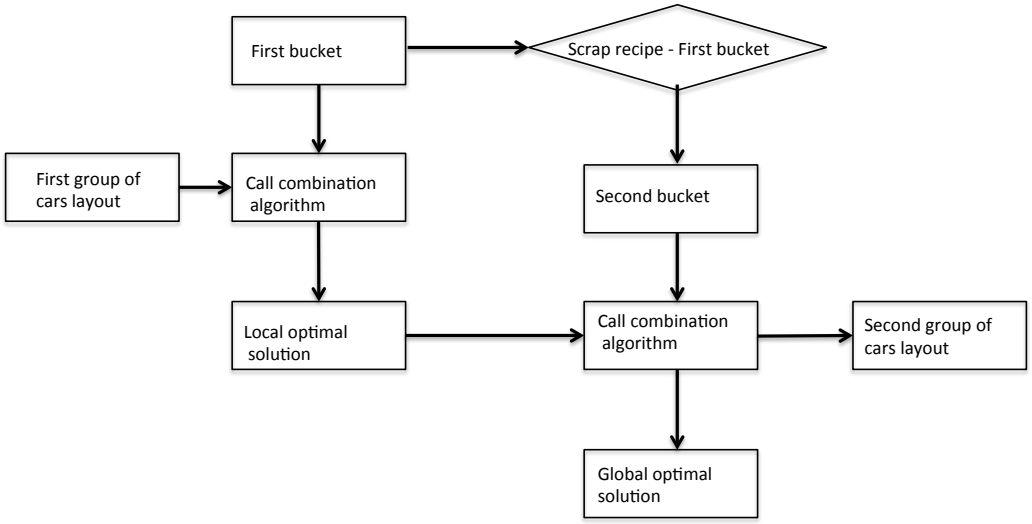

Figure 6–7: The flowchart of heuristics in the bucket optimization model.

As seen from the flowchart, the first bucket is loaded initially by taking scrap from the first group of cars for each element of the layered bucket set and a local optimal solution F^1_{cars} for the car layout in first group is found. In this part of search, an incumbent is developed, which is the time to load the first bucket. This

incumbent is used to check the optimality. For each bucket layer combination in the set of feasible solutions, **Steps 1-5** are conducted layer by layer. An incumbent, the time of loading the bucket, is used to stop completing all the layers when the accumulated time in a certain layer has exceeded the incumbent. This kind of early bounding saves a lot of search time.

Then, we move to fill the other bucket and scrap steels are taken from the second group of cars as shown in the right part of flowchart. Here we have two incumbents: the time of loading the second bucket and the time of loading two buckets. The bounding logic is the same as the first bucket, but in each **Step 5** both incumbents are checked. The idea is as follows: (1) **Steps 1-5** are conducted layer by layer. The first incumbent, the time of loading the second bucket, is used to stop completing all the layers when the accumulated time in a certain layer has exceeded the first the incumbent. (2) If that layer is not discarded, we calculate the second incumbent, the time of loading both buckets and recheck if this incumbent is greater than the current accumulated time. If not, stop exploring layers but continue next to the car layout in the second group. This approach proved to be very efficient.

The reason why we do not use two incumbents is that loading the first bucket has little chance of exceeding the time of loading two buckets. In other words, we have little chance of discarding the feasible solution in the first group. In contrast to the first bucket, loading the second bucket could easily exceed the target time of charging two buckets although the time of only charging the second bucket is acceptable. This idea does make a difference in algorithm speed.

There is a risk of missing the global optimal solution when the obtained local optimal solution F^1_{cars} is not good. This is because another potential feasible solution is better from the global view than the current local optimal solution F^1_{cars}. This risk only arises when only one car of scrap exists in the car loading recipe and is assigned to the first group. This possibility can also be reduced by properly adjusting the loading time from each car position. More specifically, cars in the middle spots take less time than so that these spots have a large chance to locate scrap within one car in the recipe.

After finishing exploring all the feasible solutions in the layered bucket set, the optimal solution is the one with the smallest bucket loading time. We can construct dynamic upper bounds in terms of the general buckets loading time, the first bucket loading time and the second bucket loading time. In the search tree of the first and second buckets, some branches are early bounded efficiently. Therefore, the speed is improved.

6.5 Result Analysis

To check the performance of the bucket charging model, a scenario is set up. Figure 6-8 shows the scrap and car loading recipe, which is the input data of our model. Simulation results are shown in Figure 6-9 and 6-10, including optimized bucket and car layout. Results are good in several aspects: (1) all the constraints are satisfied; (2) the bucket layering is reasonable; (3) the car layout is close to the ideal one.

Code	Type	Recipe	# of Cars
105	Busheling	50,000	3
123	Turnings	50,000	2
209	External Skulls	40,000	320,000
104	Shred	40,000	2
101	#1	30,000	2
205	Tin cans	30,000	3

Figure 6–8: Scrap and car loading recipe used to test the model.

Bucket 1		
Charge (lb)	120,000	
Volume (ft^3)	2,129	
Max Volume (ft^3)	2,500	
Layer #	Scrap	Amount(lb)
1	/104	10,000
2	123	15,000
3	/209	40,000
4	101	15,000
5	205	30,000
6	/104	10,000

Bucket 2		
Charge (lb)	120,000	
Volume (ft^3)	2,218	
Max Volume (ft^3)	2,500	
Layer #	Scrap	Amount(lb)
1	/104	10,000
2	105	50,000
3	123	35,000
4	101	15,000
5	/104	10,000

Figure 6–9: Optimized bucket charging with layers.

Layout of Cars						
	1	3	5	7	9	11
Track # 1	205	123	105	205	104	105
Track # 2	101	104	105	205	101	123
	2	4	6	8	10	12

Figure 6–10: Car layout in the scrap hall.

