FÚTBOL STRATEGIES APPLIED TO OPTIMIZE COMBINATORTIAL PROBLEMS TO CREATE EFFICIENT RESULTS – THE SOCCER HEURISTIC

by

KRISTA M. KUBIK

B.S., Kansas State University 2015

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Industrial and Manufacturing Systems Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Approved by:

Major Professor
Dr. Todd Easton

Abstract

Heuristics are often implemented to find better solutions to computationally challenging problems. Heuristics use varying techniques to search for quality solutions. Several optimization heuristics have drawn inspiration from real world practices. Ant colony optimization mimics ants in search of food. Genetic algorithms emulate traits being passed from a parent to a child. Simulated annealing imitates annealing metal.

This thesis presents a new variable neighborhood search optimization heuristic, fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results, which is called the SOCCER heuristic. This heuristic mimics fútbol and the closest player to the ball performs his neighborhood search and players are assigned different neighborhoods. The SOCCER heuristic is the first application of variable neighborhood search heuristic that uses a complex structure to select neighborhoods.

The SOCCER heuristic can be applied to a variety of optimization problems. This research implemented the SOCCER heuristic for job shop scheduling problems. This implementation focused on creating a quality schedule for a local limestone company.

A small computational study shows that the SOCCER heuristic can quickly solve complex job shop scheduling problems with most instances finishing in under an half an hour. The optimized schedules reduced the average production time by 7.27%. This is roughly a 2 day decrease in the number of days required to produce a month’s worth of orders. Thus, the SOCCER heuristic is a new optimization tool that can aid companies and researchers find better solutions to complex problems.

Table of Contents

List of Figures

List of Tables

Dedication

This thesis is dedicated to my parents, Richard and Elaine, and my siblings, Lauren, Kalen, and Rachel.

Acknowledgments

I would like to acknowledge Dr. Todd Easton. Without his patience and knowledge this research would not have been possible. I would also like to recognize my father, Richard Kubik, whose insight and advice concerning program architecture was an invaluable tool for implementing such a complex system.

Chapter 1: Introduction

Fútbol, called soccer in the United States, is the most popular sport in the world. During the 2010 FIFA World Cup over 3.2 billion people watched at least one minute of a game [25]. With 22 players on the field, determining the best method to achieve the most goals, while still prohibiting the other team from scoring, is intellectually challenging. The result is numerous formations with different players having different responsibilities. This complex nature of fútbol enables it to be an excellent framework for a new optimization heuristic. The primary contribution of this thesis is the development of fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results, the SOCCER heuristic, and applying this heuristic to a job shop scheduling problem.

Heuristics are often implemented in order to find better solutions to computationally challenging problems. This is typically done by moving between neighboring solutions. If two solutions are similar according to some well-defined measure, they are neighbors. Heuristics report the best found solution, but do not guarantee optimality. The method of moving between solutions determines the type of heuristic.

One common heuristic used in optimization problems is simulated annealing. This heuristic mimics the processes of annealing metal. When annealing metal, the best results are obtained when the initial temperature is high and the metal cools slowly over time. In the heuristic, one starts with a transition probability. This probability determines the likelihood of one accepting worse solutions. As the number of iterations increases, the probability decreases. Simulated annealing has been used numerous times to solve various problems [33] including job shop scheduling problems. For instance, it has been used to schedule maintenance work at power plants [32], job shop scheduling problems [39], and class scheduling [4].

Genetic algorithms have also been used in a multitude of optimization problems. This heuristics mimics evolution. When parents have children, some of their genes are passed onto their child. For a genetic algorithm, certain traits from a solution are passed on to the new child solution. Genetic algorithms have been used in numerous instances [17], including job shop scheduling [5], [40].

Another common heuristic is ant colony optimization. This search is based on how ants search for food. As an ant moves, it lays down a pheromone trail. A different ant is not likely to take the same path if only one set of pheromones have been laid because the ant that laid them did not return. However, if the ant returned on the same path, twice the pheromones exist on the path and more ants are drawn to that path. This heuristic uses this idea to reinforce certain searches, while discouraging others. The heuristic has been used to solve a wide range of problems, including job shop scheduling [35].

Heuristics are valuable tools which can provide quality solutions to complex problems. Each of the heuristics mentioned have different applications and use various methods. This research provides a new heuristic, which is based on a fútbol game and the way players move. To show the value of the heuristic, it is applied to job shop scheduling.

1.1. Job Shop Scheduling

Manufacturing facilities have been used for centuries to produce goods. Typically, a factory is either modelled as a flow shop or a job shop. A flow shop facility processes each job on all machines in the same route. A standard flow shop operation is an assembly line. For job shops, each job has a specified processing route. These routes vary based on the product produced. Furthermore, the route must be completed in a set sequential order.

A complete job shop schedule is an assignment of all jobs to machines at specified times. This assignment must follow each job's processing route. Furthermore, the time each job is assigned to each machine must be equal to the time it takes to process the job. It is important to note that a machine can only process one job at a given time.

Numerous heuristics have been used to find quality solutions to job shop scheduling problems. Some common heuristics which have been applied to job shop scheduling are simulated annealing [39], tabu search [6], and ant colony optimization [35].

1.2. Research Motivation

This research started when a local limestone company asked for help with its scheduling method. Its manufacturing facility is a job shop. The company produces a large number of standard products and has numerous machines for processing. Due to the variations in products and processes, scheduling proves to be complex.

After researching a number of heuristics, it was decided that variable neighborhood search heuristic should be considered for this job shop scheduling problem. Variable neighborhood search heuristic is another heuristic used in optimization problems. This method finds good solutions by searching through various neighborhoods. This allows for a wide variety of search techniques to be used in a single problem. It also allows one to escape from a local optimal solution, which is important when searching for quality solutions. Variable neighborhood search heuristic has been applied to job shop scheduling on numerous occasions [21], [1], [30].

The downfall with variable neighborhood search heuristic is the method for switching between neighborhoods. Two methods are commonly used for determining the neighborhood. The first method is to randomly select a neighborhood. Then, after a random number of instances, a new random neighborhood is selected. The second method has a set initial neighborhood. Then, after a set number of iterations, the next neighborhood is selected. This continues and cycles back to the initial neighborhood. These two methods simply bounce between neighborhoods randomly. Since many heuristics draw inspiration from real world situations as previously discussed, it was decided that variable neighborhood search heuristics should also be based on a real world phenomenon.

1.3. Research Contribution

This thesis presents a new heuristic, fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results – the SOCCER heuristic. The SOCCER heuristic was inspired by fútbol. The current solution corresponds to the fútbol ball. All 22 players on the field have a position, speed, and a location on the field. Each position has neighborhood associated with it.

The players move about the field in pursuit of the ball and the two teams seek to move the ball in opposing directions. The offensive team seeks to move the ball towards the goal, while the defense seeks to prevent it. The player who can reach the ball the quickest is provided with the opportunity to kick the ball. The time needed for each player to reach the ball is calculated using his speed and proximity to the ball. Once the player is determined, he uses his neighborhood to find a new solution. This corresponds to an attempted kick. If the player is on the offensive team, his solution is accepted if it is better than the previous solution. Solutions found by the defensive team are accepted if worse than the previous solution. The defensive team is used to escape local optimal solutions.

The SOCCER heuristic has a generic structure that could be applied to numerous optimization problems. In this research, the SOCCER heuristic was tailored and built to create quality job shop scheduling solutions. To test the quality of this heuristic, it was applied to data from a limestone company.

The SOCCER heuristic is capable of scheduling production for 24 days in approximately 14 minutes. This heuristic improves the production schedules by about 2 days per month’s work of data. Thus, the SOCCER heuristic is a quality optimization tool that can aid companies and researchers find better solutions to complex problems.

1.4. Outline

Chapter 2 is a literature of previous research pertaining to optimization problems. The first section explains job shop scheduling. Then, neighborhoods are explained to better understand the searches used in meta-heuristics. Next, common meta-heuristics are presented. Then background on variable neighborhood search heuristic and its applications are discussed.

Chapter 3 introduces the SOCCER heuristic. First, it explains fútbol. Detail is provided on the player's positions and characteristics. Next, the SOCCER heuristic is presented. The pseudo-code for this heuristic is also provided.

Chapter 4 details the SOCCER heuristic's application to job shop scheduling. It starts by providing background information on the limestone company, which provided the data for this heuristic. Then, the fútbol heuristic is converted to fit the limestone company's job shop scheduling. The computational study shows that the SOCCER heuristic can save the company almost 2 days' worth of production for every month's worth of orders. Implementing the SOCCER heuristic will allow the limestone company to produce approximately 24 additional days' worth of production per year.

Chapter 5 is a summary of this thesis. The chapter begins with the key contributions of this thesis. Then, computational results are summarized. The chapter concludes by presenting possible directions for future research.

Chapter 2: Background

This chapter contains the background information necessary to understand the contributions of this research. The first section of the chapter explains the job shop scheduling problem. The second section explains neighborhoods and some common meta-heuristics used to find quality solutions to optimization problems. The final section discusses variable neighborhood search heuristic techniques.

2.1. Job Shop Scheduling

Factories have been used for centuries to produce a variety of goods. In the past, parts were manufactured on an individual basis. Over time, the complexities of the processes increased as technology advanced. With the innovations of interchangeable parts and the assembly line, factories vastly changed operations. Today numerous different types of factories exist.

For over a century, industrial engineers have attempted to make factories more efficient. A common industrial engineering technique is to model these factories as scheduling problems. Factories produce orders and these orders are called jobs in the scheduling models.

The simplest scheduling models have a single machine. Other models have machines running in parallel. Parallel machines allow for a job to be processed on any machine of a given subset, which allows more flexibility. However, having parallel machines complicates finding the optimal schedule.

Large facilities are typically modeled as flow shop [18] or job shop [37]. A flow shop facility processes each job on all machines in the same route. A standard flow shop example is a factory that is an assembly line. For job shops, each job has a specified processing route. This route can contain one or more machines. Furthermore, the route must be completed in a set sequential order. A limestone company is a job shop and the focus of this thesis is on job shop scheduling. The remainder of the thesis focuses on job shop scheduling.

The input to a job shop scheduling problem (JSS) is a set of n jobs, $J = \{j_1, j_2, \dots, j_n\}$ and a set of q machines $M = \{m_1, m_2, \dots, m_q\}$. Each job has a processing route or steps which are denoted as $R_{j_i} = (m'^{i}_{1}, m'^{i}_{2}, \dots, m'^{i}_{r})$ where each $m'^{i}_{l} \in M$ for all $l = 1,\dots,r$ and $i = 1,\dots,n$. Each job also has a processing time on each machine in its route which is denoted as $(p^i_1, p^i_2, \dots, p^i_r)$.

