Abstract

AKSAKALLI, VURAL. Heuristic Methods for Gang-Rip Saw Arbor Design and Scheduling. (Under the direction of Dr. Yahya Fathi)

This research considers the problem of designing and scheduling arbors for gang-rip saw systems. Such systems are typically used within the furniture manufacturing industry for processing lumber, where lumber boards are first ripped lengthwise into strips of different widths, and then, cut to the required lengths to be used in manufacturing.

A saw with multiple cutting channels is used to perform this operation. This saw has fixed blades at specific positions on a rotating shaft which rips incoming lumber boards into required finished widths. The pattern of cutting channels (i.e., the setting of the blades) along the saw shaft is referred to as an "arbor".

A typical instance of the problem consists of (1) a set of required finished widths and their corresponding demands, (2) a frequency distribution of lumber boards in the uncut stock, (3) a shaft length, and (4) a blade width. The objective is to design a set of (one or more) arbors and the corresponding quantity of lumber to run through each

arbor, such that the total amount of waste generated is minimized while the demand is satisfied.

In the research, we focus on solving the problem using only one arbor. First, we discuss the computational complexity of the problem and propose a total enumeration procedure which can be used to solve relatively small instances. Then, we develop algorithms based on heuristic approaches such as local improvement procedures, simulated annealing, and genetic algorithms. Our computational experiments indicate that a local improvement procedure with two nested loops, performing local search with a different neighborhood structure within each loop, gives very high quality solutions to the problem within very short execution times.

HEURISTIC METHODS FOR GANG-RIP SAW ARBOR DESIGN AND SCHEDULING

by

VURAL AKSAKALLI

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

OPERATIONS RESEARCH

and

INDUSTRIAL ENGINEERING

Raleigh

1999

APPROVED BY

Dr. Richard H. Bernhard

Dr. Matthias F. M. Stallmann

Dr. Yahya Fathi
Chair of Advisory Committee

Biography

Vural Aksakalli was born in July 1975 in Turkey. He attended Middle East Technical University in Ankara, Turkey, from September 1992 to June 1996 and graduated with a Bachelor of Science degree in Mathematics as an honor student.

Upon graduation, he worked as a research assistant in the Department of Economics at Middle East Technical University until he accepted a full-time position as a researcher at the National Research Institute of Electronics in Istanbul, Turkey, in January 1997.

In January 1998, he entered North Carolina State University in Raleigh, NC, to pursue a master's degree; co-majoring in Operations Research and Industrial Engineering. He is currently working as a research assistant in the Department of Industrial Engineering.

He is a member of the Honor Society of Phi Kappa Phi, and the Institute for Operations Research and the Management Sciences.

Acknowledgments

I would like to express my deepest appreciation to my advisor Dr. Yahya Fathi for his invaluable guidance and sincere help throughout my graduate study at North Carolina State University. I would also like to thank him for his continuous support and advice in the completion of this thesis.

I would like to express my gratitude to Dr. Richard H. Bernhard for the valuable experience that I had as a teaching assistant under his direction, and to Dr. Matthias F. M. Stallmann for his constructive comments. I also would like to thank them for serving on my thesis committee.

Finally, I would like to thank my parents, Nuray and Hakki, brother Nihat, and sister Aylin for their endless support and encouragement throughout my graduate program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

A typical lumber processing method within the furniture manufacturing industry is the cutting of lumber along the length (parallel to the grain) into strips of different widths. These strips are then cut to the required lengths and used in manufacturing.

A commercial saw with multiple cutting channels has been developed to carry out this task. This saw has fixed blades at specific positions on a rotating shaft which rip incoming lumber boards lengthwise into desired finished widths. This method is commonly referred to as the *"gang-rip method"* of lumber cutting, and the setting of the blades (i.e., the pattern of cutting channels) along the saw shaft is called an *"arbor"*[1].

Several arbors may be employed during a given cutting job, but each arbor exchange requires shutting down the system and resetting the blades manually, resulting in a loss in operation time of at least five to ten minutes for each exchange. Thus, it is generally preferable to use the least number of arbors (preferably one) for a particular cutting job to avoid such time loss.

[1] *Typically, for a blade setting to be referred to as an arbor, real-valued weights have to be specified for each of the finished widths in the demand schedule. However, we will assume that these weights are the same as the finished widths throughout the thesis.*

Fathi et al. [3] developed a mathematical model for scheduling such gang-rip saw systems and used the concept of column generation of linear programming to design appropriate saw arbors to process a given stream of incoming lumber according to a demand schedule for finished widths. They obtain good results with reasonable computational requirements with this model.

However, the total number of arbors in the optimal solution they find in this manner is typically large, usually as many arbors as the number of finished widths required (typically between five and ten). Such a large number of arbors, on the other hand, results in important loss in production time as mentioned.

Our objective in this research is to reduce this significant time loss due to arbor exchange by designing various algorithms to perform a given cutting job with *only one arbor*. The next two sections describe the problem environment and present the complete problem definition.

1.2 Problem Environment

Fathi et al. [3] describe the problem environment as follows:

"The width of each incoming board is sized by the operator as it is put on to the conveyor feeding into the saw. This is accomplished by fixing two high-intensity light beams on the opposing edges of the board. The region between the two beams is the portion that the operator judges to be the usable stock (light beams are fixed as close to the edges as possible without cutting through the rough parts).

The saw is 'smart' in the sense that once the usable width of an uncut board is registered, it will move the board (parallel to the saw shaft) to the block of contiguous cutting channels which will produce the most valuable yield of finished cuts (see Figure 1.1).

The input (data) to a given cutting problem consists of (1) a demand schedule for the finished widths and (2) an inventory of available uncut stock. There are two units

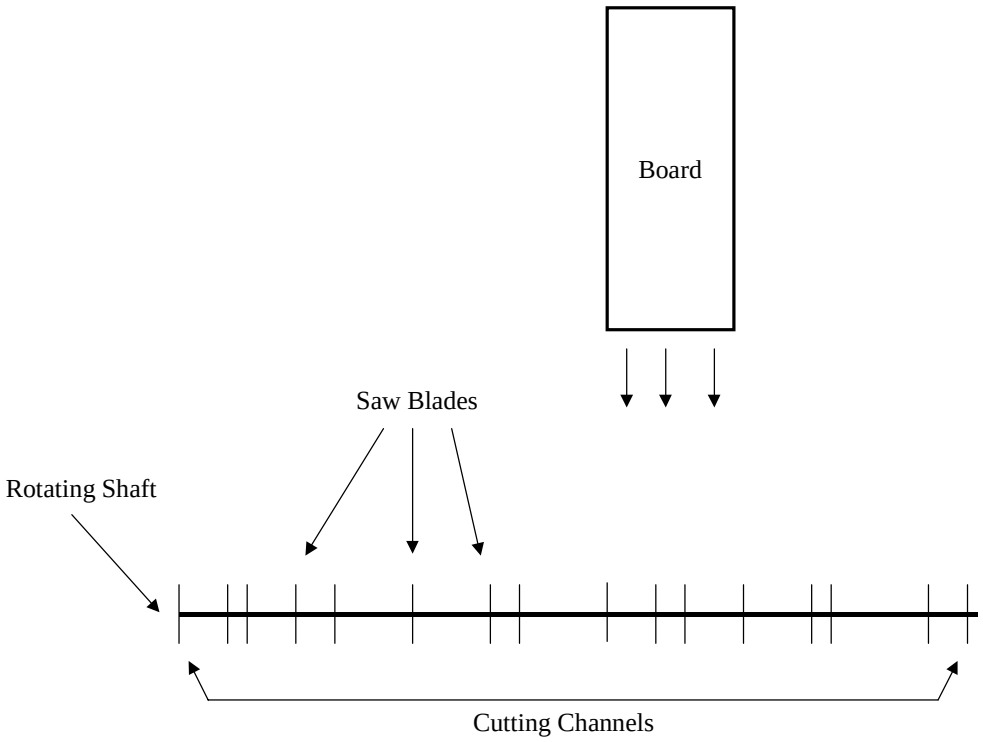

Figure 1.1: Gang-rip cutting method.