To illustrate the effectiveness of this type of heuristics used in the model, Table 6-4 shows a study of the model performance with heuristics by comparing some indicators of performance, such as the execution time, the number of the set of bucket feasible solutions generated during execution, and the number

of generated car layout solutions. As seen in Table 6-4, the execution time to complete the search tree is 4.43 second, which satisfies the factory requirement, while the model without heuristics is 36.2 minutes, which is not acceptable. The reason behind this big difference is the number of generated car layout solutions as shown in Table 6-4.

Table 6–4: Model performance study

Indicator	Model with Heuristics	Model without Heuristics
Time of running model	4.43(s)	36.2 (min)
Size of bucket set	29	29
Size of car layout	1260(1st) and 120 (2nd)	$1.2x10^{10}$
Possibility of finding an optimal solution	60%	100%

The risk of missing the optimal solution is also investigated in this thesis. The approach we used is to set up 10 simulations with different input recipes, then the times for running the model (t_1)and the scrap charging time (t_2) were compared. Results are shown in Table 6-5. The application of heuristics produce a high degree of deviation of optimal solution (40%) but the heuristic highly improves the speed of the model. The acceptable value of the time of running the model is 10 minutes. As results show, no test of model without heuristics satisfies this requirement. We conclude that the deviation is acceptable in order to get realistic speed.

There are some limitations in the bucket charging model. The most significant one is the fact that the time of charging the bucket from each car is difficult to estimate. In the developed model, car locations are assumed to be constant throughout bucket charging. However, in real-life cars frequently move along the

Table 6–5: Heuristics risk study

Test	Model with Heuristics		Model without Heuristics	
	t_1(s)	t_2(min)	t_1(min)	t_2(min)
1	4.43	17.8	36.2	16.5
2	15.2	21.2	57.2	20.3
3	25.1	19.2	31.0	19.2
4	68.7	15.6	47.5	15.6
5	11.2	15.4	70.1	15.4
6	3.5	20.1	25.5	19.5
7	70.5	19.8	46.7	18.8
8	33.0	17.5	53.2	17.5
9	42.3	16.4	44.5	16.4
10	9.7	22.1	32.1	22.1

track. Although average charging durations are empirically measured, in real production they change significantly.

CHAPTER 7
Industrial Application

The scrap optimization model and the scrap distribution model were tested at ArcelorMittal using production data. Results are presented in this chapter.

Compared with other methods used in the literature, our linear programming model has the following advantages:

1). It does not need a lot of historical data. Statistical analysis, as a common-used empirical approach, plays a very important role in modelling scrap optimization such as in [10]. However, this method needs a lot of historical data and cannot be used in cases like ArcelorMittal, where not enough data can be provided to generate a precise model.

2). Time efficiency. Each day the operator spends a few minutes to run the model to obtain a scrap recipe for the whole day's production. In case some new scrap arrives or the price of certain scrap changes, the production recipe can be updated immediately to reduce the cost.

This chapter initially introduces the software, then a graphical user interface, including both above stated models. Then, the models are validated and compared with the real production, which is presented in Section 7.2. Finally, Section 7.3 presents a test of the model's performance.

7.1 Introduction to GUI

As an objective of this project, a software was to be designed to optimize daily steel production. This software was to have the following features:

- User-friendly environment. It is necessary to have an interface where operators with little knowledge of model can successfully update the information on steel properties and available scrap and then run the model.
- Capable of dealing with a large amount of data. In the production environment, a significant amount of data needs to be handled every day, including more than 30 types of scrap properties, steel grades, scrap stock availability, scrap recipes and product quality control parameters.
- High execution speed. Scrap optimization is a combinatorial optimization problem whose structure is very large. The software should run the model fast enough to make the scrap recipe preparation phase short to avoid delaying production setup.

Microsoft Excel provides a powerful tool to satisfy the above stated requirements. This commercial spreadsheet features calculation, graphing tools, and a macro-programming language called Visual Basic for Applications. The optimization algorithms can be implemented using the Visual Basic Editor, which includes a window for writing code, debugging code and code module organization environment. VBA codes interact with the data in the spreadsheet, which makes it possible to update the model with the input data and optimization constraints written in spreadsheets. Finally, the codes can export results in spreadsheets. It

is possible to utilize code module organization to individually represent the scrap cost optimization model and scrap distribution model.

7.2 Model Validation

In this section, we compare our results with the real production conducted at ArcelorMittal Inc. (Contrecoeur, Quebec). In the first section, the scrap optimization model is tested and compared with real production data. In the subsequent section, the scrap distribution model is tested considering the scrap recipe generated from the first section as the input and then a comparison with real production is conducted.

7.2.1 Scrap Optimization Model Validation in Production Practice

Experiments on the scrap optimization model were conducted in the factory and then the results were compared with the scrap recipe being used in the production. Figure 7-1 gives the requirements for liquid steel. As seen from the Figure 7-1, liquid steel with 0.3 % of copper is needed in the production along with the requirement for the weight of scrap, volume limit and alloying elements.

Aim Scrap W.(lb)	240,000
Max Vol.(ft^3)	5,000
Max Cu (%)	0.3
Max Cr (%)	0.80
Max S (%)	0.25
Max Sn (%)	0.20

Figure 7–1: Summary of liquid steel product requirement

The candidate scrap is shown in Figure 7-2, along with scrap properties, market information and availability.