A feasible solution to a JSS is an assignment of jobs to machines at given times. This assignment must complete every job and the steps must be completed in the order of the processing route. Furthermore, the time each job is assigned to each machine must be equal to its processing time and no machine can work on more than one job at the same time.

Numerous variations of JSS are available. A common assumption is that the machine must finish an entire process. Occasionally, machines are allowed to preempt a job. The machine stops working on an existing job prior to completing it and begins working on a different job [15]. Some JSS have release dates and due dates. Jobs cannot begin to be processed prior to release dates and should be finished prior to due dates [9]. Many other variations exist in the literature [30].

Various objectives are used to judge the quality of a schedule. A common objective is to minimize the total completion time or the time of completion of the last job. This objective is referred to as the makespan [19]. The goal of this criterion is to process all jobs quickly and minimize the downtime or idle times on machines. Other objectives include minimize maximum lateness [24] and minimizing the number of tardy jobs [26]. Various weighted versions of these objectives are also common [21]. Regardless of the chosen objective, the objective value is typically denoted as z with z^* being the optimal solution

JSS problems are *NP*-complete and some of the most complex problems to find optimal solutions [10]. Some research has been done on finding the optimal solution. These methods typically use integer programming. Integer programs are difficult to solve for JSS instances and typically run for an exponential amount of time. Although heuristics cannot guarantee optimality, they are often used for job shop scheduling problems [3], [8]. This thesis creates a new heuristic based on fútbol which can be applied to JSS as well as other optimization problems.

2.2. Heuristics and Neighborhoods

Heuristics frequently use neighborhood searches to derive quality solutions. A neighborhood search encompasses a region of solutions commonly referred to as neighboring solutions or neighbors. A neighbor is a solution which was found by altering the previous instance. One typically uses neighborhoods to navigate through the search space in problems to hopefully find good solutions.

Formally, let X be the feasible solutions to an optimization problem $\prod$. Each feasible solution $x \epsilon X$ is assigned an objective value $z(x)$ where $z: X \rightarrow \boldsymbol{R}$. An optimal solution to a maximization problem $\prod$ is an $x^* \epsilon X$ such that $z(x^*) \geq z(x)$ for all $x \epsilon X$.

In many optimization problems, feasible solutions may be similar. Two such similar solutions are called neighbors. Formally, let $x' \epsilon X$ and $x'' \epsilon X$ be such that

$||x'-x''||_R < \boldsymbol{\varepsilon}$ where $||\ ||_R$ is defined by some set of rules and $\boldsymbol{\varepsilon}$ is well-defined according to these rules. Define the neighborhood of x', $N_{R,\varepsilon}(x') = \{x \epsilon X\colon ||x'-x''||_R < \varepsilon\}$. An x' is a local optimal solution to a maximization problem $\prod$ if $z(x') \geq z(x)$ for all $x \epsilon N_{R,\varepsilon}(x')$.

A wide range of problems use neighborhoods to search for optimal solutions. Typically, neighborhoods either move jobs to a new location or switch jobs. Small neighborhoods perform minimal changes while large neighborhoods consist of several jobs being altered.

Consider a JSS problem with three jobs and two machines. Table 2.1 provides the routes for each job and processing times on each machines. The processing times are provided in hours. The goal of this problem is to minimize idle time.

Table 2.1 Example Problem

Job 1	Machine	1	2	1
	Time	3	4	2
Job2	Machine	2	1	
	Time	3	4	
Job 3	Machine	2	1	2
	Time	2	4	1

The jobs are scheduled on both machines according to the initial processing order. Machine 1 has an initial processing order of job 1, job 2, job 3, job 1. Machine 2 has a processing order job 1, job 2, job 3, job 3. A job cannot be processed on a machine until its previous process has been completed. This means that machine 2 cannot process job 1 until machine 1 has processed job 1 for 3 hours. This initial production schedule has an idle time of 21 hours and is shown in Table 2.2.

Table 2.2 Initial Production Schedule

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Machine 1	Job 1	Job 1	Job 1								Job 2	Job 2	Job 2	Job 2	Job 3	Job 3	Job 3	Job 3				Job 1	Job 1
Machine 2				Job 1	Job 1	Job 1	Job 1	Job 2	Job 2	Job 2	Job 3	Job 3							Job 3	Job 3	Job 3		

The chosen neighborhood is a small neighborhood. It selects two adjacent jobs on the same machine and swaps them. In this case, machine 2 was selected. Job 1 and job 2 were switched. This changes the processing order on machine 2 to job 2, job 1, job 3, job 3. The order of jobs on machine 1 was not altered. This solution and the previous solution are considered neighbors. The new production schedule is shown in Table 2.3.

Table 2.3 New Production Schedule

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Machine 1	Job 1	Job 1	Job 1	Job 2	Job 2	Job 2	Job 2			Job 3	Job 3	Job 3	Job 3	Job 1	Job 1								
Machine 2	Job 2	Job 2	Job 2	Job 1	Job 1	Job 1	Job 1	Job 3	Job 3					Job 3	Job 3	Job 3							

The new production schedule reduced the idle time to 7 hours. It was also able to complete the three jobs 7 hours sooner than the initial. This process typically continues for the desired number of iterations.

At times some heuristics may allow infeasible solutions. In these situations, *x''* is an infeasible neighbor of *x'* if $||x'-x''||_R \leq \varepsilon$ and $x'' \epsilon\ W / X$ where *W* is the set of all possible solutions to a non-constrained instance of ∏, $||\ ||_R$ is defined by some set of rules and **ε** is well-defined according to these rules.

One common use of infeasible solutions is ejection structures such as chains [23], trees [28], and pools [29]. The goal of these ejection methods is to improve the schedule by making moves to an infeasible solution. The group of removed solutions is stored in an ejection structure. Then some of the solutions are re-inserted into the schedule. If the reinserted job interferes with a job, this job is removed and placed in the ejection structure. This process continues until all jobs from the ejection structure are scheduled and a feasible solution is found.

2.2.1. Common Heuristics

Researchers have created several meta-heuristics to aid in the search for quality solutions. This section discusses the meta-heuristics hill climbing, simulated annealing, and tabu search.

2.2.2. Hill Climbing

The simplest meta-heuristic is called hill climbing. The analogy is taken from a hiker trying to reach the summit on an incredibly foggy day. The hiker looks around in search of rising terrain. If she sees ground higher than her current location she steps there. She continues to step towards higher ground. If there is no direction to go up, she declares herself at the summit. Upon termination, she is locally optimal, but may not necessarily have reached the highest point on the mountain.

Hill climbing heuristic follows a similar process. It starts with an initial solution. Then the neighborhood of the solution is searched until an improving solution is found. Once the solution is improved, the algorithm moves to this new solution and repeats until there are no more improving solutions in the current solutions neighborhood. Thus, the solution is a local optimal solution.

Several researchers have used hill climbing to find quality solutions to scheduling problems. Hill climbing has been applied to scheduling traveling tournaments for various sporting events [22], exam time tabling problem for universities [2], and ground station scheduling [38].

The downfall of hill climbing is that it does not guarantee global optimality because hill climbing only searches neighboring solutions. Therefore, the initial solution determines the likelihood of the local optimal solution also being the global optimal. If the wrong initial solution is chosen, the solution will remain in a sub-neighborhood unable to reach the global optimal due to the neighborhood and the initial solution selected. Because of this, it is not likely for hill climbing to reach global optimality. An example of this is shown in Figure 2.1.

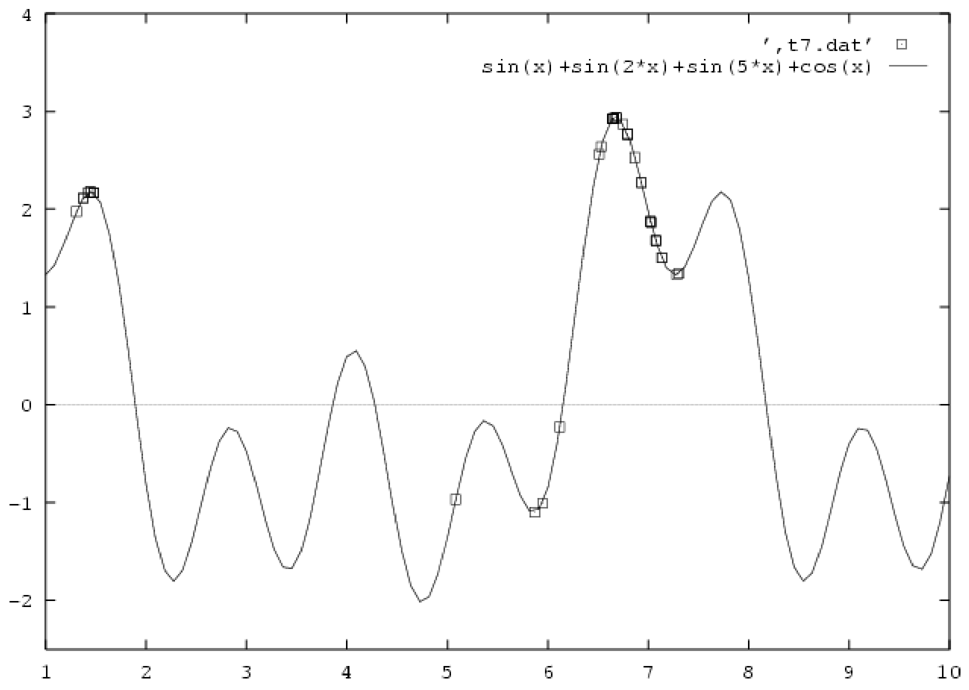

Figure 2.1 Graph of Optimality [14]

In Figure 2.1, a series of nodes are near the point (1.5, 2). These show the different solutions searched. The graph peaks at this point. The peak value represents the local optimal solution found using hill climbing. However, this is not the highest point on the trend. The global optimal is at point (6.5, 3). Using hill climbing would not provide one with the optimal solution unless the initial *x* value was between 5.9 and 7.2. This is due to only accepting a new solution if it is better than the current solution. If a worse solution is found, the heuristic ceases to search in the direction regardless of more optimal searches past the low point. To increase the chances of optimality, one could run hill climbing multiple times starting with a variety of initial solutions. The best value is reported.

By implementing a hill climbing heuristic one can find a local optimal solution in a short period of time, but cannot guarantee global optimality. Therefore, hill climbing is often used in conjunction with other heuristics. Once a different heuristic has been performed, hill climbing is frequently used to check the surrounding solutions and to guarantee that the reported solution is a locally optimal solution.