Finished width (inches)	Required board feet
1	2000
$1\frac{3}{8}$	5000
$1\frac{7}{8}$	4000
2	1000
3	1500

Table 1.1: A hypothetical demand schedule.

of measure which are commonly used to express these quantities. A *linear foot* is a unit of measure which is concerned with length only; a board which is ten feet long measures ten linear feet regardless of its width. A *board foot*, on the other hand, is a measure that refers to one square foot of lumber (of one inch thickness), and is thus sensitive to both length and width. (In fact 'board foot' is a measure of volume; however, since all cutting stock is assumed to be of the same thickness, we use the term 'board foot' to refer to one square foot of lumber.)

The demand schedule for a cutting job is typically specified in board feet. A hypothetical demand schedule, which consists of five distinct finished widths, is presented in Table 1.1. The number of distinct finished widths is typically between five and ten.

To construct a mathematical model, we need to convert the demand schedule quantities from units of board feet to units of linear feet. Conversion between these two units of measure is readily accomplished using the following formula:

$$units\ of\ linear\ feet = (units\ of\ board\ feet) \times (12/finished\ width)$$

The inventory of available cutting stock is measured in units of linear feet. Uncut lumber is delivered in lots containing many different rough widths (typically from $3''$

Board width (inches)	Linear feet in sample
3	1250
$3\frac{1}{8}$	1075
$3\frac{1}{4}$	2040
$3\frac{3}{8}$	4000
-	-
-	-
-	-
$14\frac{7}{8}$	5410
15	4900

Table 1.2: A hypothetical uncut stock sample.

to 15″). A random sample of boards is drawn and the usable width of each board in the sample is determined to within some convenient fraction of an inch (usually 1/8″ or 1/16″). The quantity of each uncut width in the sample (in linear feet) is tabulated in the form of a frequency distribution. This distribution is then used as representative of the mix of widths among all incoming boards. (Indeed, sample measurement is usually done automatically during the normal operation of the machine. As such, the sample size is typically large, and the resulting frequency distribution is a good estimate of the true distribution of the incoming board widths). A hypothetical distribution is presented in Table 1.2.

During the cutting operation, uncut boards are pulled randomly from an incoming lot and fed into the saw; no presorting by width is performed. There, the probability distribution of the width of an incoming board may be estimated based on the empirical frequency distribution derived from the sample tabulation described above.

The amount of waste produced over the course of a cutting job is the final key component to the problem. Waste is produced as a result of (1) sawdust loss, and (2) strips cut at the edges of an incoming board which do not satisfy any finished width in the demand schedule (edge strips). The width of the sawdust loss associated with each saw cut is equivalent to the width of the saw blade (typically $1/8''$ or $3/16''$). We assume that the number of saw cuts along the length of an incoming board is one more than the number of finished widths obtained. This is consistent with common industry practice which requires that at least a full saw blade width be cut from each edge of an incoming board. The width of the edge strips (if any) would be added to the sawdust loss to obtain the width of the 'waste' generated.

It is [also] desirable (but not required) that the quantity of each finished width produced does not significantly exceed its corresponding demand (5%-10% overproduction in any finished width is considered acceptable)."

1.3 Problem Definition

Considering the situation described above, Fathi et al. [3] define the "Gang-Rip Saw Arbor Design and Scheduling Problem" (GRSADSP) as follows:

> ($GRSADSP$) "For a given (1) set of desired finished widths and their corresponding quantities (demand schedule), (2) frequency distribution of incoming boards (raw material), (3) shaft length, and (4) blade width, find a set of (one or more) arbors (i.e., blade settings) and the corresponding quantity of incoming lumber (in linear feet) to run through each arbor, such that the demand is met while the total board feet of waste is minimized."

In other words, the objective of the above problem is to minimize the total waste produced in the cutting job subject to the restriction that the production of each finished width is at least as much as the corresponding demand. Note that in this definition,

the number of arbors used to achieve this objective is not specified, and the problem solver is free to choose as many arbors as he/she finds appropriate.

In this research, we focus on solving the $GRSADSP$ using a specified number of arbors. Therefore, we define the "Gang-Rip Saw Arbor Design and Scheduling Problem with N Arbors" ($GRSADSP/N$) as follows:

> ($GRSADSP/N$) "Given a positive integer N, solve $GRSADSP$ using exactly N arbors."

In particular, we focus on solving $GRSADSP/1$, which refers to solving $GRSADSP$ using only one arbor.

1.4 Notation and Terminology

The following notation characterizes the incoming lumber boards and the demand schedule:

M	Number of distinct finished widths in the demand schedule.
w_i	The ith distinct finished width (inches), $for\ i = 1\ to\ M$.
r_i	Required board feet of finished width w_i, $for\ i = 1\ to\ M$.
K	Number of distinct incoming board widths in the uncut stock.
b_k	The kth distinct incoming board width (inches), $for\ k = 1\ to\ K$.
l_k	Linear feet quantity of the board b_k in the random sample, $for\ k = 1\ to\ K$.
l	Total length (linear feet) of all incoming boards in the random sample. That is,

$$l = \sum_{k=1}^{K} l_k.$$

p_k Probability that the width of the incoming board fed into the saw (at a randomly selected instant) is b_k, $for\ k = 1\ to\ K$ (Estimated by l_k/l).

L Shaft length (inches).

S Blade width (inches).

The following notation describes the arbors:

Ω The set of all possible arbors.

A_{ijk} Number of strips of width w_i cut from an incoming board of width b_k by arbor j, $j \in \Omega$, $for\ i = 1\ to\ M$, and $for\ k = 1\ to\ K$.

a_{ij} Expected linear feet of finished width w_i cut by arbor j from one linear foot of incoming lumber. Therefore,

$$a_{ij} = \sum_{k=1}^{K} w_i A_{ijk}.$$

T_{jk} Width (inches) of the waste (sawdust plus the edge strips) produced when arbor j cuts incoming board width b_k. So,

$$T_{jk} = b_k - \sum_{i=1}^{M} w_i A_{ijk}.$$

d_j Expected board feet of waste produced by arbor j from one linear foot of incoming lumber. It follows that

$$d_j = \sum_{k=1}^{M} p_k T_{jk}/12.$$

WP_j Expected waste percentage corresponding to arbor j, defined as

$$WP_j = \frac{\sum_{k=1}^{M} p_k T_{jk}}{\sum_{k=1}^{K} p_k b_k} * 100.$$

Notice that in the above definition,

1. $\sum_{k=1}^{M} p_k T_{jk}$ is the expected width of the waste (inches) produced by arbor j from one linear foot of incoming lumber, and

2. $\sum_{k=1}^{K} p_k b_k$ is the expected width of incoming boards (inches).

YP_j Expected yield percentage corresponding to arbor j, defined as:

$$YP_j = 100 - WP_j.$$

OP_j Expected overage percentage corresponding to arbor j, defined as:

$$OP_j = \frac{\sum_{i=1}^{M} \left(x_j \frac{w_j a_{ij}}{12} - r_i \right)}{\sum_{i=1}^{M} r_i} * 100.$$

Note that in this definition,

1. $\sum_{i=1}^{M} \left(x_j \frac{w_j a_{ij}}{12} - r_i \right)$ is the expected total board feet of finished widths produced by arbor j above their corresponding demands, and

2. $\sum_{i=1}^{M} r_i$ is the total board feet of finished widths in the demand schedule.

x_j Linear feet of incoming lumber cut by arbor j, $j \in \Omega$.

n_j Number of cutting channels on arbor j, $j \in \Omega$.

$\mathbf{R}_j$ Vector whose successive entries represent the widths of successive cutting channels on arbor j, $j \in \Omega$.

R_{ij} The width of the ith cutting channel (inches) from left on arbor j, $for\ i = 1\ to\ n_j,\ j \in \Omega$.