Code	Type	Min (p. heat)	Max (p. heat)	Cost	Cu(%)	Cr(%)	S (%)	Sn (%)	Melt Yield(%)	Density
105	Busheling	0	100,000	359	0.12	0.04	0.01	0.008	0.95	43.3
504	Shred	0	70,000	395	0.18	0.1	0.04	0.008	0.925	69.5
101	#1	30,000	100,000	385	0.37	0.15	0.03	0.015	0.88	34.5
123	Turnings	0	30,000	277	0.3	0.1	0.07	0.008	0.85	102.7
108	Internal Skulls	0	25,000	14	0.3	0.4	0.03	0.008	0.85	200
110	Tire wire	0	15,000	297	0.15	0.04	0.002	0.004	0.9	75
201	Billets	0	10,000	1	0.3	0.6	0.025	0.008	0.98	250
208	External Skulls	0	30,000	196	0.05	0.06	0.07	0.006	0.85	200
111	P&S	0	60,000	382	0.15	0.04	0.015	0.008	0.94	42.9
104	Shred	0	70,000	570	0.28	0.1	0.04	0.008	0.925	69.5
404	Shred	0	60,000	401	0.35	0.1	0.04	0.008	0.925	69.5

Figure 7–2: Properties of used scrap in the EAF

Figure 7-3 and Figure 7-4 show the scrap recipe and properties both generated from our model and used in real production. As seen from the figure, our model has many advantages:

- The optimized recipe provides more parameters to control production and knows more about the process. In real production, only total charge, copper content and cost per ton are given to control the scrap charging. Besides the above stated parameters, the model also gives the estimated information about copper content, chromium content, sulpher content, tin content, volume and the estimated tapped weight and cost per heat. These additional parameters are useful to control the scrap charging.
- The optimized recipe reduce the production cost in terms of scrap cost, which is decreased from $ 342.51 to $ 330.50 per ton. This was our main objective.

Code	Type	Optimized Recipe	Original Recipe
105	Busheling	86,000	40,000
504	Shred	40,000	59,000
101	#1	30,000	30,000
123	Turnings	30,000	0
108	Internal Skulls	24,000	25,000
110	Tire wire	14,000	20,000
201	Billets	10,000	10,000
208	External Skulls	6,000	5,000
111	P&S	0	40,000
104	Shred	0	0
404	Shred	0	0
205	Tin cans	0	5,000

Figure 7–3: Scrap recipe in the optimization model and in real production

Output Parameters	Optimized	Original
charge (lb)	240,000	234,000
Cu (%)	0.21	0.20
Cr (%)	0.13	N/A
S (%)	0.03	N/A
Sn (%)	0.01	N/A
Volume (ft^3)	4,100	N/A
Tap Weight (lb)	218,500	N/A
Steel Cost/t	$ 330.50	$ 342.51
Cost/Heat	$36,107.00	N/A

Figure 7–4: Properties of liquid steel and production in the optimization model and in real production

Table 7-1 shows a cost reduction comparison of the scrap optimization model and their real production. The solutions generated from our model promise scrap combinations of significantly reduced cost than those from the present method they use. The relative cost savings range from 3.51% to 5.97%. This cost is compared in the unit of $ per ton. The comparisons of scrap recipe with maximal Cu content of 0.35 and 0.8 % are shown in Appendex B.

Table 7–1: Cost reduction

Max Cu content (%)	Original cost	Optimized cost	Cost reduction
0.3	342.51	330.50	3.51%
0.35	307.05	293.93	4.27%
0.8	294.80	277.20	5.97%

7.2.2 Scrap Distribution Model Validation by Historical Production Data

After obtaining the scrap recipe, we then tested and validated the scrap distribution model with the scrap recipe as input data. This model is only validated by the historical production data not by real production practice due to the limited duration of the project at ArcelorMittal.

Figure 7-5 shows an example scrap recipe. Figure 7-6 depicts the results obtained with the developed model, compared with the solution given in Figure 7-7, which refers to the bucket loading applied in the real production. The distribution layout of cars obtained from the model is shown in Figure 7-8.

Code	Type	Number Of Cars
123	Turnings	1
104	Shred	2
205	Tin cans	4
101	#1	2
105	Busheling	2
108	Internal Skulls	192,000
201	Billets	1
208	External Skulls	80,000

Figure 7–5: Car loading recipe

Bucket 1		
Charge (lb)	120,000	
Volume (ft^3)	1,750	
Max Volume (ft^3)	2,500	
Layer #	Scrap	Amount(lb)
1	/104	10,000
2	205	5,000
3	/108/208	34,000
4	101	15,000
5	105	26,000
6	201	20,000
7	/104	10,000

Bucket 2		
Charge (lb)	120,000	
Volume (ft^3)	2,446	
Max Volume (ft^3)	2,500	
Layer #	Scrap	Amount(lb)
1	/104	10,000
2	123	50,000
3	205	35,000
4	101	15,000
5	/104	10,000

Figure 7–6: Layered bucket with scrap from model

Bucket #1		
Charge (lb)	125,000	
Layer #	Scrap	Amount(lb)
1	/104	10,000
2	105	35,000
3	/108/208	35,000
4	205	5,000
5	110	25,000
6	/104	15,000

Bucket #2		
Charge (lb)	110,000	
Layer #	Scrap	Amount(lb)
1	/104	20,000
2	105	20,000
3	123	15,000
4	101	30,000
5	201	5,000
6	/104	20,000

Figure 7–7: Layered bucket with scrap in real production

Layout of Cars						
	1	3	5	7	9	11
Track # 1	105	123	104	205	205	101
Track # 2	205	101	201	205	104	105
	2	4	6	8	10	12

Figure 7–8: The layout of railcars in the scrap hall.

At ArcelorMittal, there is a rough estimation of charging time as shown in Table 6-1. Due to the limitation of modelling time taken to carry scrap from

railcars to buckets, the simulated results cannot be compared to the time in Table 6-1. We do not have any information about the car layout in the historical data. Therefore, the result we have agrees closely with the ideal layout. However, comparison to real production practice is needed to experimentally validate our model in the future.