2.2.3. Simulated Annealing

Another common meta-heuristic is simulated annealing. Simulated annealing is named after annealing metal. When annealing metal, the strongest metal occurs when the temperature is high and the metal is cooled slowly over time. The simulated annealing algorithm takes its inspiration from this process.

In application, simulated annealing starts with an initial solution. The solution is then compared to a neighboring solution. If the new solution is better than the previous, it is accepted. If the new solution is worse than the previous solution, it is accepted based on a transition probability. When a solution is rejected, a new neighbor solution is found and the process repeats.

Typically, the transition probability is a function based on the number of iterations. As the iterations increase, the probability decreases. Therefore, when the model starts, the probability is high and the algorithm is more likely to accept a worse solution than to reject it. As the process continues, the transition probability decreases and more inferior solutions are rejected. This transition probability is often represented as $e^{-T_o(t)}$ where T_0 is the initial temperature and t is the number of iterations. The exponential function is chosen because metal cools according to an exponential function.

Researchers have used simulated annealing to find quality solutions to scheduling problems. For instance, simulated annealing is used to schedule maintenance work at power plants [32], job shop scheduling problems [36], [39], and class scheduling [4].

In theory, if simulated annealing ran forever, it would explore the entire state space and the optimal solution would be found. The downfall of this method is that the duration depends on the initial solution and the parameters used for the transition probability. A slight change in the initial probability could drastically change the effectiveness of the model. Also, the search could take an infinite amount of time depending on the complexity. In practice, simulated annealing does not guarantee an optimal solution.

2.2.4. Tabu Search

The final meta-heuristic discussed here is tabu search. This heuristic starts with an initial solution. Then, a neighbor solution is obtained. Tabu search does not judge the solution based on its perceived quality. Instead, tabu search judges the solution based on whether or not it has already found the solution. If the neighbor solution is new, it becomes the new solution. If it is a repeated solution, it is consider tabu and rejected.

The goal for tabu search is never to repeat the same solution. To minimize the computational effect, the previous *k* solutions are stored. When a new solution is found, it is added to the chain and the oldest solution is removed. This process can be seen in the Figure 2.2.

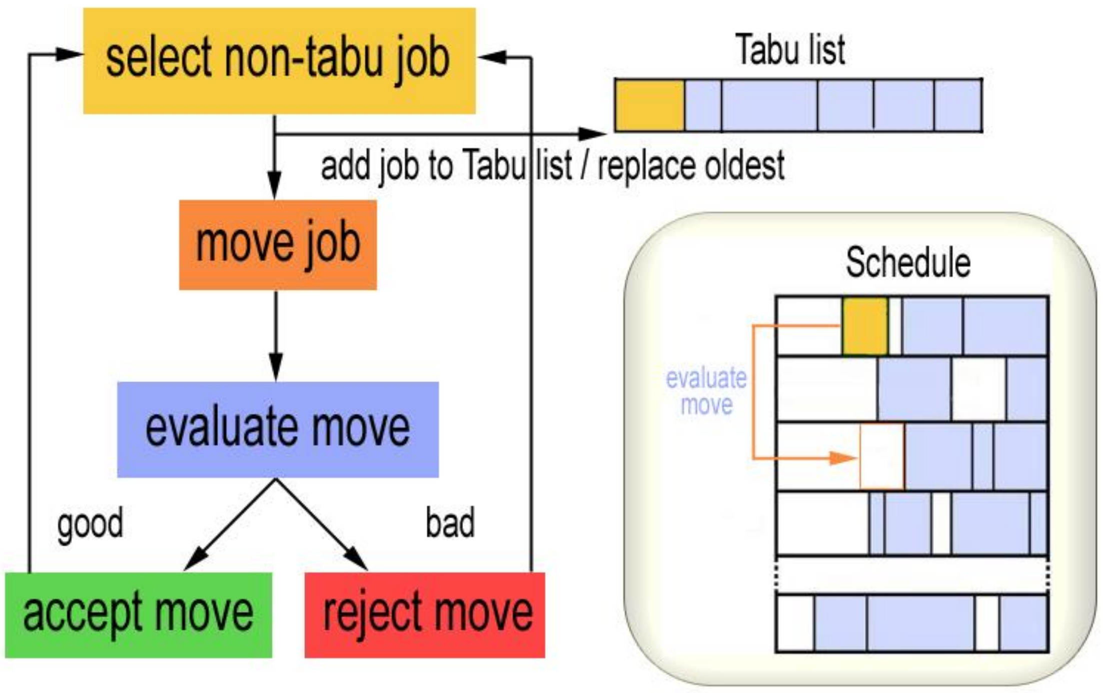

Figure 2.2 Tabu Search [16]

Researchers have used tabu search heuristics to find quality solutions for a variety of scheduling problems. For instance, tabu search has been used to schedule shifts for nurses and constrain solutions based on the number of shifts a nurse has worked as well as the number of nurses needed at a given time [7]. Tabu search has also been used to schedule tournaments for various sports leagues. For this application, the number of home games versus the number of away games must be balanced for every team. In addition, constraints are needed to see if the chosen teams and facility are all available at the specified time [12]. Tabu search has also been used in job shop scheduling [6].

For tabu search, the downfall is storage. If not enough storage is provided in the list of solutions the model may backtrack or create a loop. If too many solutions are stored the process becomes time consuming. The time will also increase if the stored data has a complex data structure. Simplifying the string of data will decrease the time, but increase the chances of looping. One must find a balance between the two to obtain quality results.

2.3. Variable Neighborhood Search Heuristic

The heuristic created in this thesis is a new type of variable neighborhood search heuristic (VNS). VNS is used to escape the local optimal solution by using various neighborhood searches. It was suggested by Mladenovi´c and Hansen in 1997 [27]. With this method the heuristic cycles between searches by changing the neighborhood. This allows for a wide variety of search techniques to be used in a single problem. The goal of VNS is to escape the less optimal peaks and valleys by changing neighborhoods.

VNS can be applied to a variety of instances [13], [20], [31], [34]. For instance, VNS can be applied in a hill climbing setting. To initialize the search, one must start in a given neighborhood. A neighborhood is denoted as k with $k = 1..k_{max}$. For $k <= k_{max}$ the same steps are followed. Next, a random solution denoted by x' must be found in neighborhood k. Then, a local search should determine the local optimal denoted as x''. Finally, the solution is judged. If x'' is better than x', $x' = x''$ and the search continues with the new x' neighborhood. If x'' is worse than x', then the neighborhood is increased by one (k=k+1) and the process continues.

VNS has been applied to single machine shops. For this case, the researchers used two neighborhoods in order to have faster computation. The neighborhoods were swap and insertion. Swap randomly locates two jobs and switches each location for the other. Insertion also identifies two random jobs. Then, it places one of the jobs directly before the other job [21].

Variable neighborhood search heuristic has also been applied to JSS. In this instance, the researchers used three neighborhoods. Each neighborhood was a different insertion method. The insertion point is based on random numbers and the heuristic cycles through the neighborhoods as local optimal solutions are found [30]

VNS can be altered to fit any scenario. To apply VNS, three things should be determined. First, how many neighborhood structures should be used. If one is looking to minimize the computation time, fewer neighborhoods should be used. Second, in what order should the neighborhood searches be performed. Lastly, what strategy should be used to change the neighborhood. Since the applications for this heuristic are vast, the complexity and optimality of the solution will depend on the individual instances.

A primary weakness of VNS techniques is the lack of structure to select which neighborhood should find the next solution. The next chapter introduces a new technique to switch between neighborhood searches. This switching is based upon the progression of the ball and players in a fútbol game.

Chapter 3: The SOCCER Heuristic

Based on the author's research, variable neighborhood search heuristic has switched between neighborhoods primarily based upon random or cyclic selections and lacks a complex structure for picking neighborhoods. The SOCCER heuristic - fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results - is a new framework to perform variable neighborhood search heuristic and mimics a fútbol game. Instead of randomly choosing neighborhoods, neighborhoods are selected based on fútbol players, their positions, and the ball's position on the field. This chapter begins with an explanation of fútbol with the intention of motivating an optimization heuristic. The final section discusses how to transform fútbol into a variable neighborhood search heuristic for an optimization problem.

3.1. Fútbol

It is assumed that the reader is familiar with the game of fútbol, which is called soccer in the United States. Therefore, only limited information regarding the game is presented here. In fútbol, the objective is to score more than the opposing team. In order to score, one must get the ball past the goal line and inside of the goal. Several methods can be used to move the ball. The primary methods are dribbling, passing, clearing and punting.

The purpose of dribbling is for a player to move the ball with the intention of that player being the next player to touch the ball. The majority of dribbles are short kicks. From a strategic standpoint more dribbling occurs near the opponent's goal than near one's own goal.

The purpose of passing is to play the ball with the intent of another player on the same team being the next player to touch the ball. One primarily passes the ball with the feet or head. If one passes the ball with his head it is called a header. The majority of passing occurs in the midfield.

The purpose of clearing is to move the ball as far away from one's own goal as possible. Thus, a player clearing a ball either kicks or heads the ball as far as possible. No consideration is given as to whom may receive this ball. Clearing is primarily done on the defensive end of the field.

Only the goalie can punt the ball since the goalie is the only player that can touch a ball with his hands. To punt, the goalie drop kicks the ball and the result is a long kick. Typically, the goalie punts the ball after catching the opposing team's attempt to score.

Scoring a goal only requires the ball to cross the end line inside of the goal. Players may dribble, pass, clear, or punt the ball into the goal. Occasionally a player may even score on their own team, which is the humiliating own goal. Goals are hard to come by in fútbol and a goal typically has a substantial celebration.

For professional fútbol, each team has eleven players on the field. The positions are defender, midfielder, striker, and goalie. A player's method for moving the ball is highly dependent on the assigned position since each position has a certain responsibility. The manager's strategy largely determines the number of players assigned to each position.

Some of the typical team formations are shown in Figure 3.1. However, before one can understand the formations one must first understand the field. In the image below, there are three fields. Standard fútbol fields have the same lines asthe fields depicted in the image. Anything outside the scope of the field is considered out of bounds. The top most edge of the image is the end line. The goal is centered on the field and adjacent to the end line. The small box located at the top center of the field is the 6 yard box. The larger box is the 18 yard box. The line across the center of the field is called midfield. The bottom half of the image is simply a mirrored version of the top half and are referenced the same way.