L_j Length of arbor j (inches), $j \in \Omega$. So, $L_j \leq L$, and

$$L_j = S + \sum_{i=1}^{n_j}(S + R_{ij}).$$

Notice that arbors are denoted by vectors whose entries correspond to the widths of the cutting channels. For instance,

$$\mathbf{R}_1 = (w_1 w_3 w_2 w_1 w_3)$$

represents an arbor with five cutting channels whose widths are successively w_1, w_3, w_2, w_1 and w_3; arranged from left to right. (i.e., $R_{11} = w_1,\ R_{21} = w_3$, and so on).

The following two terms characterize an arbor: the term "*feasible*" describes an arbor on which there is at least one cutting channel of width w_i for all $i = 1$ *to* M; and the term "*full-length*" describes an arbor for which the following inequality holds:

$$L - L_j < w_1 + S$$

where w_1 represents the smallest finished width in the demand schedule. In other words, for a full-length arbor, the remaining length on the shaft is less than the smallest finished width plus the blade width.

1.5 Research Objectives

We have four major objectives in this research:

1. Investigate the computational complexity of $GRSADSP/1$,

2. Develop a total enumeration procedure which can be used to determine the optimal solution to small instances of $GRSADSP/1$,

3. Apply various heuristic methods such as local improvement procedures, simulated annealing and genetic algorithms to $GRSADSP/1$,

4. Perform computational experiments to compare the efficiency of these heuristics in terms of solution quality and computational requirements[2].

1.6 Organization of the Thesis

The rest of the thesis is organized as follows: Chapter 2 discusses the computational complexity of $GRSADSP/1$ and describes a total enumeration procedure for the problem. The next three chapters present the heuristics implemented for $GRSADSP/1$: Chapter 3 presents local improvement procedures, Chapter 4 describes simulated annealing, and Chapter 5 discusses genetic algorithms. These three chapters can be read independent of each other. Chapter 6 presents the computational experiments with the heuristic procedures. Finally, Chapter 7 presents the summary of the research, conclusions, and recommendations for future research.

[2] *The algorithms developed in this research were implemented in C++ using object-oriented methodology and the computational experiments were performed on a 380 MHz personal computer.*

Chapter 2

On the Complexity of GRSADSP/1

In this chapter, the computational complexity of $GRSADSP/1$ is discussed and it is shown that a closely related problem is NP-complete. Then a total enumeration procedure for $GRSADSP/1$ is described and the complexity of the enumeration procedure is discussed.

2.1 Theoretical Upper Bound on the Yield Percentage

2.1.1 Problem Definition

Fathi et al. [3] compare the waste percentage in a cutting job with a theoretical lower bound:

"A theoretical lower bound for each instance of the problem is obtained if we assume that each incoming board width b_k is cut according to a pattern that minimizes the total waste for that particular b_k, regardless of the demand schedule and whether or not the cutting pattern is indeed available on the arbor."

This lower bound is independent of the shaft length and also the number of arbors, since it is already assumed that each incoming board is cut such that the total waste for

that board is minimized regardless of whether or not that specific cutting pattern is actually available on the shaft. Therefore, this lower bound also applies to $GRSADSP/1$. In other words, it specifies a lower bound on WP_j for any arbor j for a specific instance of $GRSADSP/1$.

Recall that

$$YP_j = 100 - WP_j.$$

So, the lower bound on the waste percentage can actually be used to obtain an upper bound on YP_j, simply by subtracting this lower bound from 100. This theoretical upper bound will be referred to as the *"yield limit"* for the instance.

We now define "Yield Percentage Upper Bound Problem" ($YPUBP$) as follows:

> ($YPUBP$) "Can the yield limit in $GRSADSP/1$ be achieved for a given shaft length and blade width regardless of the demand schedule?".

In the next section, we show that $YPUBP$ is indeed an NP-complete problem.

2.1.2 NP-Completeness

For a specific board width b_k, the cutting pattern that minimizes the waste for b_k can be found by solving the following knapsack problem:

Max

$$\sum_{i=1}^{M} w_i Y_{ik} \tag{2.1}$$

subject to

$$\sum_{i=1}^{M} Y_{ik}(w_i + S) \leq b_k - S$$

$$Y_{ik} \in \left\{0, 1, 2, ..., \lfloor \frac{b_k - S}{w_i + S} \rfloor \right\} \ for\ all\ i = 1\ to\ M$$

where $\lfloor \theta \rfloor$ represents the largest integer smaller than or equal to θ, and Y_{ik} denotes the number of strips of width w_i cut from an incoming board of width b_k. The first constraint states that the total width of all usable strips (plus sawdust loss) cut from an incoming board width b_k must not exceed b_k. The second constraint indicates that the number of strips of finished width w_i cut from an incoming board width b_k must be a nonnegative integer not larger than the maximum number of such strips that could physically be cut from b_k.

Notice that the optimal solution to the kth knapsack problem identifies the most desirable way in which an arbor can cut an incoming board width b_k. For an arbor to actually cut b_k according to this pattern, i.e., *realize* this optimal solution, it must have a *block of contiguous cutting channels* that has the same set of channel widths as in the optimal solution to the corresponding knapsack problem. An arbor which has this property is said to *embed* the kth optimal knapsack solution[1].

We now define "Consecutive Optimal Knapsack Solutions Problem" ($COKSP$) as follows:

> ($COKSP$) "Can the optimal solutions for all of the above K knapsack problems be embedded on a shaft with a given length L and a blade width S?".

It is clear that $YPUBP$ and $COKSP$ are essentially the same, since yield limit, i.e., the theoretical upper bound on the yield percentage, can be achieved *if and only if* one can find (i.e., construct) a pattern of cutting channels which simultaneously realizes the optimal solutions for all the K knapsack problems.

Next, we quote "Consecutive Sets Problem" (CSP) from Garey and Johnson [4]:

> (CSP) "$INSTANCE$: Finite alphabet Σ, collection $C = \{\Sigma_1, \Sigma_2, \ ... \ , \Sigma_n\}$ of subsets of Σ, and a positive integer Q. $QUESTION$: Is there a string $g \in \Sigma^*$with $|g| \leq Q$ such that, for each i, the elements of Σ_i occur in a consecutive block of $|\Sigma_i|$ symbols of g?"

[1] P*rovisions have to be made if the knapsack problem has multiple optimal solutions to avoid inconsistency.*

In this definition, Σ^* represents the set of all strings made up from the symbols in Σ. This problem has been shown to be NP-complete by Kou [9]. We now define "Consecutive Sets with Weights Problem" (CSP/W) as follows:

> (CSP/W) "$INSTANCE$: Finite alphabet $\Sigma = \{s_1, s_2, \ ... \ , s_m\}$, collection $C = \{\Sigma_1, \Sigma_2, \ ... \ , \Sigma_n\}$ of subsets of Σ, positive real-valued weights c_j corresponding to s_j, for $j = 1$ to m, and a positive real number Q. $QUESTION$: Is there a string $g \in \Sigma^*$with $\sum_{p=1}^{|g|} c_{g_p} \leq Q$, where g_p is the pth symbol in string g, such that for each i, the elements of Σ_i occur in a consecutive block of $|\Sigma_i|$ symbols of g?"

It can be seen that CSP is a special case of CSP/W, where the weights, c_i, are equal to 1 for $i = 1$ to m. Observing that one can easily verify a given solution to CSP/W in polynomial time, we conclude that CSP/W is also NP-complete.

Now, let the symbols in Σ in CSP/W represent the finished widths in the demand schedule in $GRSADSP/1$, and let the weight of each symbol in Σ be equal to the width of the finished width that the symbol represents. Then it can be observed that CSP/W is a special case of $COKSP$, where (1) Y_{ik} is equal to either 0 or 1 in each of the K knapsack solutions (i.e., there is at most one strip of each finished width in each of the K knapsack solutions), (2) the blade width S is equal to zero, and (3) the real number Q is equal to the shaft length L. Since a given solution can clearly be verified in polynomial time, this establishes that $COKSP$ is also NP-complete.