Table 7–2: Bucket charging time

Max Cu content (%)	Time (first)	Time (second)	Time (both)
0.3	20.15	18.57	39.72
0.35	19.48	15.23	34.71
0.8	20.83	15.12	35.95

7.3 Model Performance Test

The performance of the scrap optimization model refers to (1) the speed of obtaining the needed recipe and (2) how accurately the optimal objective function value is reached. In most cases, these two indicators of performance are paradoxical. Although optimizing the objective function is our final goal, we have to consider the usability of the model. It is noted that this model provides some options according to the requirement of the performance.

There are various factors influencing the scrap optimization model and scrap distribution model. It is necessary to evaluate these factors' strength and significance. In this section, we investigate the role they play in the model. The first subsection presents the analysis of factors influencing the scrap optimization model and the second one introduces the influence of some factors on the scrap distribution model. At the end of each subsection, a summary of decision criteria is given.

7.3.1 Scrap Optimization Model Performance Analysis

Factors for the scrap optimization model performance are the number of scrap types in the input data, the availability of scrap per heat, and the increment in the branching function. In order to investigate these factors that influence on the performance of the model, we set 3 scenarios. The system used was Mac OS X of version 10.6.8 whose processor was 2.4 GHz Intel Core 2 Duo and memory was 4 GB and 1067 MHz DDR3. The VBA was Microsoft Excel for Mac 2011 with version 14.1.0.

Scenario 1 The first scenario investigates the impact of the number of scrap types on the model performance. Simulations were conducted with the number of scrap types ranging from 6 to 20 and meanwhile other factors were set as constants. The time taken to obtain the recipe is compared in Figure 7-9 and the optimized scrap cost are shown in Figure 7-10.

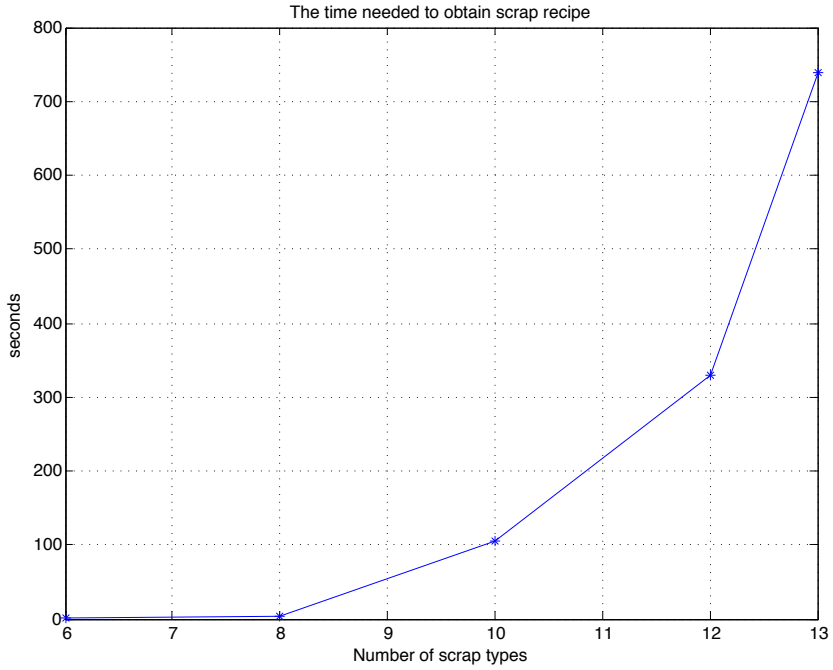

Figure 7–9: The time needed to obtain the recipe when the number of scrap types ranges from 6 to 13.

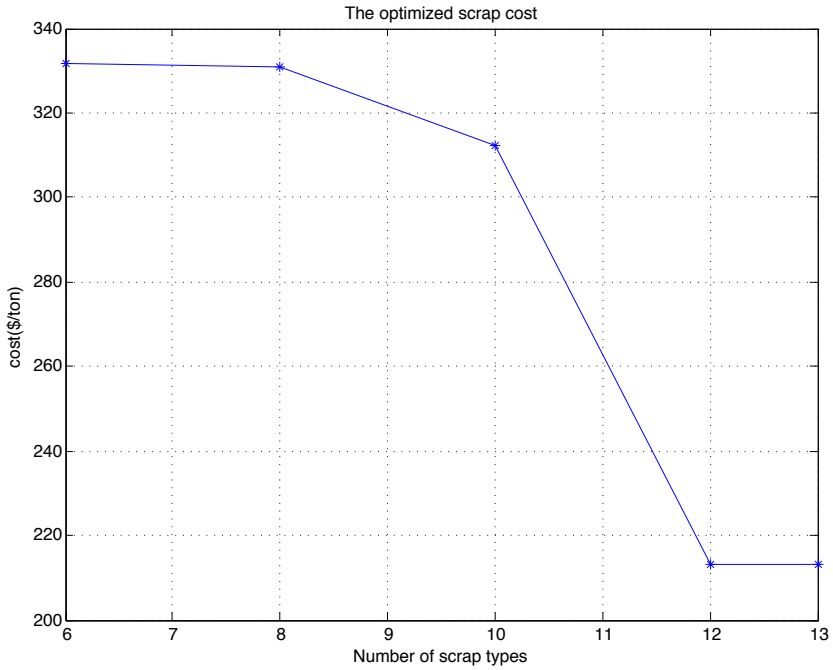

Figure 7–10: The optimized scrap cost when the number of scrap types ranges from 6 to 13.

As seen from the above figures, the greater the number of scrap types we used, the more time the model needs to obtain the final recipe. In addition, the cost of scrap, which depends on whether the additional types of scrap are used, can significantly decrease. In fact, when we simulate the model, the time exceeds an acceptable limit when the number of scrap types is too large. The number of scrap types determines the size of combinatorial search tree since this changes the number of levels in the search tree.

Scenario 2 The second scenario studies the impact of the availability of scrap per heat. In the simulation, the number of scrap types is 8 and the maximum

amount of scrap changes from 20000 to 120000 lbs. Results are plotted in Figure 7-11 and Figure 7-12. The maximum amount of scrap is limited in practice.