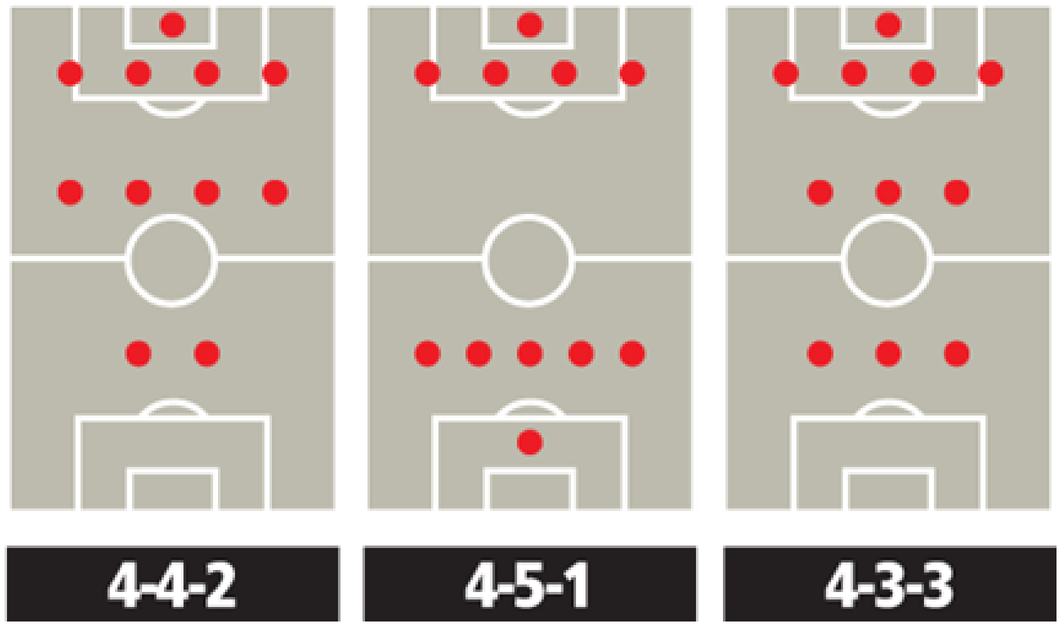

Figure 3.1 Fútbol Formations [11]

In fútbol, there are several different formations managers typically use. Three example formations are shown in Figure 3.1. Each formation has a corresponding three-digit number. The first number is the number of defenders per team, the second is the midfielders, and the third is the strikers. The goalie is neglected when providing this number formation. To further explain the formations, each player is discussed below according to a 4 3 3 formation.

The player at the top of the image is the goalie. Each team has one goalie. The goalie can only use his hands inside of the 18-yard box. Thus, the goalie is not likely to move far and is considered a slow player. A goalie's job is to block shots and prevent the other team from scoring. If the opposing team takes a shot and the goalie catches it, then the goalie punts the ball.

The next set of players is the defenders. Four defenders are used per team with this formation. Defenders are located between the opposing team's end line and midfield. When the ball is close, defenders are quick. However, defenders are relatively slow when the ball is far away. Their objective is to stop opposing players from scoring and to move the ball to the other end of the field. When defenders become desperate, they clear the ball.

The next tier is the midfielders. For this strategy, each team consists of three midfielders. These players are typically located between the 18-yard boxes. Midfielders move at an average pace, which is fairly constant regardless of the ball's location. Their job is to move the ball from the defensive end to the offensive end. Typically this is done by passing the ball.

The final set of players is the strikers. On the field, strikers are located near the opposing team's defenders. Strikers are incredibly fast when the ball is close, but are slow and often lazy when the ball is farther away. Typically they dribble the ball. Strikers are also more likely to score than the other positions.

3.2. Transforming Fútbol into an Variable Neighbor Search Heuristic

The SOCCER heuristic - fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results - is inspired by fútbol and is a new type of variable neighborhood search heuristic. The heuristic mimics a fútbol game. The ball's location is related to the objective function of the current solution. Each player is assigned a neighborhood. When a player approaches the ball he is given the opportunity to use his neighborhood search. The offensive team performs neighborhood searches in hopes of improving the solution, while the defensive team performs searches in hopes of finding worse solutions. The defensive team and their pursuit of worse solutions helps the SOCCER heuristic avoid being stuck at a locally optimal solution. A goal is scored if a new best solution is found.

Fútbol was chosen as inspiration for this heuristic because of the dynamic aspects it allows. With two opposing teams, one is able to search for quality solutions while occasionally accepting worse solutions, thus escaping local optimal solutions. It also allows for a variety of neighborhood searches to be used. The neighborhood search selected is based on each player's proximity to the ball and speed. This allows for a more flexible search instead of simply cycling through neighborhoods as variable neighborhood search heuristic techniques have done in the past. The SOCCER heuristic explained here assumes one is seeking to minimize the objective function. The reader can make the obvious changes for a maximization problem.

The SOCCER heuristic starts by initializing the players and their characteristics. Then, it searches for new solutions. Each time the ball is kicked, the players move since fútbol players are rarely stationary. When a new best solution is found, a goal is scored and a celebration dance occurs.

The SOCCER heuristic requires that each team is given q players. The offensive team is denoted by $O = \{p_1, \ldots, p_q\}$ and the defensive team is $D = \{p_{q+1}, \ldots, p_{2q}\}$. Each player is provided with a starting width location (w_i), and a starting height scalar (α_i), an athletic ability consisting of a slow (v_{s_i}) and fast (v_{f_i}) velocity, and a neighborhood (N_i). The height scalars are typically constant among similar positional players such that $0 \le \alpha_{goalie_{defense}} < \alpha_{defender_{defense}} < \alpha_{midfielder_{defense}} < \alpha_{striker_{defense}} < 1 = \alpha_{striker_{offense}} < \alpha_{midfielder_{offense}} < \alpha_{defender_{offense}} < \alpha_{goalie_{offense}}$. These characteristics should be based upon the player's position: goalie, defender, midfielder, and striker.

The SOCCER heuristic begins with a feasible solution and a corresponding z value. The y location of the ball is always the z value of the current solution. The x location can be based upon other criteria and here the x location is the difference between the z values of the two most recent solutions. To start, the ball is placed at location $(0,z)$. Observe that the ball may go negative in the x direction and this merely indicates that the ball is on the left side of the field.

The first player discussed is the offensive goalie. The goalie's initial position is $(0, z\alpha_{goalie_{offense}})$. A goalie is the team's last defense before a goal is scored. This player uses either an extremely large neighborhood or may find a new starting solution that is not related to the current solution. A goalie solution is always taken whether or not the solution is better. This neighborhood search occurs when the defensive team kicked the ball too many times and the ball needs to move to a new solution. For this reason, the goalie is only allowed to play the ball if the ball is closer than some threshold to their starting position.

The four defenders are located on the defensive end of the field away from quality solutions (x_i, $z\alpha_{defender_{offense}}$). Since there are four defenders, the x_i values typically have two negative and two positive values with an average of 0. Defenders are often fast players when near the ball, yet slow when away from the ball. This type of player has a large neighborhood, which corresponds to a clearance. This means that the change between solutions is significant.

Midfielders start with the coordinates (x_i, $z\alpha_{midfielder_{offense}}$). Since there are three midfielders, these x_i values typically have one negative, one positive and a 0 value with an average of 0. This type of player is of average speed, but is not slow either. Midfielders should have medium sized neighborhoods, which corresponds to a pass.

Strikers start with the coordinates (x_i, $z\alpha_{striker_{offense}}$). Since there are three strikers, these x_i values typically have one negative, one positive and a 0 value with an average of 0. Strikers are typically fast and extra slow. Thus, their v_{f_i} should be among the fastest on the team, but the v_{s_i} should be slower than most, which translates into the lazy striker adage. Strikers have small neighborhoods, which corresponds to a dribble.

The defensive team has the same types of players and the same neighborhoods as mentioned above. The starting locations are similar except that the y values are smaller than z. This means that these players have initial locations, which are better than the best known solution. Therefore, the offensive team is trying to move the ball towards the defensive end. Since the defensive team is attempting to find worse solutions, their speeds should be slower than the speed of the offensive team and thus they should play the ball less frequently. One never desires the defensive goalie to drastically move from an extremely good solution, so the defensive goalie's fast and slow speed are both 0.

The main step of the SOCCER heuristic involves players kicking the ball. The first step in this process is to determine which player approaches the ball first. This player is selected based on each player's proximity to the ball and each player's fast velocity. The player who can reach the ball in the shortest time is selected. Next, he attempts to kick the ball. This attempt is accomplished by finding a new solution according to the player's neighborhood.

For an offensive player, if the *z* value of the neighboring solution is better than the current solution, the ball is kicked. With this kick, the current solution and *z* values are updated to equal this new solution. Then, the new solution is compared to the best solution. If it is better, the best solution is updated to be the current solution and a goal has been scored. When a goal is scored, The *Celebrate Goal* subroutine is called.

For a defensive player, the ball is kicked if the *z* value of the neighboring solution is worse than the current solution. If this occurs, the current solution and *z* values are updated to this neighboring solution. In some instances, a defensive player's neighborhood search will result in a better *z* value than the best solution. If this is the case, an own goal was scored. The best solution is updated to equal its values and a celebration by the offense occurs.

If the new solution was not accepted, the current player miss handled the ball. As such, another player attempts to kick the ball. To achieve this, the first player to reach the ball is marked. This mark is simply to track which players have already missed the ball. The first unmarked player to approach the ball is selected and this entire process continues. If all players except the goalies are marked without successfully kicking the ball, the marks are cleared and the process continues.

Once a player, say p_j, is identified as having kicked the ball, all players' positions are first updated. In fútbol players adjust their position according to the location of the ball. Thus, players are moved before the location of the ball is updated. The act of the player kicking the ball is one iteration.

The first step in moving players is to determine the time required by the kicking player to reach the ball. This is merely the distance between the player and the ball divided by the players fast velocity, *time* = $\frac{\sqrt{(b_x - x_j)^2 + (b_y - y_j)^2}}{vf_j}$. Every player moves toward the ball according to either their fast or slow speed. If the player is closer to the ball than a specified threshold, the player's fast velocity (v_{f_i}) is used. Otherwise, the slow velocity (v_{s_i}) is used. Thus, the player's current x_i position becomes

$$x_i + time(v)\ \frac{b_x - x_i}{\sqrt{(x_i - b_x)^2 + (y_i - b_y)^2}},$$

and the yi location is updated to

$$y_i + time(v)\ \frac{b_y - y_i}{\sqrt{(x_i - b_x)^2 + (y_i - b_y)^2}}.$$

where *v* is either the fast or slow velocity. This repositioning may imply that a player over ran the ball.

When all players have moved, the balls *x* and *y* location is updated. The *y* location becomes the *z* value of the neighboring solution. Various options are available for an *x* location. If there exists a secondary objective function, this would be an excellent candidate for an *x* value. If this does not exist, it is recommended to let the ball's *x* location be the difference between objective value of the current solution and the objective value of the neighboring solution. This strategy tends to have the defensive try to keep the ball on the left side of the field and the offense keeps the ball on the right side of the field.