Since $COKSP$ and $YPUBP$ are essentially the same problem, we conclude that $YPUBP$ is NP-complete. We conjecture that $GRSADSP/1$, the problem that we primarily focus on in this research, is also an NP-complete problem even though we do not have a proof at the time of this writing.

2.2 Total Enumeration

In this section, *evaluation* of an arbor within the context of $GRSADSP/1$ is explained. Then a total enumeration procedure for $GRSADSP/1$ is described, and the procedure's

complexity is discussed.

2.2.1 Evaluating an Arbor

In $GRSADSP/1$, an arbor j is evaluated by finding the *expected total waste* corresponding to that arbor. The evaluation procedure starts with simulating the operation of the saw for each incoming board width b_k (i.e., all of the cutting channels on the arbor are scanned, and the cutting pattern which minimizes the waste for b_k is determined). Then a_{ij}'s (i.e., the expected linear feet of finished width w_i cut by arbor j from one linear foot of incoming lumber) are calculated using the frequency distribution of the incoming board widths. Next, d_j, the expected waste corresponding to one linear foot of incoming lumber, is computed.

At this point, we define y_j as the minimum linear feet of incoming lumber that must be cut by arbor j in order to satisfy the demand for all of the finished widths. The expected total waste for the arbor j is then obtained by multiplying d_j by y_j.

Notice that this evaluation procedure is computationally expensive due to the saw shaft simulation for every distinct incoming board width in the uncut stock.

2.2.2 A Total Enumeration Procedure

$GRSADSP/1$ is essentially a combinatorial problem: we are trying to find a sequence of objects (i.e., cutting channels), not necessarily distinct, subject to several constraints (i.e., arbor feasibility and shaft length restrictions) to minimize an objective function value (i.e., expected total waste corresponding to a given demand schedule).

As an example of how the sequence of the cutting channels affects the objective function value in $GRSADSP/1$, look at the following two feasible, full-length arbors

for the demand schedule in Table 1.1 with $L = 18''$ and $S = 1/8''$ [2]:

$$\begin{aligned} \mathbf{R}_1 &= (w_1 w_5 w_2 w_4 w_2 w_3 w_1 w_4 w_1 w_4) \\ \mathbf{R}_2 &= (w_1 w_4 w_1 w_2 w_1 w_2 w_2 w_3 w_3 w_5) \end{aligned}$$

Suppose further that the uncut stock used in the cutting job has a frequency distribution presented in Table 6.5 on page 56. After evaluating the arbors as described in the previous section, we see that the expected waste corresponding to the first arbor is $1,772$ board feet, whereas the expected waste corresponding to the second arbor is almost $6,863$ board feet. Apparently, the first arbor is a much better solution for this instance of the problem.

A natural approach to solving a combinatorial problem is to list all the feasible solutions of the problem, determine their objective function values, and pick the best. This approach is referred to as *total enumeration.*

Although it is possible in theory to solve any combinatorial problem *exactly* by a total enumeration procedure, this approach is not feasible in practice due to the fact that most combinatorial problems of reasonable size have very large number of possible solutions.

The total enumeration approach is likely to be inefficient for $GRSADSP/1$ also, because of the extremely large solution set Ω for most instances. However, we describe a total enumeration procedure for $GRSADSP/1$ in this section which gives insight into the solution structure and can actually be used to solve small instances of the problem.

In this procedure, all possible (not necessarily feasible) full-length arbors are generated *recursively*, and the feasible ones are evaluated. The arbor with the lowest expected total waste among them is then picked as the optimal solution to the current instance. The procedure for *three* distinct finished widths (i.e., $M = 3$) is described in Figure 2.1, where w_1 represents the smallest finished width in the demand schedule. The enu-

[2] *Throughout the thesis, it will be assumed that the blade width, S, is equal to $1/8''$, unless specified otherwise.*

meration procedure is implemented in the same manner for other possible values of M.

In the figure, "*Propose(j)*" refers to a subroutine which (1) checks whether the arbor j is feasible and evaluates the arbor if it is indeed feasible, and then, (2) compares the arbor j with the current best arbor and updates it if the arbor j is actually better.

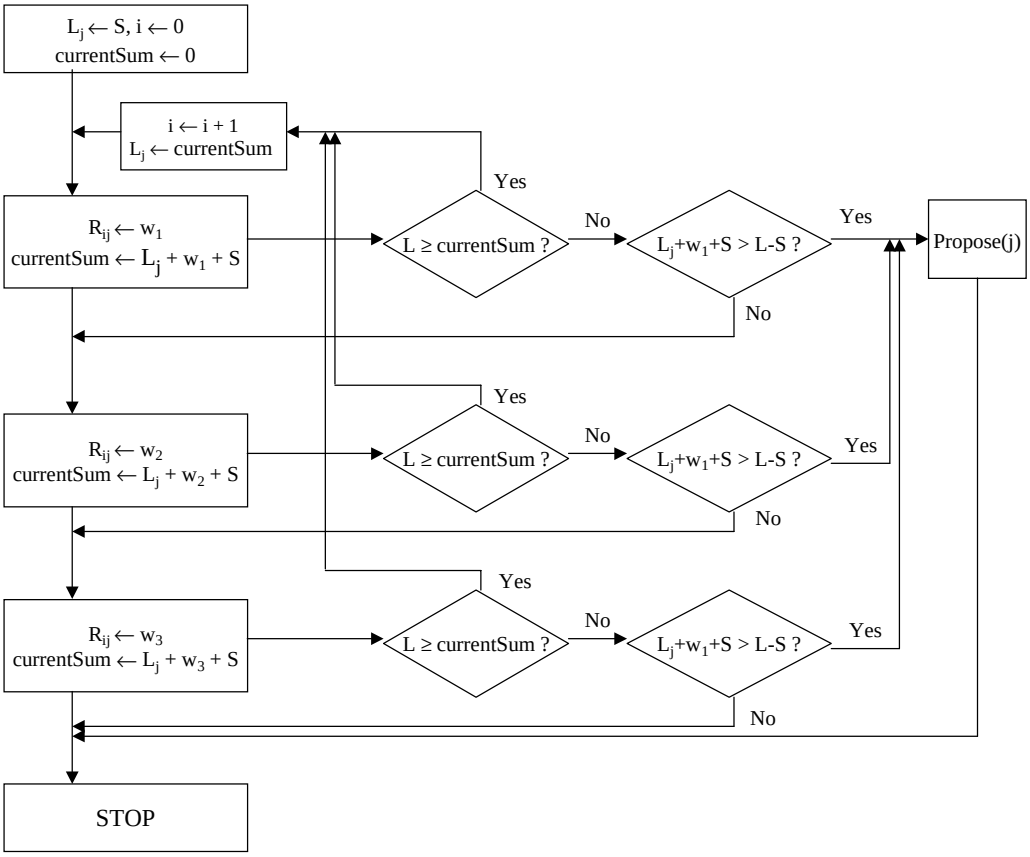

Figure 2.1: A total enumeration procedure for $GRSADSP/1$.