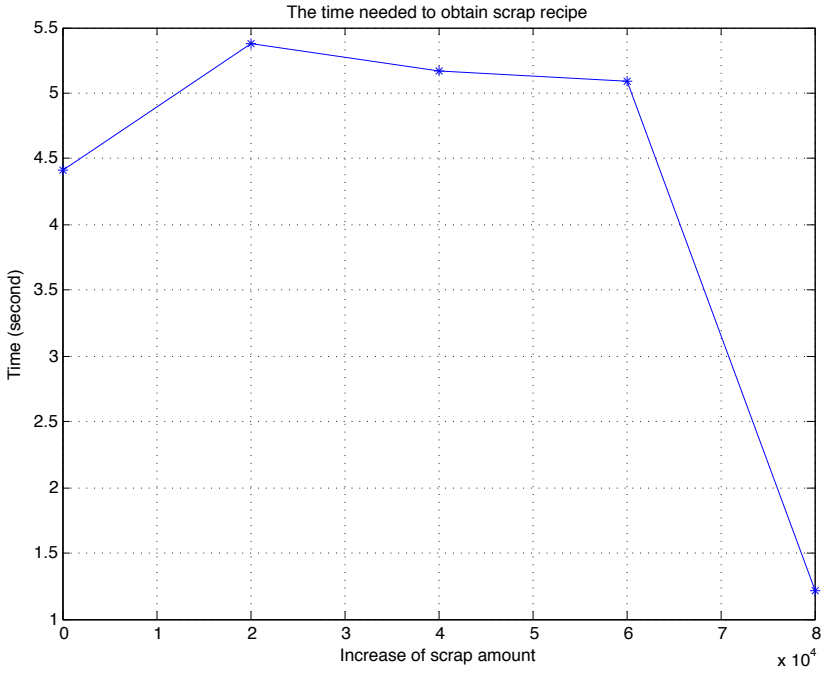

Figure 7–11: The time needed to obtain the recipe while increasing the amount of scrap from 0 to 80000 lbs.

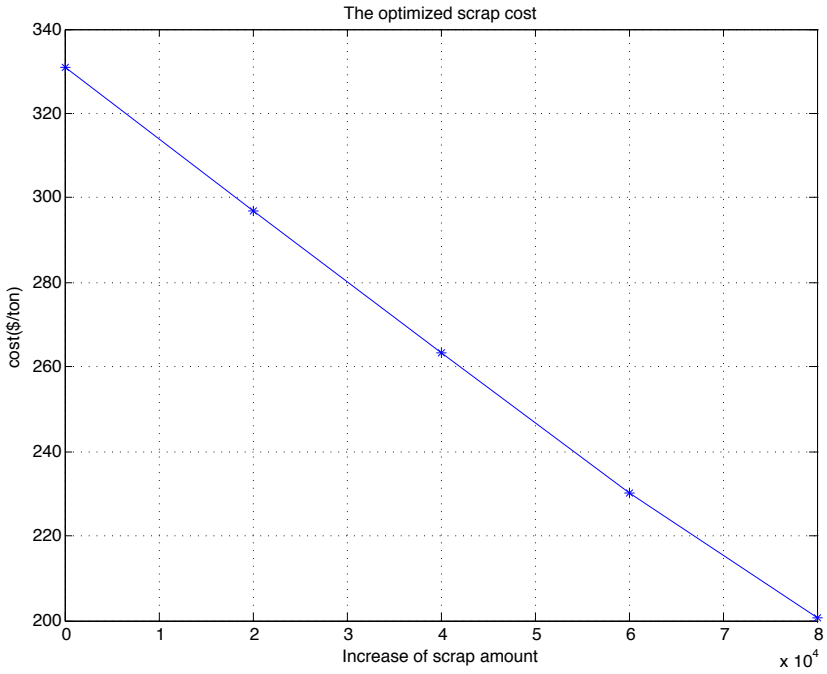

Figure 7–12: The optimized scrap cost as the amount of scrap increases from 0 to 80000 lbs.

Results shows that the maximum values for scrap are likely to increase algorithm efficiency and reduce the cost. Intuitively, the increase of maximum values can reduce model running speed because it increases the branches at each node in the search tree. In fact, the effect is opposite as observed in the results. This is because the increase of maximum values make the cost small and the upper bound of bounding functions is updated quickly. Thus, a lot of branches are bounded early and unexplored.

Scenario 3 The last scenario performs the simulation to investigate how the increment in the branching function influences model performance. We consider

8 types of scrap and the increment is set to 2000, 4000, 5000, 8000, and 10000.

Figure 7-13 and 7-14 show the results.

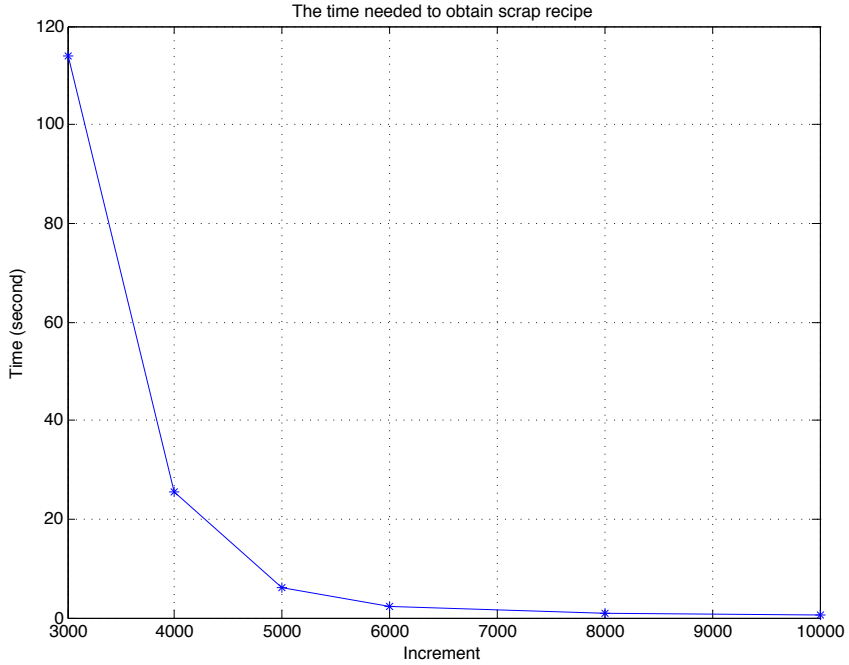

Figure 7–13: The time needed to obtain the recipe with the increment changing from 2000 to 10000.