When the team scores a goal, a celebration occurs. The celebration is used to check various neighboring solutions to see if a better solution is nearby. This celebration is a short hill climbing heuristic that seeks to find a locally optimal solution near the new best solution. Some players are selected to celebrate. Each player can cheer for a short duration or an extended amount of time. The celebrating player finds a neighboring solution and if this is better than the best, the player carries the ball to this new solution. Once the current player is done celebrating, the next player in the celebration does their dance. Any player can join the celebration and players may celebrate multiple times. At the end of the goal celebration the best solution becomes the current solution and the field (ball and players) is reset according to the initial parameters. That is, for offensive players, all strikers are returned to their initial position of (x_i, $z\alpha_{striker_{offense}}$), midfielders to ($x_i$, $z\alpha_{midfielder_{offense}}$), defenders to ($x_i$, $z\alpha_{defender_{offense}}$), and the goalie to ($x_i$, $z\alpha_{goalie_{offense}}\ z$). The defensive players return to their initial positions as well. Please observe that the length of the field has changed with this repositioning.

Several inputs are needed for the SOCCER heuristic. Each fútbol player needs five characteristics – a starting width (w_i), a scaling height (α_i), a well-defined neighborhood (N_i), a fast velocity (v_{f_i}), and a slow velocity (v_{s_i}). Several other parameters are needed as input. These parameters include: *Threshold, MaxIterations, Number of Celebrating Players,* and *Number of Celebrations*.

The width, scaling height and speed values of the players should be chosen to mimic a fútbol alignment. Thus, the average widths for all strikers, midfielders, defenders and goalies on each team should be 0. The scaling heights for the defensive players should follow the relationship, $0 \le \alpha_{goalie_{defense}} < \alpha_{defender_{defense}} < \alpha_{midfielder_{defense}} < \alpha_{striker_{defense}} < 1$. The scaling heights for the offensive players should follow the relationship, $1 \le \alpha_{striker_{offense}} < \alpha_{midfielder_{offense}} < \alpha_{defender_{offense}} < \alpha_{goalie_{offense}}$. The neighborhoods should also mimic the player's responsibilities with strikers, midfielders, defenders and goalies having small, medium, large and extremely large neighborhoods, respectively.

The pseudocode for the SOCCER heuristic is as follows.

Fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results (The SOCCER Heuristic)

```
Let $X$ be a feasible solution with objective value $z$.
Let ball be an initial position $(b_x, b_y) \leftarrow (0,z)$ of the ball.
$X^* \leftarrow X, z^* \leftarrow z, z^{old} \leftarrow z, iterations \leftarrow 0$
Position Players ($z$)
While iterations<MaxIterations Do
    Kick ← 0, Unmark all players
    Mark defensive goalie
    If $\sqrt{(x_{Goalie_{Offense}} - b_x)^2 + (y_{Goalie_{Offense}} - b_y)^2} > Threshold$ Then
        Mark offensive goalie
```

```
End If
While ($Kick$ = 0) do
    Let $p_i$ be the quickest unmarked player to the ball $\frac{\sqrt{(b_x - x_i)^2 + (b_y - y_i)^2}}{\mathrm{vf_i}}$
    Time $\leftarrow \frac{\sqrt{(b_x - x_i)^2 + (b_y - y_i)^2}}{\mathrm{vf_i}}$
    Let $X' \epsilon N_{p_i}(X)$ with objective value $z'$
    If $p_i$ is on offense and $z' \leq b_y$ Then
        Move Players (Time, ball)
        $X \leftarrow X'$, $b_y \leftarrow z'$, $b_x \leftarrow z' - z^{old}$
        $z^{old} \leftarrow z'$
        Kick $\leftarrow 1$
        If $z' < z^*$ Then
            $X^* \leftarrow X$, $z^* \leftarrow z'$
            Celebrate Goal ($X^*$)
            Position Players ($z^*$)
            $z^{old} \leftarrow z^*$
        End If
    End If
    If $p_i$ is on defense and $z' \geq z_x$ Then
        Move Players (Time, ball)
        $X \leftarrow X'$, $b_y \leftarrow z'$, $b_x \leftarrow z' - z^{old}$
        $z^{old} \leftarrow z'$
        Kick $\leftarrow 1$
```

```
                End IF
                If $p_i$ is on defense and If $z' < z^*$ Then
                    $X^* \leftarrow X$, $z^* \leftarrow z'$
                    Celebrate Goal ($X^*$)
                    Position Players ($z^*$)
                    $z^{old} \leftarrow z^*$
                    Kick $\leftarrow 1$
                End If
                If Kick = 0 Then
                    Mark player $p_i$
                    If all player's marked, Then
                        Unmark all players except the goalies
                    End If
                End If
            End While
            iterations $\leftarrow$ iterations+1
        End While
        Report $X^*$, $z^*$
```

The SOCCER heuristic subroutine *Position Player* (z) initializes each player's starting location on the field. The inputs necessary for this subroutine are the starting widths of the players and the scaling height coefficients of the players. This function is called during the initialization and after celebrating a goal. This function requires the current *z* value and thus the starting position of the players improves each time a goal is scored. Therefore, this fútbol field is not static in size.

Position Players (*z*)

```
For i = 1 to 2q
        $x_i \leftarrow w_i$
        $y_i \leftarrow \alpha_i z$
End For
Report $x_i, y_i$ for each player $p_i$
```

The next subroutine provides the pseudo-code for moving player. Players are moved based on their speed and the time allowed for the move. If the player is within the specified *Threshold*, he moves according to his fast velocity. Otherwise, he moves according to his slow velocity. Each player moves in the direction of the ball and may even run past it. Each of these parameters may be altered to fit various optimization problems. The pseudocode is provided next.

Move Players (*time*, *ball*)

```
For i = 1 to number players begin
    If $\sqrt{(x_i - b_x)^2 + (y_i - b_y)^2}$ > Threshold Then
        $x_i \leftarrow x_i + time * v_{s_i} \frac{b_x - x_i}{\sqrt{(x_i-b_x)^2+(y_i-b_y)^2}}$
        $y_i \leftarrow y_i + time * v_{s_i} \frac{b_y - y_i}{\sqrt{(x_i-b_x)^2+(y_i-b_y)^2}}$
    Else
        $x_i \leftarrow x_i + time * v_{f_i} \frac{b_x - x_i}{\sqrt{(x_i - b_x)^2 + (y_i - b_y)^2}}$
        $y_i \leftarrow y_i + time * v_{f_i} \frac{b_y - y_i}{\sqrt{(x_i - b_x)^2 + (y_i - b_y)^2}}$
    End If
End For
Report $x_i$, $y_i$ for each player $p_i$
```

In fútbol, few goals are scored. Therefore, when a goal is scored, a celebration dance occurs. The player that scored the goal dances first. The number of dances per celebrating participants and the number of participants is determined by the user. The optimization goal is to search several neighboring solutions in an effort to find an even better best solution that is locally optimal. The pseudocode for this sub routine is as follows.

Celebration Dance (X^*)

```
    For $i$ = 1 to Number of Celebrating Players Begin
        For $j$ = 1 to Number of Celebrations Begin
            Let $X' \epsilon N_{p_i}(X^*)$ with objective value $z'$.
            If $z' < z^*$ Then
                $X^* \leftarrow X'$, $z^* \leftarrow z'$
            End If
        End For
```

```
End For
Report $X^*$, $z^*$
```

The SOCCER heuristic can be altered to fit a variety of optimization problems. By following the steps above, one could implement this heuristic on numerous classes of optimization problems. One would simply have to alter the parameters and objective function to customize the heuristic to the scenario. In the next chapter, the SOCCER heuristic is transformed into a job shop scheduling heuristic.

Chapter 4: The SOCCER Heuristic for Job Shop Scheduling

Chapter 4 starts by providing background information on a job shop scheduling instance occurring at a limestone company. The next section converts the SOCCER heuristic into the SOCCER heuristic for a job shop scheduling problem. This conversion uses assumptions based on machines and processes that are relevant to the limestone company. The chapter concludes with a computational study and results for randomized JSS instances.

4.1 Problem Specifics

The limestone company has many quarries throughout the Midwest and a centralized manufacturing facility. As the company continues to grow, production scheduling has become increasingly complex. This section explains this job shop scheduling so one can better understand the steps taken and the computational complications of this instance.

The limestone company has numerous standard product types. Each of which is offered in varying dimensions, finishes, and stone colors. The finishes offered provide the customer with the desired texture and aesthetic appearance for the stone product. The available finishes include brushed, blasted, bush hammered, tumbled, honed, and split. The stone colors can also be specified. These colors consist of cottonwood, plaza grey, prairie shell, and numerous others. Each standard product has a predetermined processing route based on the product type and the desired finish. Some examples of standard products are landscape blocks, thin veneer, and pavers. In total, over 1.2 million standard product combinations exist.

Custom products are also available. These products range in size and shape, according to the customer's desire. Examples of custom pieces include a kitchen sink and a cross for a church. One can even add a desired finish or vary the stone type for further customizations of the piece. Since each custom piece is vastly different, the processing route must be created for each piece. This path must be created before the piece can be scheduled.

The limestone company has seven areas for production in its facility. Every product starts in the belt saw area. The belt saws convert rough stone into slabs with rough edges. The factory moves the equivalent of one slab to different areas according to the processing route of the product. These seven areas have between one machine and four parallel machines. Thus, a schedule consists of an assignment of customers' slabs to machines to be processed between certain times.

The information above provides a base knowledge of the limestone company and its production facility. This information is needed as framework before implementing the SOCCER heuristic for this company. The next section converts the SOCCER heuristic to fit job shop scheduling instances.

4.2 The SOCCER Heuristic to JSS

The SOCCER heuristic starts by finding a feasible solution *X*. This feasible solution is created by scheduling each job in sequential order. This means that the job for the first customer is scheduled first, then the job for the second customer. This continues until every slab for every customer has been scheduled.

To schedule jobs, the number of slabs needed per order is first determined. This process starts by generating a random block. Each dimension is generated uniformly between the specified measurements. The width of the product is then subtracted from the length of the block. This cut stone is referred to as a slab. Slabs are cut from this block until the square footage requirement for the order has been met or the block is no longer wide enough to meet the requirement. For the latter, a new block is generated and the process continues. This process determines the number of slabs needed to fill the order. During production, slabs are cut further to produce pieces. The pieces have the dimensions of the final product. A slab can contain between one and 100 pieces; therefore, all pieces associated with a given slab move through the facility together.

Slabs move through production based on its predetermined product processing route. Typically, a product starts on the belt saw. This limestone company currently has three belt saws running in parallel. Slabs are scheduled on the machine with the earliest available start time. This methodology is used for all areas with parallel machines.