For example, for a demand schedule with three distinct finished widths where $w_1 = 1''$, $w_2 = 1\frac{1}{8}''$, *and* $w_3 = 1\frac{1}{4}''$; the procedure generates all the possible full-length arbors in the following fashion for $L = 5''$ (notice that only the arbors marked with an asterisk are feasible):

$$(w_1w_1w_1w_1)$$
$$(w_1w_1w_1w_2)$$
$$(w_1w_1w_1w_3)$$
$$(w_1w_1w_2w_1)$$
$$(w_1w_1w_2w_2)$$
$$(w_1w_1w_2w_3)*$$
$$(w_1w_1w_3w_1)$$
$$(w_1w_1w_3w_2)*$$
$$(w_1w_2w_1w_1)$$
$$(w_1w_2w_1w_2)$$
$$(w_1w_2w_1w_3)*$$
$$(w_1w_2w_2w_1)$$
$$(w_1w_2w_2w_2)$$
$$(w_1w_2w_3w_1)*$$
$$(w_1w_3w_1w_1)$$
$$(w_1w_3w_1w_2)*$$
$$(w_1w_3w_2w_1)*$$
$$(w_1w_3w_3)$$
$$(w_2w_1w_1w_1)$$
$$(w_2w_1w_1w_2)$$
$$(w_2w_1w_1w_3)*$$
$$(w_2w_1w_2w_1)$$
$$(w_2w_1w_2w_2)$$
$$(w_2w_1w_3w_1)*$$

$(w_2 w_2 w_1 w_1)$

$(w_2 w_2 w_1 w_2)$

$(w_2 w_2 w_2 w_1)$

$(w_2 w_2 w_3)$

$(w_2 w_3 w_1 w_1)*$

$(w_2 w_3 w_2)$

$(w_2 w_3 w_3)$

$(w_3 w_1 w_1 w_1)$

$(w_3 w_1 w_1 w_2)*$

$(w_3 w_1 w_2 w_1)*$

$(w_3 w_1 w_3)$

$(w_3 w_2 w_1 w_1)*$

$(w_3 w_2 w_2)$

$(w_3 w_2 w_3)$

$(w_3 w_3 w_1)$

$(w_3 w_3 w_2)$

$(w_3 w_3 w_3)$

2.2.3 Complexity of the Procedure

Our objective in this section is to find the total number of arbors generated by the above enumeration procedure for a given instance of $GRSADSP/1$.

We could not determine a closed-form expression for the exact number of arbors generated by the procedure. However, we present a formula which can be used to obtain an approximate value for that number.

First, we $\overline{w}$ define as

$$\overline{w} = \frac{\sum_{i=1}^{M} w_i}{M}.$$

Demand Schedule	Actual number	M^n
4A	$105,519$	$53,231$
4B	$40,565$	$45,073$
5A	$4,230,795$	$2,116,793$
5B	$198,882$	$111,317$

Table 2.1: Number of arbors generated by the enumeration procedure: actual versus approximation.

The number of slots (i.e., cutting channels) on the saw shaft, n, is then approximated by

$$n = \frac{(L - S)}{(\overline{w} + S)}.$$

There are M possible choices (i.e., cutting channels) for each slot (note that the arbors generated need not to be feasible). So, the total number of arbors generated by the enumeration procedure will approximately be equal to M^n.

For instance, consider the four demand schedules shown in Tables 6.1 and 6.2 on page 53 (notice that the two demand schedules in Table 6.1 have four distinct finished widths, and the other two in Table 6.2 have five distinct finished widths). Table 2.1 presents the comparison of the approximate value obtained by the above formula to the actual number of arbors generated by the enumeration procedure for these four demand schedules for $L = 18''$.

The form of the above formula essentially indicates that this total enumeration procedure is exponential in M and L. That is, its computational requirements grow exponentially in the number of distinct finished widths and the shaft length.

As an example of this exponential growth, consider an instance with the demand schedule in Table 1.1 (where $M = 5$) and the uncut lumber distribution presented in Table 6.5 on page 56. The enumeration procedure takes approximately 6 minutes for $L = 18''$, whereas it takes almost 90 minutes for $L = 21''$. Moreover, in some cutting jobs encountered in practice, the number of distinct finished widths is as many as ten,

and the saw shaft is as long as $36''$. Considering furthermore that the problem has to be solved several times a day within the company performing the cutting operation, it is clear that this enumeration approach is not practical in general.

Since the performance of the above enumeration procedure is not satisfactory in general, and we do not know of another exact algorithm for $GRSADSP/1$, we resort to *heuristics*. Heuristics are inexact algorithms (i.e., they do not guarantee optimal solutions) which have relatively small computational requirements, and lead to "reasonable" solutions. The reader is referred to Steiglitz [13] and Reeves [12] for an introductory discussion of heuristics. We present three different heuristic approaches for solving GRSADSP/1 in the following chapters.

Chapter 3

Local Improvement Procedures

This chapter presents several Local Improvement Procedures ($LIPs$) for solving GR-SADSP/1 and discusses the results of a computational experiment to compare these procedures on an empirical basis.

3.1 Introduction

The basic idea in a local improvement procedure is to start with an initial feasible solution to the problem of interest as the "current solution" and try to find an improved solution by a set of local perturbations. If such an improved solution is found, it becomes the new "current solution", and the process is repeated. Otherwise, the search is terminated.

The set of solutions that can be obtained by a series of local perturbations from an arbitrary solution S is called the "*neighborhood* of S", and denoted by $N(S)$. A generic LIP description within the context of a minimization problem is shown in Figure 3.1, where $f(S)$ is the value of the objective function evaluated at S.

For a few combinatorial problems, $LIPs$ are guaranteed to find the global optima with some special neighborhood structures. However, the final (i.e., locally optimal) solutions obtained by $LIPs$ usually fail to be globally optimal. The reason is that $LIPs$

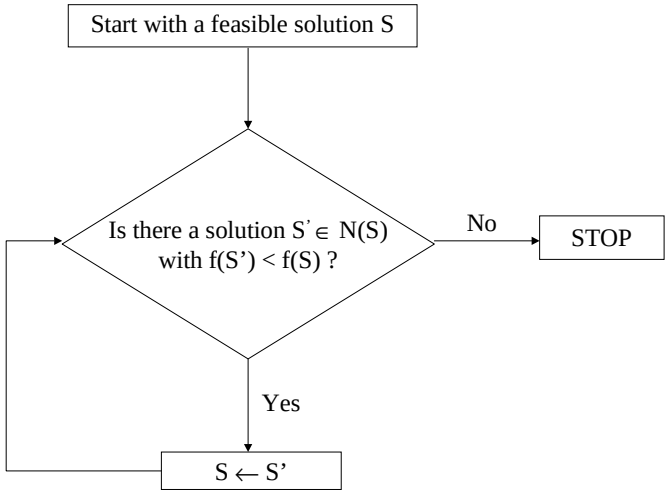

Figure 3.1: A generic local improvement procedure.

search a defined neighborhood of the current solution, and this results in search of only a small subset of the solution space. The hope is that the solutions obtained by $LIPs$ will be "good enough".

In order to implement a LIP within the context of a specific problem, its fundamental elements need to be clearly defined: (1) solution representation and evaluation, (2) neighborhood structures, and (3) search strategy. In the next section, these issues are addressed for $GRSADSP/1$.

3.2 Fundamental Elements

3.2.1 Solution Representation and Evaluation

In our LIP implementations for $GRSADSP/1$, any sequence of cutting channels along the saw shaft represents a solution (i.e., an arbor).

A solution is evaluated simply by computing the expected total waste corresponding

to the arbor as described in section 2.2.1.

3.2.2 Neighborhood Structures

The most important issue in LIP design is the selection of neighborhood structures. For $GRSADSP/1$, two different neighborhood structures are defined: *shift* and *unit neighborhoods*.

The shift neighborhood is defined as follows:

> $N_S(j) =$ (All arbors j' that can be obtained from j by selecting one cutting channel on j, removing it from its current position, and inserting it to another position).

The size of the shift neighborhood for an arbor j is *at most* $n_j(n_j - 1)$, where n_j is the number of cutting channels on j (notice that some of these neighbors may be identical to j. In that case, the number of distinct neighbors will be less than that number).

A serious drawback of this neighborhood structure is that it is not *viable*; i.e. not every solution is reachable from every other solution.

The unit neighborhood structure, on the other hand, is defined as follows:

> $N_U(j) =$ (All arbors j' that can be obtained from j by selecting one cutting channel on j, changing the width of that channel to another finished width, and "repairing" the resulting arbor if necessary).

Notice that after changing the width of a channel on arbor j to another finished width, the resulting arbor, j', can be either too long or too short for the shaft. In that case, we perform a "*repair operation*" on j' to make it full-length again. This operation consists of two steps:

1. "*trimming":* if the arbor is too long for the shaft, cutting channels are removed from the right until the arbor is not longer than the shaft anymore,

2. *"filling":* if the arbor is too short, the remaining space on the arbor is filled from the right with randomly generated channels until it is full-length again (this step is implemented *recursively* where the remaining space on the shaft is taken into account whenever a random channel is to be generated).