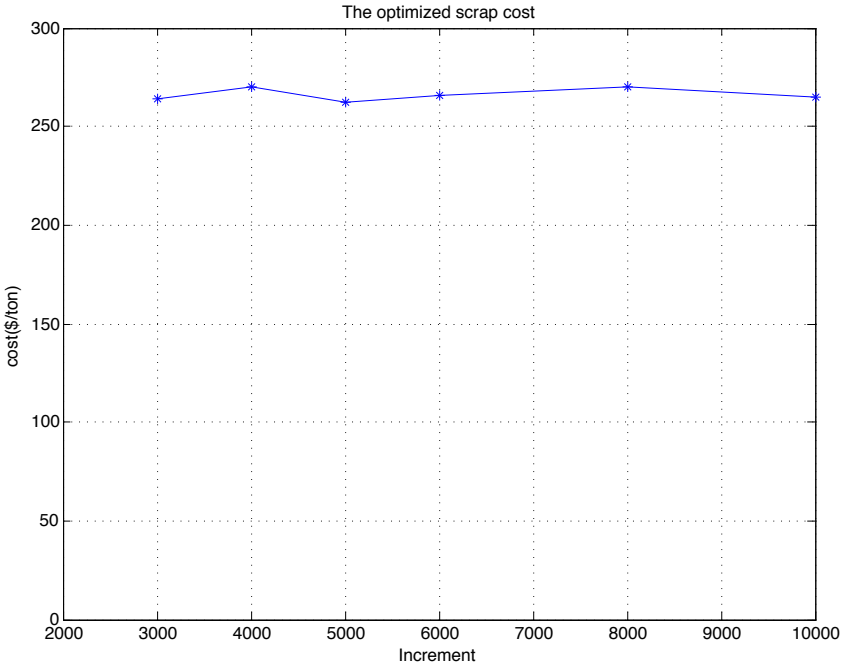

Figure 7–14: The optimized scrap cost with the increment changing from 2000 to 10000.

It is observed from the plots, the increment of branching function have a large impact on the speed of running the model. Speed is decreased with the decrease of increment.

Criteria of Choosing the Model

In the end, when the model is employed in production, the best strategy to reduce scrap cost and shorten recipe preparation is

- to use the least number of scrap types
- to increase the maximum value of each type of selected scrap within a realistic range

- to choose a large increment with a guarantee of the recipe being in a realistic range

In real-world production, the model is chosen by a compromise of the above three rules.

7.3.2 Scrap Distribution Model Performance Analysis

Unlike the scrap optimization model, the scrap distribution model considers the loading time as an objective function. Thus, the performance indicators are the loading time and the time of reaching an optimal solution. Factors influencing scrap distribution model performance are the scrap recipe and the increment in the branching function of the bucket layer optimization algorithm. In order to investigate the impact of these factors on the performance of the model, we set a scenario: take a population of 8 which refers to 8 different scrap recipes where the increment is set as 2000 (blue line), 4000 (red line) and 5000 (green line) for each sample. Simulation results are shown in Figures 7-15 and 7-16.

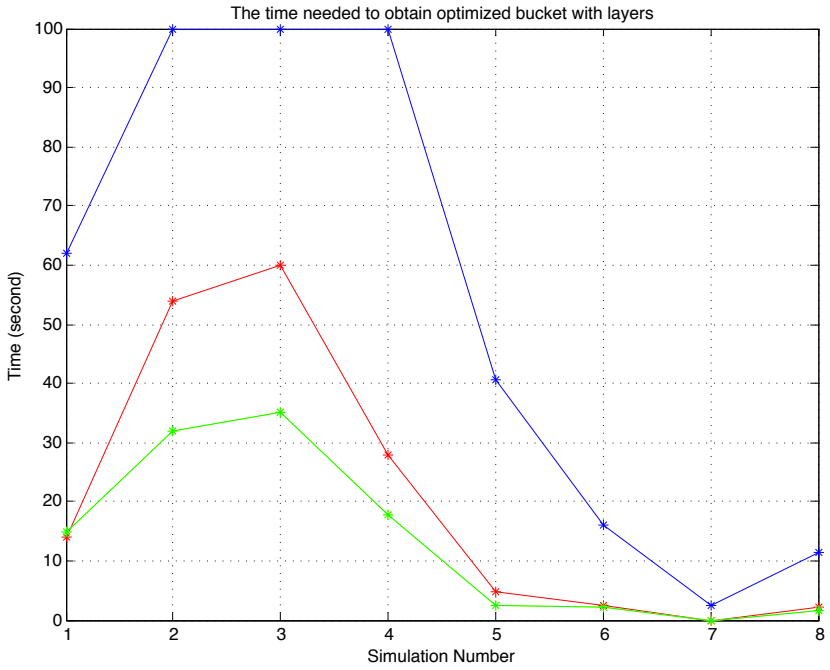

Figure 7–15: The algorithm running time with the increment changing from 2000 to 5000.