The processing time for a slab varies for each area. The slab’s processing time in an area is based on the dimensions of the slab, the dimensions of the end product, the product type, and the stone type. In some instances, the processing time may even vary between machines in the same area.

Once a slab is completed on a machine, it incurs an estimated five minute transit time. Therefore, the slab’s earliest possible start time on the next machine is the end time from the previous process plus five minutes. If a machine is available, the slab can be scheduled for that time. If all machines in the area are being utilized, the slab is scheduled on the machine with the earliest available start time. If there is only one machine in the area, the slab is scheduled immediately after the last currently scheduled slab on the machine finishes production. It is important to note that a slab can only be scheduled on one machine at a time and each machine can only process one slab at a time.

As mentioned in Chapter 2, numerous objective functions exist for JSS instances. In this instance, the objective function is to minimize the makespan. The makespan is the time when the last slab is completed. Thus, every machine is idle. This objective function also minimizes the cumulative amount of idle time on all machines. This objective function is denoted as *z*, which is reported in minutes.

Once the initial solution is created, the heuristics uses the ball and players to alter the solution. The ball represents the initial solution with an initial *x* location of 0 and an initial *y* location of *z*. Each team has eleven players. All players have a neighborhood, starting *x* and *y* locations, a slow velocity and a fast velocity. The characteristics for each player are based on the position on the field. The positions are goalie, defender, midfielder, and striker.

Each team has one goalie. The offensive goalie has an initial *x* position of 0 and an initial *y* position of 1.75 times the *z* value. Typically, the goalie does not move very far. For this scenario, the offensive goalie should only kick the ball if the solution is very bad and is not improving. Therefore, his slow speed is set to 0 and the fast speed is set to 3. This prohibits the goalie from drifting towards quality solutions.

The offensive goalie's neighborhood is a scramble of jobs which corresponds to the goalie punting the ball far away from its current location. This neighborhood search is used if too many worse solutions are found; therefore, an offensive goalie's new solution is always accepted regardless of the *z* value. In this search, the order of customers in the customer list is shuffled by randomly swapping each customer with a different customer. Then, the jobs are scheduled based on this random order of the customers, thus providing a new solution and corresponding *z*-value.

On offense, the four defenders have starting *x* positions of -75, -25, 25, and 75. Their starting *y* positions are 1.5 times the current *z* value. They have a slow speed of 4 and a fast speed 16.

The defender's neighborhood randomly selects two areas. For each of these areas, a selection of slabs is moved on every machine in the area. The set of slabs is chosen based on the largest idle time between scheduled slabs. The selected slab and all consecutive slabs following it for that customer are moved. The slabs may be inserted on any machine in the area. The entry point is the earliest available time on the machine with the most idle time. This is a large neighborhood and corresponds to a clearance.

On offense, the three midfielders are in a line centered across the width of the field. Their starting *x* positions are -50, 0, and 50. The initial *y* positions are 1.25 times the *z* value. The slow speed for these players is 7 and their fast speed is 12.

The midfielder's neighborhood randomly selects one area. Then, a set of slabs is identified and moved. The chosen slab is one which has the largest idle time between it and the previous slab. This slab and all consecutive slabs following it for the customer are selected. Then, this selection of jobs is moved to the machine with the most idle time and inserted at the first available time. This would be a small neighborhood. To make this a medium neighborhood, which is equivalent to a pass, this is repeated for each machine in the area.

For offensive strikers, the starting *x* locations are -30, 0, and 30. The starting *y* location for each player is equal to 1.1 times the *z* value. The slow speed of each is 5 and the fast speed is 20.

The striker's neighborhood randomly selects an area and a machine. Then, for the specified machine in the given area, the slab with the largest idle time between it and the previous slab is chosen. The slab and all consecutive slabs following it for the same customer are selected with a maximum of 30 slabs selected. The slabs are inserted in the first available time on the same machine they were removed from. If the removed and inserted slab locations are the same, a new area and machine are selected. This small neighborhood mimics dribbling.

For the defensive team, the goalie has an initial *x* position of 0. His initial *y* position is 0.25 *z*. Since goalies typically do not move far, this goalie has a fast and slow speed of 0. If the solution is close to this goalie's position, he does not act. This would be a very good solution and one does not want to move far away from here. Therefore, the defensive goalie is not given a speed or a neighborhood search.

For defensive players, defenders have starting *x* locations of -75, -25, 25, and 75. The initial *y* location is 0.5 times the initial *z* value. Defensive players are slower than offensive players; therefore, defenders on defense have a slow speed of 3 and a fast speed of 12. Defenders have the same neighborhood regardless of the team.

The midfielders on defense have starting *x* positions of -50, 0, and 50. The initial *y* positions are 0.75 times the current *z* value. Since defensive players are slower, the slow speed for midfielders is 6 and the fast speed is 10. Midfielders on defense have the same neighborhood search as the midfielders on offense.

The defensive strikers have starting *x* locations of -30, 0, and 30. The starting *y* positions are 0.9 times the current *z* value. They have a slow speed of 4 and a fast speed of 14. The strikers on defense have the same neighborhood as those on offense.

A goal is scored when the solution found is better than the best solution. Once a goal is scored, a celebration dance occurs. This dance consists of four celebrating players. Each player dances ten times. Each dance is a different iteration of the player's neighborhood search. If a better solution is found, the next dance starts with this solution. The purpose of the celebration dance is to find a locally optimal solution in the current region. This is viewed as a hill climbing routine.

4.3. Computational Study

To test the SOCCER heuristic, small, medium and large instances were created. The small instances each had 50 customers, the medium had 100 and the large had 500 customers. The instances were randomly created and to avoid random anomalies 20 instances in each class were created. The template for these random instances followed a limestone company's product line.

The following method was used to generate a random instance. Each customer randomly selected one of the 1.2 million product combinations. The number of pallets was selected by randomly generating an integer between one and fifty. This number corresponds to the number of pallets in the order. The route that this customer's product takes through the factory is dependent upon the product ordered. Some paths only required a single area and the longest path visits 7 different areas. There were no release dates and any of the orders could be started at any time. The objective is to minimize the makespan, which also reduces idle time.

To determine the importance of the length of the fútbol game, each instance was run with 100, 1,000 and 10,000 kicks. Eventually, the ball will be kicked, but it may take several attempts as offensive players may find worse solutions and defensive players find a better, but not a new best, solution. Each kick equates to one iteration.

All runs were performed on a PC with a 3.4 GHz Intel Core i7-2600 CPU that had 4 GB of RAM. The primary statistics recorded for each run is the makespan for the initial schedule, the makespan for the best schedule, and the time in seconds required for the SOCCER heuristic. Tables 4.1-4.3 provide these numbers for all 20 instances of each data set. Thus, the SOCCER heuristic was tested on 180 instances.

The data collected for 50 customers is in Table 4.1 with the *z* value represented in minutes and the time required by the SOCCER heuristic provided in seconds. For this data set, when set to run for 100 kicks, solution improved 60% of the time. In the remaining 40% the best solution was the initial solution. The SOCCER heuristic ran with an average 1.9 seconds per data set and had an average 6.24% improvement. For 1,000 kicks, 85% of the solutions improved. On average, it took 20.3 seconds to complete one data set. The solutions improved by an average of 8.42%. When the SOCCER heuristic was run for 10,000 kicks, 95% of the solutions were improved. It took an average of 3.6 minutes to complete one instance and had an average 9.17% improvement.

Table 4.1 Various Iterations for 50 Customers

		50 Customers								
		100 Kicks			1,000 Kicks			10,000 Kicks		
Data	Initial Z	Best	% Improvement	Time	Best	% Improvement	Time	Best	% Improvement	Time
1	3,097	3,097	0.00%	0	2,921	5.68%	7	2,834	8.49%	81
2	5,797	5,503	5.07%	2	5,503	5.07%	26	5,369	7.38%	333
3	5,459	5,103	6.52%	3	5,103	6.52%	23	5,103	6.52%	242
4	38,458	36,440	5.25%	3	36,328	5.54%	47	36,328	5.54%	502
5	4,748	4,748	0.00%	2	4,604	3.03%	20	4,604	3.03%	227
6	6,030	5,922	1.79%	2	5,244	13.03%	23	5,132	14.89%	236
7	3,654	3,654	0.00%	0	3,297	9.77%	6	3,245	11.19%	70
8	5,582	5,088	8.85%	2	4,870	12.76%	17	4,773	14.49%	146
9	2,173	1,928	11.27%	0	1,727	20.52%	4	1,727	20.52%	39
10	5,138	5,138	0.00%	2	5,138	0.00%	15	5,138	0.00%	159
11	4,631	4,631	0.00%	2	4,631	0.00%	22	4,576	1.19%	225
12	4,200	4,197	0.07%	2	4,094	2.52%	14	4,067	3.17%	151
13	4,395	4,395	0.00%	1	4,395	0.00%	10	4,279	2.64%	127
14	8,048	5,612	30.27%	1	5,612	30.27%	15	5,334	33.72%	164
15	6,650	6,121	7.95%	3	5,983	10.03%	29	5,937	10.72%	295
16	4,940	4,555	7.79%	2	3,705	25.00%	16	3,705	25.00%	155
17	5,483	5,483	0.00%	1	5,025	8.35%	15	4,902	10.60%	171
18	51,795	47,968	7.39%	7	47,896	7.53%	70	47,896	7.53%	852
19	3,344	3,344	0.00%	1	3,177	4.99%	10	3,050	8.79%	92
20	5,582	5,088	8.85%	2	4,870	12.76%	17	4,773	14.49%	146
Total	179,204	168,015	6.24%	1.9	164,123	8.42%	20.3	162,772	9.17%	220.65

The data collected from the 100 customers' data is in Table 4.2. When set to run for 100 kicks, the initial solution improved 80% of the time. The data set took an average of 7.35 seconds to run and had an average improvement of 4.83%. When set to run for 1,000 kicks, 90% of the solutions improved. On average, it took 77 seconds to complete one data set. The solutions improved by an average of 7.02%. When the SOCCER heuristic was set to run for 10,000 kicks, 90% of the solutions improved. It took an average of 14 minutes to complete one instance and had an average 8.67% improvement.