The size of the unit neighborhood for an arbor j is *less than or equal to* $(M-1)n_j$: after the neighbors are generated and the repair operation is applied to the resulting arbors which need repairing, some of the arbors may turn out to be identical. In that case, the size of the neighborhood for the arbor j will be less than $(M-1)n_j$ (notice also that this neighborhood structure is viable).

3.2.3 Search Strategy

Search strategy is another important issue in LIP design. Two natural strategies are *first-improvement* and *steepest descent*. In the first-improvement strategy, a favorable change is accepted as soon as it is encountered, and no further search is done with the current solution. In the steepest descent strategy, on the other hand, the entire neighborhood is searched, and the solution with the lowest cost is selected. First-improvement strategy is implemented in our $LIPs$, since the extra time is usually not justified for searching the entire neighborhood as in steepest descent.

Another issue that arises within the context of search strategy is the order in which the neighborhood is searched. The search can be done randomly, or using some lexicographic ordering. In our implementations, the search is started from the first cutting channel on the arbor and continued with the remaining channels. Whenever an improved solution is found, the search is started from the first channel of the new solution and continued in that fashion. Finally, the local searches are started with randomly generated feasible full-length arbors.

3.3 Description of the Algorithms

The first two $LIPs$ developed for solving $GRSADSP/1$ are based on the generic LIP shown in Figure 3.1, using the shift and unit neighborhoods as the neighborhood structures. These algorithms are called LIP/S and LIP/U, respectively.

The other two algorithms are called *Composite Shift LIP* ($CLIP/U$) *and Composite Unit LIP* ($CLIP/S$), which are described, respectively, in Figures 3.2 and 3.3.

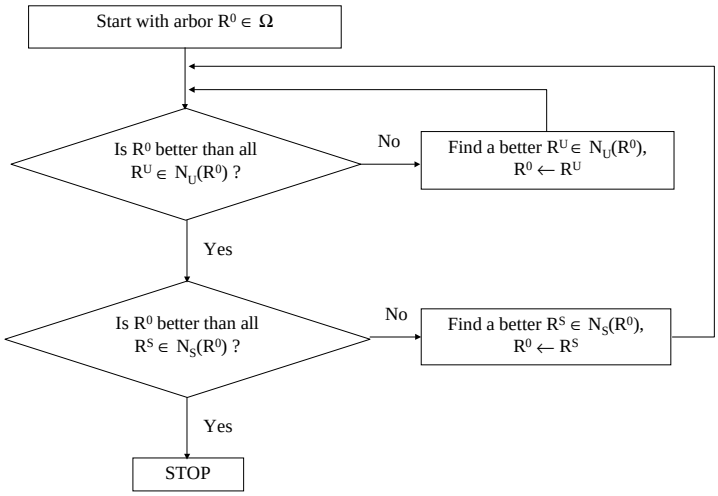

Figure 3.2: CLIP/S algorithm.

Each of these composite $LIPs$ has two nested loops, performing local search with a different neighborhood structure within each loop. The idea is that before starting the local search with an arbor with respect to one neighborhood, a local search with respect to the other neighborhood is performed on that arbor.

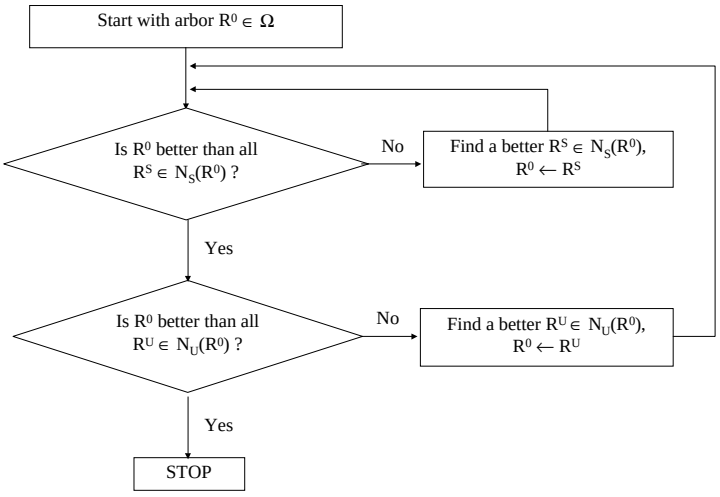

Figure 3.3: CLIP/U algorithm.

3.4 Comparison of the Algorithms

Table 3.1 presents a comparison of the four $LIPs$ for a specific instance when *the running times were equalized for one minute*. In this instance, the demand schedule is as in Table 1.1, and the incoming lumber distribution is as in Table 6.5 on page 56, with $L = 24''$.

Multiple runs of each of the four $LIPs$ were performed for one minute with different random starting arbors in each run. The total number of runs within that time is shown in the second row of the table. The numbers in the remaining rows are the values of the corresponding statistical measures in the first column for the expected waste of the locally optimal solutions found by the algorithms.

It can be observed from Table 3.1 that the best solution value was found by $CLIP/S$. The other statistical measures of $CLIP/S$ were also better than those of other $LIPs$.

	LIP/S	LIP/U	$CLIP/U$	$CLIP/S$
Number of runs	364	753	102	196
Mean	2,727	2,088	1,888	1,843
Std. deviation	632	249	152	142
Maximum	8,499	3,858	2,543	2,512
Minimum (best found)	1,694	1,665	1,659	1,645

Table 3.1: Comparison of the local improvement procedures.

Furthermore, we observed a similar behavior on several other instances of the problem. Therefore, we decided to compare the other two heuristics presented in the following chapters only with $CLIP/S$ in our computational experiments.

Chapter 4

Simulated Annealing

This chapter presents a brief discussion of the simulated annealing algorithm followed by its application to $GRSADSP/1$.

4.1 Introduction and Description of the Algorithm

Simulated Annealing (SA) is a *heuristic strategy,* or *a family of algorithms* which can be viewed as *randomized* local search. It has been applied successfully to many optimization problems occurring in a variety of areas such as sequencing and scheduling, $VLSI$ design, image processing, molecular biology, and design of statistical experiments [2].

The main disadvantage of traditional local search methods is their likelihood of finding a local, rather than global optimum: they never move to a new solution unless the direction is *downhill,* i.e., the value of the cost function decreases[1]. However, such a strategy often results in convergence to a locally optimal solution. A number of approaches have been suggested which partially overcome this problem: the algorithms can be repeated using several different starting solutions, or the complexity of the neighborhoods can be increased, widening the scope of the search. Unfortunately, none of these variants has proven to be entirely satisfactory [12].

[1] *The discussion of the algorithms in this section is within the context of a minimization problem.*

SA attempts to avoid this entrapment in poor local optima by allowing occasional *uphill* moves. This is done under the influence of a random number generator and a control parameter T, called the *temperature.* The basic idea is that a neighbor S' of the current solution S is accepted according to the following acceptance criterion, where $f(S)$ is the value of the objective function evaluated at S:

$$p_T(accept\ S') = \left\{ \begin{array}{c} 1,\ if\ \Delta = f(S') - f(S) \leq 0 \\ e^{-\ \Delta/T},\ if\ \Delta > 0 \end{array} \right\} \tag{4.1}$$

Notice that $e^{-\Delta/T}$ will be a number in the interval $(0, 1)$ when Δ and T are positive, and can be interpreted as a probability that depends on Δ and T. The probability that 'an uphill move of size Δ will be accepted' decreases as the temperature is lowered, and small uphill moves have higher probabilities of acceptance than larger ones for a fixed temperature T. As typically implemented, the temperature T is initially kept high, and decreased gradually as the search continues [6].

The algorithm typically involves a pair of nested loops and two additional parameters: a cooling ratio, $r,\ 0 < r < 1$, and an integer called *temperature length*, denoted by *nreps*. A generic SA implementation is presented below.