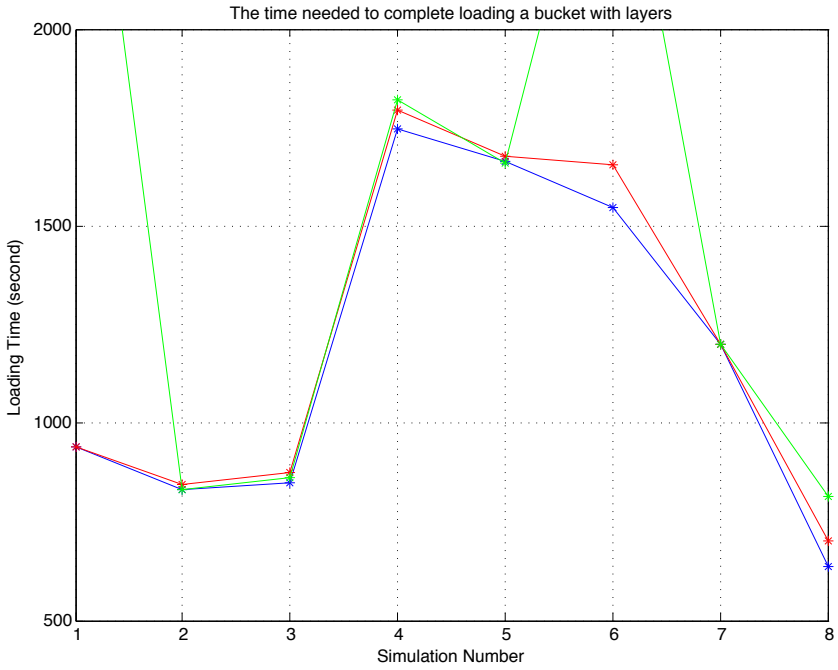

Figure 7–16: The optimized loading time with the increment changing from 2000 to 5000.

In Figure 7-16, points whose value exceeds 2000 are discarded due to their being unrealistic. As observed in the above figures, the model has a similar effect with an increment of 2000 and 4000 but the algorithm runs faster with an increment of 4000 compared to 2000. The performance of the increment of 5000 is quite unstable and takes more time. This is because loading scrap layering buckets generates more movements for a crane when the increment is 5000.

Criteria of Choosing the Model

To conclude, 4000 is a good option for the increment if the production has a strong requirement on the speed of recipe preparation; under the conditions that

slow bucket loading can impact the normal production 2000 is better to guarantee the speed of loading process.

CHAPTER 8
Conclusion and Future Work

The goal of this work was to develop a scrap cost optimization model for steel production using an Electric Arc Furnace. In the simulation, special attention was paid to two areas: optimization of the scrap recipe and scrap charging for the EAF. Compared with the historical data at ArcelorMittal, the model was found to offer a good reduction of both production cost and scrap charging time. In the sections below, concluding remarks are given for each of the main areas of this work.

8.1 Scrap Optimization Model

In Chapter 4, the mathematical optimization model for scrap cost was presented followed by the application of branch and bound rules and heuristics used in it. The industrial test of the model was given in Chapter 7. As simulation results showed, the results of the model quite matched the real production situation well. What is better is that the total cost of the optimized scrap recipe was lower than the current used one.

Since we consider the volume as a constraint, some production problems are solved by the model. For example, some scrap recipes can result in difficulty in charging furnace, but the model succeeds in estimating the volume of scrap in production. Thus, candidate scrap recipes with large volumes and low costs are

not considered as the final solution; so, there is no resultant difficulty in charging the furnace.

Thanks to the branch and bound rules, the model has good speed performance to obtain the final recipe. This made it possible to let the model become a useful tool for engineers at ArcelorMittal at the end of the project. The model can update the recipe according to the price change in the scrap market. Successful validation proved that any assumptions we made in the model were reasonable.

8.2 Scrap Distribution Model

In steel production, charging time is a problem engineers often meet. Any delay of production can result in a cost increase. A bad scrap dispatch system can delay the production due to a lack of scrap in the scrap hall and scrap steel has to be fetched from the scrap yard. So, an estimation of needed scrap in multi-heat based production is made possible by developing a scrap distribution model.

As part of the production model, the scrap distribution model can help charge scrap more quickly and efficiently. Chapters 5 and 6 illustrated the structure of the model and the approach of the branch and bound rules and heuristics. The way of determining the car layout in the scrap hall is quite scientific and powerful instead of just using operators' and engineers' experience. This is a significant contribution to the production efficiency.

Due to time issues, this model was only validated by historical production data. So, we cannot give any comment on the improvement of the EAF charging in real production practice. As seen from the industrial test in Chapter 7, the car layout is close to ideal case and the bucket layering avoids the problem of

charging the EAF. The total volume of scrap in the bucket is lower than the furnace capacity.

The model has a good running speed to calculate the optimal solution on a normal desktop computer which is advantageous for industrial use.

8.3 Future Work

The following suggestions pertain to the model:

- Artificial intelligence techniques can be applied to the model to improve the model performance.
- In the scrap cost optimization model, more cost factors (operation cost, yield) could be considered in the future. A slag model and caster model have been developed as well by my colleague in this project. They can be combined with the scrap charging model.
- Develop weekly, monthly or even longer time based model, instead of daily based.
- In order to react to more market information, future work should focus on incorporating some factors of international trade due to the high level of globalization of steel production.
- In North America, ArcelorMittal has other facilities using an Electric Arc Furnace. The model can be extended to connect all related facilities.
- The scrap distribution model could be tested in real production resulting in possible improvement.
- The scrap distribution model could incorporate additional cars in the scrap hall and consider more heats when determining the car layout.

References

[1] http://canadasteel.ca.

[2] http://www.msa.com/metal/systour/systour.html.

[3] http://www.worldsteel.org.

[4] Image of eaf at http://www.steeluniversity.org.