Table 4.2 Various Iterations for 100 Customers

		100 Customers								
		100 Kicks			1,000 Kicks			10,000 Kicks		
Data	Initial Z	Best	% Improvement	Time	Best	% Improvement	Time	Best	% Improvement	Time
1	10,524	9,884	6.08%	7	9,861	6.30%	62	9,706	7.77%	718
2	48,087	47,011	2.24%	15	47,011	2.24%	196	47,011	2.24%	1,981
3	10,446	10,446	0.00%	9	10,367	0.76%	93	10,320	1.21%	1,207
4	7,313	7,091	3.04%	4	6,549	10.45%	45	6,549	10.45%	476
5	22,948	19,412	15.41%	14	19,347	15.69%	126	19,347	15.69%	1,184
6	9,914	9,474	4.44%	5	8,909	10.14%	70	8,360	15.67%	663
7	8,538	8,485	0.62%	4	8,435	1.21%	42	8,258	3.28%	580
8	9,479	9,022	4.82%	9	8,865	6.48%	93	8,353	11.88%	918
9	7,081	6,807	3.87%	3	6,605	6.72%	35	6,437	9.09%	410
10	10,939	10,939	0.00%	8	9,409	13.99%	76	9,201	15.89%	755
11	59,106	59,106	0.00%	14	59,106	0.00%	133	59,106	0.00%	1,523
12	15,639	10,695	31.61%	11	10,682	31.70%	85	10,682	31.70%	843
13	9,217	8,845	4.04%	4	8,820	4.31%	45	8,734	5.24%	533
14	10,264	10,058	2.01%	6	9,938	3.18%	84	9,933	3.22%	988
15	9,714	9,581	1.37%	7	9,197	5.32%	63	9,108	6.24%	587
16	7,697	7,697	0.00%	4	7,697	0.00%	40	7,697	0.00%	487
17	6,977	6,916	0.87%	3	6,916	0.87%	35	6,916	0.87%	386
18	8,939	8,399	6.04%	6	8,244	7.77%	71	7,915	11.46%	715
19	8,935	8,453	5.39%	5	8,318	6.91%	54	8,083	9.54%	625
20	9,479	9,022	4.82%	9	8,865	6.48%	93	8,353	11.88%	918
Total	291,236	277,343	4.77%	7.35	273,141	6.21%	77.05	270,069	7.27%	824.85

Table 4.3 presents the data collected for the 500 customers' data sets. When set to run for 100 kicks, the initial solution improved 90% of the time. The data set took an average of 2.67 minutes to run and had an average improvement of 6.47%. When set to run for 1,000 kicks, 100% of the solutions improved. On average, it took 29 minutes to complete one data set. The solutions improved by an average of 7.77%. When the SOCCER heuristic was set to run for 10,000 kicks, 100% of the solutions improved. It took an average of 5.37 hours to complete one instance and had an average 8.02% improvement.

Table 4.3 Various Iterations for 500 Customers

		500 Customers								
		100 Kicks			1,000 Kicks			10,000 Kicks		
Data	Initial Z	Best	% Improvement	Time	Best	% Improvement	Time	Best	% Improvement	Time
1	47,837	47,568	0.56%	166	47,345	1.03%	2,078	47,345	1.03%	23,722
2	44,184	44,184	0.00%	124	43,871	0.71%	1,306	43,871	0.71%	16,408
3	39,882	39,882	0.00%	90	38,370	3.79%	1,269	38,275	4.03%	14,285
4	135,826	133,328	1.84%	288	133,328	1.84%	3,365	133,328	1.84%	35,548
5	55,338	53,404	3.49%	146	51,808	6.38%	1,814	51,794	6.40%	18,801
6	43,677	43,420	0.59%	164	43,029	1.48%	1,552	43,029	1.48%	17,360
7	66,969	51,117	23.67%	149	50,055	25.26%	1,401	48,258	27.94%	15,438
8	63,486	56,269	11.37%	214	56,269	11.37%	1,936	56,269	11.37%	18,143
9	79,694	57,952	27.28%	272	57,952	27.28%	2,577	57,952	27.28%	26,933
10	41,285	40,298	2.39%	109	40,194	2.64%	1,354	40,102	2.87%	16,627
11	42,511	42,308	0.48%	149	42,308	0.48%	1,840	42,073	1.03%	20,394
12	36,807	34,790	5.48%	110	34,256	6.93%	1,112	34,256	6.93%	14,298
13	38,579	38,521	0.15%	109	38,348	0.60%	1,094	38,156	1.10%	14,577
14	46,688	42,170	9.68%	153	41,858	10.35%	1,588	41,624	10.85%	19,167
15	77,309	72,769	5.87%	275	72,769	5.87%	3,241	72,769	5.87%	30,170
16	40,367	39,840	1.31%	108	39,840	1.31%	1,208	39,840	1.31%	15,136
17	35,606	34,787	2.30%	95	34,475	3.18%	997	34,475	3.18%	11,904
18	39,595	39,283	0.79%	131	39,283	0.79%	1,354	39,197	1.01%	16,390
19	67,287	66,644	0.96%	128	58,854	12.53%	1,562	58,818	12.59%	16,879
20	63,486	56,269	11.37%	214	56,269	11.37%	2,010	56,269	11.37%	24,564
Total	1,106,413	1,034,803	6.47%	159.7	1,020,481	7.77%	1732.9	1,017,700	8.02%	19,337.20

Although not reported in the tables, additional data was collected. These included the success rate of players, percent of successful kicks for both offense and defense, the number of goals for both offense and defense, the amount of successful celebrations, and various other objectives. This data is presented in Tables 4.4 – 4.7.

The success rate of a player is the percentage of times he successfully kicked a ball. This data was collected for each offensive position and presented for each number of kicks. Since the offensive goalie's solution is accepted 100% of the time, this player was omitted from the table. The data can be seen in Table 4.4. The SOCCER heuristic performs over a half million kicks. For these kicks, strikers performed successful kicks

5.90% of the time, midfielders had a success rate of 4.19%, and defenders had a success rate of 9.16%.

Table 4.4 Success Rate of Offensive Players

	Success Rate		
	Striker	Midfielder	Defender
100 Kicks	8.43%	6.88%	12.19%
1,000 Kicks	7.03%	5.16%	10.10%
10,000 Kicks	5.77%	4.08%	9.05%
Overall	5.90%	4.19%	9.16%

Table 4.5 provides the overall success rate of the offensive and the defensive teams, as well as the average number of goals scored by each team. The recorded statistics show that the success rate for each team is fairly constant. The offensive team is successful less than 10% of the time and the defensive team is successful approximately 28% of the time. This indicates that the offensive team found more solutions, which were worse than the current solution. Thus, the SOCCER heuristic spends the majority of its effort searching in the proximity of quality solutions.

As expected, the number of goals for each team increases as the number of iterations increases. This is because there are more opportunities to score. Typically, the offensive team scores; however, the defense occasionally scores an own goal.

Table 4.5 Success Rate and Goals

	% Successful		Goals	
	Offense	Defense	Offense	Defense
100 Kicks	9.43%	31.14%	2.05	0.53
1,000 Kicks	7.77%	28.69%	3.5	0.57
10,000 Kicks	6.61%	28.15%	4.71	0.62
Average	6.73	28.22	3.42	0.57

The celebration statistics are presented in Table 4.6. The offensive team celebrates when a goal is scored. For this application, all three offensive strikers and one midfielder were selected to celebrate. During celebrations, on average, striker neighborhoods were searched 103 times per data set. Of those, 2.3% were successful. Midfielders performed an average of 34 neighborhood searches per data set with 1.11% successful. Thus, the celebration dance is important for improving solutions.

Table 4.6 Celebration Statistics

	Striker		**Midfielder**	
	Iterations	Success Rate	Iterations	Success Rate
100 Kicks	85	2.30%	28	0.83%
1,000 Kicks	100	1.96%	33	0.70%
10,000 Kicks	124	2.54%	41	1.64%
Average	103	2.29%	34	1.11%

Table 4.7 provides the average improvement and average time for each combination of the number of customers and the number of kicks. The success rate for each number of kicks increases as the number of customers increase. The success rate of each customer data set increases as the number of kicks increase and the percent improvement for each data set also increases as the number of kicks increase. However, the running time of the heuristic also increases with the number of kicks. The smallest combination – 50 customers for 100 kicks – can run one data set in approximately 2 seconds. The largest combination – 500 customers for 10,000 kicks – takes an average of 5.37 hours per data set.

Table 4.7 Summary Statistics

	100 Kicks			1,000 Kicks			10,000 Kicks		
	Improvement	Success	Time	Improvement	Success	Time	Improvement	Success	Time
50 Customers	6.24%	60%	2	8.42%	85%	20.3	9.17%	95%	220.65
100 Customers	4.77%	80%	7.35	6.21%	90%	77.05	7.27%	90%	824.85
500 Customers	6.47%	90%	159.7	7.77%	100%	1,732.90	8.02%	100%	19,337.20
Total	6.12%	77%	56.32	7.54%	92%	610.08	7.99%	95%	6,018.07

As one can see, there is a trade-off between percent improvement and time per data set. Therefore, when determining the settings for the heuristic one should consider these two points.

For small data sets, a large number of kicks should be used. This will yield the best results and the required time is small. In the instances tested, a quality solution for 50 customers was found in less than four minutes when the SOCCER heuristic was run for 10,000 iterations.

For medium data sets, the SOCCER heuristic should be run for a large number of iterations. For the instances tested, it took an average of 14 minutes to find a quality solution with 10,000 kicks. This is not a significant amount of time when one considers the added value of the improved solution.

For large data sets, the best results are found when the heuristic is run for a large number of kick. However, this is not always feasible since it may take hours to create a quality solution. Therefore, one should find the desired balance between required time and the quality of the solution.

For the limestone company, one month of production equates to roughly 100 customers. Therefore, this company should run the SOCCER heuristic for 100 customers and 10,000 kicks. With these settings, the limestone company will have a quality schedule in less than 14 minutes. With these settings, the SOCCER heuristic improved their production schedules by an about 2 days per month. This is an additional month’s worth of production per year.

Chapter 5: Conclusion

Variable neighborhood search heuristic is a heuristic commonly used in optimization problems. With this method the heuristic cycles between searches by changing the neighborhood. It allows for a wide variety of search techniques to be used in a single problem. It also allows one to escape from a local optimal solution, which is important when searching for quality solutions.

The downfall with variable neighborhood search heuristic is the method for switching between neighborhoods. Typical variable neighborhood search heuristic instances switch between neighborhoods cyclically or randomly. Variable neighborhood search heuristics need a structure or method to switch between neighborhoods.

This thesis presents a new heuristic, fútbol Strategies applied to Optimize Combinatorial problems to Create Efficient Results – the SOCCER heuristic, which was inspired by fútbol. For this heuristics, the current solution corresponds to the fútbol ball. Each player on the field has a position, speed, and location. Every player has neighborhood search associated with him. The player who can reach the ball in the least amount of time performs his neighborhood search. The time it takes a player to reach the ball is calculated using each player’s speed and proximity to the ball.