$SA()$

1. Get an initial solution S
2. Get an initial temperature $T > 0$
3. Repeat
   (a) Repeat
      i. Randomly select $S' \in N(S)$
      ii. Let $\Delta = f(S') - f(S)$
      iii. If $\Delta \leq 0$ then $S \leftarrow S'$ (downhill move)

iv. If $\Delta > 0$

Generate a random number, x, in the range $(0, 1)$

If $x < e^{-\Delta/T}$ then $S \leftarrow S'$ (uphill move)

(b) Until $iteration\_counter = nreps$

(c) $T \leftarrow rT$ (reduce temperature)

4. Until the system is not $frozen$

5. S is the approximation to the optimal solution

In the inner loop, *nreps* iterations are made with the current temperature T, and then the temperature is reduced according to the following reduction function[2]:

$$T = rT$$

The outer loop is executed until no further decrease in $f(S)$ seems likely (i.e., until the system is *frozen*).

Notice that any local optimization algorithm can be converted into SA by generating the neighbors randomly and allowing the acceptance of inferior solutions according to the criterion 4.1. The level of acceptance of uphill moves then depends on the magnitude of the increase in the cost function and on the temperature [12].

Aarts and Korst [1] report the following conclusions regarding the performance of the SA algorithm:

1. The algorithm has the potential of finding high quality solutions at the cost of substantial computational efforts.

2. The algorithm is robust; i.e. its final solution does not strongly depend on the choice of the initial solution.

[2] *This approach to temperature reduction is commonly referred to as "geometric cooling".*

3. The quality of the final solution is predominantly dependent on the parameter r which determines the speed at which the temperature is lowered. The values of the parameters determining the initial and final values of the control parameter play only a minor role as long as they are chosen with a reasonable extent of accuracy.

4. The average case running time is close to the worst-case running time.

4.2 A Physical Analogy

The SA algorithm is based on ideas from statistical mechanics and motivated by an analogy between *physical annealing* of solids and optimization problems.

Physical annealing refers to the process of finding low energy levels of a solid by first melting it and then cooling it back into the solid state. The purpose of this process is to minimize the irregularities in the crystal structure of the solid substance. If the cooling is done very slowly, large crystals with low energy levels can be grown. However, if the cooling is done quickly, there is a high possibility that widespread irregularities will be locked into the crystal structure and the trapped energy level will be much higher than a perfectly structured crystal [6].

In 1953, Metropolis et al. [10] published an algorithm to simulate the annealing process of solids as described above. Almost thirty years later, Kirkpatrick et al. [8] suggested that this type of simulation can be used to solve optimization problems, observing that physical annealing can be viewed analogous to optimization: (1) the energy of the system corresponds to the objective function in the optimization problem, (2) the different states of the substance correspond to different solutions, and (3) the minimum energy state corresponds to an optimal solution.

The name *simulated annealing* thus refers to the application of the simulation technique developed by Metropolis et al. for annealing of solids to optimization problems.

The next section outlines the details of our SA implementation for $GRSADSP/1$.

4.3 Implementation of the Algorithm

As stated earlier, SA is a family of algorithms and should not be regarded as a specific algorithm like the traditional descent methods. Although the basic features are presented in section 4.1, various decisions still have to be made for the *meanings* of the undefined terms and the parameter values in order to implement the algorithm within the context of a particular problem. These decisions can be classified as *problem-specific* and *generic* decisions [6], which can be summarized as follows:

1. Problem-Specific Decisions:

   (a) What is a solution?

   (b) What are the neighbors of a solution?

   (c) What is the cost of a solution?

   (d) How is an initial solution determined?

2. Generic Decisions:

   (a) How is an initial temperature determined?

   (b) How is the cooling ratio r determined?

   (c) How is the temperature length *nreps* determined?

   (d) How do we know the system is frozen?

These issues are addressed below for our SA implementation for $GRSADSP/1$.

4.3.1 Problem-specific Decisions

Recall that in $GRSADSP/1$, we are given a frequency distribution of uncut lumber boards of various widths and a demand schedule for certain finished widths, and asked to find an arbor, (i.e. a sequence of cutting channels along a rotating saw shaft), that minimizes the total waste during the cutting job.

In our annealing scheme for $GRSADSP/1$, any sequence of cutting channels along the saw shaft represents a solution (i.e., an arbor). The cost of a solution is then the expected waste corresponding to that arbor.

As the neighborhood structure, the *unit neighborhood* defined in section 3.2.2 will be used. Thus, two arbors will be *neighbors* if one can be obtained from the other by changing the width of a cutting channel to another finished width and repairing the resulting arbor. Finally, the algorithm is started with randomly generated feasible full-length arbors.

4.3.2 Generic Decisions

The algorithm starts with an initial temperature which approximately corresponds to a specified fraction of initially accepted moves, called $INITPROB$. This starting temperature is determined by several abbreviated trial runs for the current instance. The cooling ratio, r, is specified as a parameter called $TEMPFACTOR$.

As for the temperature length, first a parameter called $SIZEFACTOR$ is defined, and then the temperature length, *nrep*, is set to be $N \times SIZEFACTOR$, where N is the approximate neighborhood size. Note that N will approximately be equal to $n(n-1)$, where n is the approximate number of slots (i.e., cutting channels) on the saw shaft defined in section 2.2.3. This way, the temperature length always remains proportional to the number of neighbors, and a wide range of instances is handled with a fixed value of $SIZEFACTOR$.

The last generic decision is the stopping criterion, or when to declare the system to be frozen. For this purpose, a counter is maintained that is incremented by one each time a temperature is completed for which the percentage of accepted moves is $MINPERCENT$ or less. The counter is reset to zero whenever a solution is found that is better than the previous champion. If the counter ever reaches $MAXCOUNTER$, the algorithm is terminated (these generic decision criteria are adopted from Johnson et al. [6])

Parameter	*Value*
$INITPROB$	70%
$TEMPFACTOR$	0.99
$SIZEFACTOR$	25
$MINPERCENT$	3%
$MAXCOUNTER$	5

Table 4.1: Simulated annealing parameter values.

4.3.3 Parameter Values

We performed extensive computational experiments on numerous instances to optimize the values of the parameters defined above. Table 4.1 presents the values that we have settled on for the five parameters in our annealing implementation.

Chapter 5

Genetic Algorithms

In this chapter, fundamental features of genetic algorithms are discussed and their application to $GRSADSP/1$ is described.

5.1 Introduction

Genetic Algorithms (GAs) are a family of computational models founded upon the alleged principle of evolution, i.e. survival of the fittest. Although an analogy can be established between GAs and the principle of natural selection, genetic algorithms can be viewed as search methods where better and better populations of solutions, (i.e. sets of solutions), are produced for a particular problem within each iteration.

The most important difference between GAs and other search methods is the number of solutions maintained during the execution of the algorithm: whereas most search methods maintain a *single* solution from start to termination, GAs always maintain and manipulate a *population* of solutions.

GAs have recently received remarkable attention, and the interest in GAs is currently undergoing exponential growth. Some reasons for this increasing popularity are:

1. GAs do not make strong assumptions about the objective function such as linearity, convexity, differentiability or such, in contrast to most other conventional

optimization techniques.

2. They are general purpose search methods which can be employed to maintain a balance between exploration and exploitation of the search space.

3. It is generally easy to change a GA to model variations of the original problem, whereas many other heuristics usually require substantial changes for minor modifications [11, 12].

5.2 Description of the Algorithm

In the strict sense, the term "genetic algorithm" refers to the model introduced by Holland [5]. Other forms of simulated evolution are usually called "evolutionary algorithms". However, as a common practice in the area, we will use the terms "evolution algorithms" and "genetic algorithms" synonymously throughout the chapter. A generic genetic algorithm is presented below.