[5] Cook W. J.; Cunningham W. H.; Pulleyblank W. R.; Schrijver A. Combinatorial optimization. *Wiley*, 1998.

[6] Wilson E.; Kan M.; Mirle A. Intelligent technologies for electric arc furnace optimization. *ISS Technical Paper*, 21:16, 2005.

[7] Gouweloos D.. Design of a standard operating procedure for the loading of buckets in camps drift remelt. *A project in University of Pretoria*, 2011.

[8] Bernatzki K. P.; Bussieck M. R.; Lindner T.; Lubbecke M. E. Optimal scrap combination for steel production. *OR Spektrum*, 20:251–258, 1998.

[9] MacRosty R. D. M.; Swartz C. L. E. Dynamic modeling of an induxtrial electric arc furnace. *American Chemical Society*, 44:8067–8083, 2005.

[10] Sandberg E. Energy and scrap optimization of electric arc furnaces by statistical analysis of process data. *Thesis in Lulea University of Technology*, 21:1402–1757, 2005.

[11] Clausen J. Branch and bound algorithms - principles and examples. *University of Copenhagen*, 1999.

[12] Liu X. B.; Pan R. L.; Meng Q. N.; Cui F. J. Study on recipe cost opimization system based on ant colony algorithms. *International Seminar on Business and Information Management*, 78:287–290, 2005.

[13] Nocedal J.; Wright S. J. Numerical optimization. *Springer*, 1999.

[14] Snell J. J. Improved modeling and optimal control of an electric arc furnace. *Master's Thesis*, University of Iowa, 2010.

[15] Hoffman K.. Combinatorial optimization: Current success and directions for the future. *School of Information Technology and Engineering, George Mason University*, 2010.

[16] Hoffman KL.; Padberg M. Lp-based combinatorial problem solving. *Annals of Operations Research*, 4:145–194, 1991.

[17] Savelsbergh MWP. Preprocessing and probing techniques for mixed integer programming problems. *ORSA J Comput*, 6(4):445–454, 1994.

[18] Algadrie; Palyama. Cost saving in eaf operation neural network controller. *SEAISI Quarterly*, 27:55–60, 1998.

[19] Rong A.; Lahdelma R. Fuzzy chance constrained linear programming model for optimizing the scrap charge in steel production. *European Jounal of Operational Research*, 186:953–964, 2008.

[20] Garvin W. W. Introduction to linear programming. *McGraw-Hill*, 1960.

Appendix A

Properties of scrap steel

Code	Type	Cost($/t)	Cu(%)	Cr(%)	S (%)	Sn (%)	Yield(%)	Density(lb/ft^3)
101	#1	385	0.37	0.15	0.03	0.015	0.88	34.5
112	#2	250	0.5	0.12	0.04	0.012	0.88	20.4
326	#2	250	0.5	0.12	0.04	0.012	0.88	20.4
201	Billets	1	0.3	0.6	0.025	0.008	0.98	250
408	Boulettes (DRI)	450	0	0	0.005	0	0.8	100
160	Briquette	308	0	0	0.02	0	0.8	100
105	Busheling	359	0.12	0.04	0.01	0.008	0.95	43.3
126	Busheling	359	0.12	0.04	0.01	0.008	0.95	43.3
305	Busheling	359	0.12	0.04	0.015	0.008	0.95	43.3
209	External Skulls	196	0.02	0.09	0.08	0.006	0.85	200
109	External Skulls	196	0.02	0.1	0.5	0.02	0.85	200
208	External Skulls	196	0.05	0.06	0.07	0.006	0.85	200
211	External Skulls	196	0.21	0.11	0.002	0.015	0.85	200
308	External Skulls	336	0.03	0.18	0.5	0.008	0.74	150
508	External Skulls	196	0.05	0.06	0.07	0.006	0.85	200
609	External Skulls	160	0.04	0.04	0.09	0.004	0.85	200
709	External Skulls	150	0.12	0.95	0.006	0.006	0.85	200
509	Fonte	336	0.04	0.04	0.02	0.008	0.74	150
108	Internal Skulls	14	0.3	0.4	0.03	0.008	0.85	200
111	P&S	382	0.15	0.04	0.015	0.008	0.94	42.9
104	Shred	431	0.3	0.1	0.04	0.008	0.925	69.5
204	Shred	401	0.2	0.07	0.02	0.013	0.925	69.5
304	Shred	390	0.4	0.1	0.02	0.2	0.925	69.5
404	Shred	401	0.35	0.1	0.04	0.008	0.925	69.5
205	Tin cans	323	0.04	0.04	0.01	0.3	0.85	28.3
110	Tire wire	297	0.15	0.04	0.002	0.004	0.9	75
117	Tire wire	250	0.2	0.1	0.025	0.008	0.9	75
123	Turnings	277	0.3	0.1	0.07	0.008	0.85	102.7

Appendix B

Scrap recipe for steel with maximal Cu content of 0.35

Code	Type	Optimized Recipe	Original Recipe
404	Shred	40,000	60,000
111	P&S	36,000	40,000
101	#1	30,000	30,000
112	#2	30,000	20,000
108	Internal Skulls	30,000	30,000
123	Turnings	30,000	15,000
201	Billets	24,000	23,000
110	Tire wire	20,000	20,000
109	External Skulls	0	0
104	Shred	0	0

Scrap recipe for steel with maximal Cu content of 0.8

Code	Type	Optimized Recipe	Original Recipe
112	#2	40,000	22,000
201	Billets	40,000	40,000
404	Shred	40,000	70,000
105	Busheling	32,000	20,000
101	#1	30,000	30,000
123	Turnings	30,000	20,000
108	Internal Skulls	20,000	20,000
205	Tin cans	8,000	15,000
109	External Skulls	0	0
111	P&S	0	0
104	Shred	0	0