The competing teams in fútbol are also important to this heuristic. The teams seek to move the ball in opposing directions. The offense attempts to move the ball forward towards the goal (better solutions), while the defense seeks to prevent it. For the heuristic, this means that the offensive team’s solution is accepted if it is better than the previous solution. Solutions found by the defensive team are accepted if worse than the previous solution. Thus, the defensive team keeps the heuristic from being stuck at a locally optimal solution.

This thesis applies the SOCCER heuristic to a job shop scheduling instance. The SOCCER heuristic is capable of scheduling production for 24 days in approximately 14 minutes. In the instances tested, the heuristic improved the production schedules by an about 2 days. This is an additional month of production per year. By implementing the SOCCER heuristic, the limestone company will incur a large increase in profits.

5.1 Future Research

The SOCCER heuristic provides the preliminary research for numerous other research topics. This section provides other researchers with brief ideas on only a few of these research topics. Since the SOCCER heuristic could be applied to numerous other optimization problems, this section merely focuses on improvements related to JSS and the reader should extend these comments to any optimization problem.

Foremost, minimal effort was spent customizing and optimizing the SOCCER heuristic and this heuristic has numerous settings that could improve its performance. One could perform additional research into the initial settings, such as the speed of the offense versus the defense, the starting locations and the *threshold*. Additionally, the length of celebrations should also be optimized.

Future customization could also be performed on the neighborhoods assigned to each position. In the current heuristic, all players in same position have the same neighborhood. It would be interesting to vary the neighborhoods within each position and see if certain neighborhoods yield better results. This would allow a deeper analysis of neighborhoods and allow one to analyze the importance of selecting neighborhoods.

One could attempt to better mimic a fútbol game. Limits could be set on players so that a center back defender could never play a striker's ball. Additionally an out of bounds could be incorporated. Briefly, the ball is out of bounds if a solution is infeasible. Therefore, when the ball is kicked out of bounds, a new solution could be found using a different search technique, such as ejection structures. Once the solution returns to feasibility, the ball has been thrown in bounds and the search continues. This would allow further manipulation of possible solutions and may lead to better results.

An exciting extension of the SOCCER heuristic is to expand it into multicriteria optimization. Imagine a JSS instance that wanted to minimize the makespan and also minimize the tardiest order. This would allow the field to be multi-dimensional and one would correspond to the *x* coordinate to the makespan and the maximum tardiness to the *y* coordinate. One would accept an offensive kick if either objective value is better than the existing. In all likelihood, one would maintain any Pareto optimal solution. This concept could be expanded beyond a two dimensional field and into an arbitrary number of criteria.

In summary, my view of fútbol will forever be distorted. When a player mishandles a ball, I will yell, "Find a neighborhood that is improving." If a celebration is weak, I will know that they could have found a better solution with more dancing. If a player is beaten, he should run at his fastest speed to save the goal. This knowledge will enable me to gain more enjoyment out of watching a fútbol game.

References

[1]. Adibi, M., Zandieh, M., Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications, v 37, n 1*, pp. 282-287.

[2]. Alzaqebah, M. and Abdullah, S. (2014). An adaptive artificial bell colony and late-acceptance hill-climbing algorithm for examination timetabling. *Journal of Scheduling, v 17,* pp. 249-269.

[3]. Asano, M. and Ohta, H. (2002). A heuristic for job shop scheduling to minimize total weighted tardiness. *Computers and Industrial Engineering, v 42, n 2-4,* pp 137-147.

[4]. Ceschia, S., Gaspero, L., and Shaerf, A. (2012). Design engineering and experimental analysis of a simulated annealing approach to the post-enrollment course timetabling problem. *Computers and Operations Research, v 39, n 7*, pp. 1615–1624

[5]. Davis, Lawrence. (1985). Job shop scheduling with genetic algorithm research. *Proceedings of an International Conference on Genetic Algorithms*, pp. 136-140.

[6]. Dell'Amico, M. and Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research, v 41, n 1-4,* pp. 231-252.

[7]. Dowsland, K. A. and Thompson, J.M. (2000). Solving a nurse scheduling problem with knapsacks, networks and tabu search. *Journal of the Operational Research Society, v 51, n 7,* pp. 825-833.

[8]. El-Bouri, A., Azizi, N., and Zolfaghari, S. (2006). A comparative study of a new heuristic based on adaptive memory programming and simulated annealing: The case of job shop scheduling. *European Journal of Operational Research, v 177, n 3,* pp. 1894 – 1910.

[9]. Feng, X., Leung, H., and Tang, L. (2005). An effective algorithm based on GENET neural network for job shop scheduling with release dates and due dates. *Advances in Neural Networks – ISNN 2005. Second International Symposium on Neural Networks. Proceedings, Part 1, v 1,* pp. 776-781.

[10]. Gonzalez, T. and Sahni, S. (1978). Flowshop and jobshop schedules: complexity and approximation. *Operations Research, v 26, n 1,* pp. 36-52.

[11]. Got Soccer, LLC. (2015). The end of 4-4-2? Retrieved from http://home.gotsoccer.com/magazine.aspx?PageID=146

[12]. Hamiez, J. P. and Hao, J.K. (2001). Solving the sports league scheduling problem with tabu search. *Lecture Notes in Computer Science*, *v 2148,* pp. 24-26.

[13]. Hansen, P., Mladenovic, N., and Urosevic, D. (2006). Variable neighborhood search and local branching. *Computers & Operations Research, v 33, n 10,* pp 3034 – 3045.

[14]. Humphrys, M. (1987). A population of hill-climbers. *Dublin City University : School of Computing.* Retrieved from http://computing.dcu.ie/~Humphrys/Notes/GA/evolution.html

[15]. Jiang, Y., Dong, J., and Ji, M. (2013). Preemptive scheduling on two parallel machines with a single server. *Computers & Industrial Engineering, v 66, n 2,* pp. 514-518.

[16]. Klusáček, D. (2013). Applied solution techniques. Retrieved from: http://www.fi.muni.cz/~xklusac/index.php?page=en-grid.

[17]. Koubi, S. and Shalaby, N. (2008). The combined use of a genetic algorithm and the hill-climbing algorithm to find difference triangle sets. *Journal of Combinatorial Mathematics and Combinatorial Computing, v 66,* pp. 289-296.

[18]. Langston, M. (1987). Interstage transportation planning in the deterministic flow-shop environment. *Operations Research, v 35, n 4*, pp. 556-564.

[19]. Lei, D. (2011). Scheduling stochastic job shop subject to random breakdown to minimize. *International Journal of Advanced Manufacturing Technology, v 55, n 9-12,* pp. 1183-1192.

[20]. Liang, Y and Chuang, C. (2013). Variable neighborhood search for multi-objective resource allocation problems. *Robotics and Computer-Integrated Manufacturing, v 29, n 3,* pp. 73-78.

[21]. Liao, C. J., and Cheng, C. C. (2007). A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. *Computers and Industrial Engineering*, *v 52, n 4*, pp. 404–413.

[22]. Lim, A., Rodrigues, B, and Zhang, X. (2005). A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research, v 174, n 3,* pp. 1459-1478.

[23]. Lim, A. and Xingwen, Z. (2005). A two-stage for the vehicle routing problem with time windows and a limited number of vehicles. *Proceedings of the Annual Hawaii International Conference on System Sciences,* pp. 82.

[24]. Ling-Huey, S and Pei-Chann, C. (1998). A heuristic for scheduling general job shops to minimize maximum lateness. *Mathematical and Computer Modelling, v 27, n 1,* pp. 1-15.

[25]. Lipka, Michael. (2014). 5 facts about the World Cup – and the people who are watching. *Pew Reseach Center.* Retrieved from http://www.pewresearch.org/fact-tank/2014/06/16/5-facts-about-the-world-cup-and-the-people-who-are-watching/

[26]. Mattfeld, D.C. and Bierwirth, C. (2004). An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research, v 155, n 3,* pp. 616-630.

[27]. Mladenovi´c, N., Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research, v 24, n 11,* pp. 1097–1100.

[28]. Muggy, L. and Easton, T. (2012). Generating class schedules within a complex modular environment with application to secondary schools. *Journal of Scheduling.*

[29]. Nagata, Y. and Tojo, S. (2009). Guided ejection search for job shop scheduling problem. *Lecture Notes in Computer Science, v 5482,* pp. 168-179.

[30]. Roshanaei, V., Naderi, B., Jolai, F., Khalili, M. (2008). A variable neighborhood search for job shop scheduling with set-uptimes to minimize makespan. *Future Generation Computer Systems, 2009, v 25, n 6,* pp. 654-661.

[31]. Samorani, M., Laguna, M. (2012). Data-mining-driven neighborhood search. *Journal on Computing, v24, n2,* pp. 210-227.

[32]. Satoh, T. and Nara, K. (1991). Maintenance scheduling by using simulated annealing method [for power plants]. *IEEE Transactions on Power System, v 6, n 2,* pp. 850-857.

[33]. Schöpflin, R., Teif, V., Müller, O., Weinberg, C., Rippe, K., and Wedemann, G. (2013). Modeling nucleosome position distributions from experimental nucleosome positioning maps. *Bioinformatics, v 29, n 19*, pp. 2380-2386.

[34]. Steinhofel, K., Albrecht, A., Wong, C.K., 2003. An experimental analysis of local minima to improve neighborhood search. *Computers and Operations Research, v 30,n 14,* pp. 2157–2173.

[35]. Turguner, C. and Sahingoz, O.K. (2014). Solving job shop scheduling problem with ant colony optimization. *IEEE 15th International Symposium on Computational Intelligence and Informatics,* pp. 385-389.

[36]. Van Laarhoven PJM, Aarts EHL, Lenstra JK.(1992). Job shop scheduling by simulated annealing. *Operations Research, v 40, n 1,* pp.113–25.

[37]. Yang, C., Chuang, S., and Hsu, T. (2011). A genetic algorithm for dynamic facility planning in job shop manufacturing. *International Journal of Advanced Manufacturing Technology, v52, n 1-4,* pp. 303-309.

[38]. Xhafa, F., Herrero, X., Barolli, A., and Takizawa, M. (2013). A hill climbing algorithm for ground station scheduling. *Lecture Notes in Electrical Engineering, v 253,* pp. 131-139.

[39]. Zhang, R. and Wu, C. (2007). A hybrid immune simulated annealing algorithm for the job shop scheduling problem. *Applied Soft Computing Journal, v 10, n 1,* pp. 79-89.

[40]. Zhao, X., Zhang, A., Sun, W., and Liang, J. (2009). An improved genetic algorithm for multiple-machine scheduling problem. *2009 International Conference on Management and Service Science (MASS),* pp. 4.