$GA()$

1. Set generation counter $t \leftarrow 0$

2. Create the initial population, $Pop(t)$, of size N.

3. Evaluate $Pop(t)$; i.e. determine the fitness of each individual in the population by applying the objective function

4. Increment to the next generation, $t \leftarrow t + 1$

5. Create a new population, $Pop(t)$, by randomly selecting N individuals (not necessarily distinct) from the previous population, $Pop(t-1)$. Then,

    i. Randomly select R parents from the new population to generate the new children by the application of the genetic operators

ii. Evaluate the fitness of the newly formed children by applying the objective function

6. If the termination criterion has not yet been met, go to step 4

7. Print out the best solution encountered during the execution

---

In a GA, each solution, or individual, in the population is described by a vector of variables, called the *chromosome representation*. The first step in the GA is to initialize the population, which is typically done randomly. Once the initial population is generated, each individual is evaluated using the objective function to determine its fitness value. A subset of the population is then selected to produce the next generation: a probabilistic selection is performed such that the fittest individuals have an increased chance of being selected (note that an individual can be selected more than once at the selection step).

These parents then undergo reproduction to produce a new population using the genetic operators. The GA moves from generation to generation in this fashion until some specified stopping criterion is met [7].

5.3 Implementation of the Algorithm

There are six fundamental issues in implementing a genetic algorithm for a particular problem: chromosome representation, initialization of the population, evaluation measure, selection mechanism, genetic operators, and termination criteria. These issues are introduced below and described specifically for our GA implementation for solving $GRSADSP/1$.

5.3.1 Chromosome Representation

As stated earlier, potential solutions to the problem of interest are represented by a vector of variables typically referred as "chromosomes"; as an analogy with the simulated biological process. The chromosome representation describes the individuals in the population maintained by the *GA*. Determining a suitable representation of the chromosomes is the first major step within the context of *GA* design.

The chromosome representation in our *GA* implementation for $GRSADSP/1$ is straightforward: any sequence of cutting channels which forms a feasible full-length arbor constitutes a chromosome.

5.3.2 Initialization of the Population

GAs maintain and manipulate a population of solutions to the problem. Therefore, after choosing an appropriate chromosome representation, the population has to be initialized (step 2 of the generic algorithm) to start the genetic search. As the initialization process, the size of the population (i.e., the number of individuals in the population), and the method to create the individuals in the initial population need to be determined.

In our implementation, the members of the initial population are randomly generated feasible full-length arbors. The size of the initial population (which is kept fixed throughout the execution of the algorithm) is specified in Table 5.1 on page 46.

5.3.3 Evaluation Measure

The fitness of the individuals has to be evaluated once the initial population has been created. At this step, the evaluation function is used to determine which individuals are better to help guide the search. The minimum requirement for the evaluation function is the mapping of the population into a totally ordered set.

Although determination of this evaluation function may require some ingenuity in some problems, it is straightforward in our case: the fitness of a chromosome is simply

the expected waste corresponding to the arbor that the chromosome represents.

5.3.4 Selection Mechanism

Once the current population has been evaluated, a new population of the same size is selected from the previous generation. The selection strategy of the new population is of extreme importance in the GA, where the purpose is to allow better individuals to be selected more often to reproduce in the next generations (note that the individuals in the new generation need not be distinct).

There are two important issues in the evolution process of the genetic search: population diversity and selectivity pressure (in some sense, this is another variation of the idea of exploration versus exploitation). These factors are closely related: an increase in the selectivity pressure decreases the diversity of the population and vice versa. In other words, strong selectivity pressure "supports" the premature convergence of the genetic search, and a weak selectivity pressure can make the search ineffective. Thus, it is important to strike a balance between these two factors. The fundamental tools to achieve this balance are sampling mechanisms [11].

In our GA implementation, the new population is selected according to a *ranking method*: the individuals for the new population are selected based on their ranks in the current population, i.e. their relative fitness compared to that of the other individuals in the population. This way the absolute difference between the individuals' fitnesses is ignored, and only their rank in the population is taken into account.

In our rank-based selection mechanism, the "normalized geometric distribution" suggested by Michalewicz [11] is adopted. In this method, individuals in the population are first ranked from best to worst according to their fitness values. Then, each individual is assigned a probability of selection proportional to the individual's rank in the sorted population based upon the following distribution:

$$Prob[selecting\ the\ rth\ individual] = q'(1-q)^{r-1}$$

where

$$\begin{aligned} 0 &\leq q \leq 1, \\ q' &= \frac{q}{1-(1-q)^N} \end{aligned}$$

and

$r \equiv$ rank of the individual, where 1 is the best,

$N \equiv$ number of individuals in the population.

Notice that lower values of q put less selectivity pressure on selecting the best individuals and vice versa. The motivation for selecting the rank-based selection mechanism is that it prevents scaling problems and allows a significant flexibility in balancing the two opposing search directions, i.e. population diversity and selectivity pressure [11]. The value of q used in our GA is specified in Table 5.1 on page 46.

5.3.5 Genetic Operators

After a new generation is selected biased toward the better individuals of the previous one, reproduction is carried out by the application of genetic operators on a selected number of parents (step 5 (ii) of the algorithm).

There are two basic types of genetic operators: *mutation* and *crossover*. Mutation operators alter, or mutate, one parent by changing one or more variables by some random amount to form one offspring. Crossover operators, on the other hand, combine information from two parents such that the two children contain a "likeness" (a set of building blocks) from each parent [7]. The application of these operators to our problem is described below.

Mutation Operator

Mutation operators inject genetic diversity into the population, thus allowing the algorithm to explore new regions of the search space. In our algorithm, *random uniform*

mutation is employed: each cutting channel of the arbors in the chromosome representation has a fixed probability p_M to be mutated. For each cutting channel in each of the chromosomes, a real number between 0 and 1 is generated randomly. If this number is less than p_M, a cutting channel different from the current channel width is generated randomly and the current channel is replaced with the new one. The value for p_M is specified in Table 5.1 (note that this mutation operator is analogous to the unit neighborhood structure defined in section 3.2.2). The resulting arbor is then "repaired" if necessary as described in section 3.2.2.

Crossover Operator

The purpose of crossover is to combine good portions of the parents into one child, creating an even better child. However, the children produced by the crossover may be worse than either parent. In that case, these new individuals will eventually die off in the subsequent generations. But even these poor children may still contain some good building blocks that might be passed to later generations.

In our *GA*, *a single point crossover operator* is implemented: each individual in the population has a fixed probability p_C to be chosen as a parent to be used in the crossover. For each individual, a random number between 0 and 1 is generated and compared to p_C. If that number is less than p_C, the crossover operator is applied to this individual. Of those selected individuals, two individuals are paired up, and the crossover operator is applied to this pair of parents: first a cut point in the first parent is selected randomly. The cut point in the second parent is then determined by taking the widths of the cutting channels on the second parent into account. Next, the two parents are split at these points, and the children are created by concatenating different segments from the parents. Finally, the resulting children are repaired. The value for p_C is specified in Table 5.1.

A simple example of this operation is presented below.

Suppose that for the demand schedule in Table 1.1 with $L = 19''$, the following two

arbors are selected as the parents in the crossover:

$$\begin{aligned}\mathbf{R}_1 &= (\mathbf{w}_2\mathbf{w}_3 w_4 w_1 w_5 w_5 w_1 w_1 w_3 w_2)\\ \mathbf{R}_2 &= (\mathbf{w}_2\mathbf{w}_1\mathbf{w}_1 w_4 w_3 w_3 w_2 w_5 w_4 w_1 w_1).\end{aligned}$$

1. Select a cutting channel, u, on the first arbor: $u \leftarrow 2$.

2. Determine the smallest channel index v for the second arbor such that

$$\sum_{m=1}^{u} R_{m1} \leq \sum_{n=1}^{v} R_{n2}$$

(In other words, find the smallest cutting channel index v on the second arbor such that the length of the arbor segment from the first channel to channel v is larger than or equal to the length of the arbor segment on the first arbor from the first channel to the channel u). In this example, $v = 3$.

3. Split the first arbor at channel u and the second arbor at channel v, and swap the remaining arbor segments of the two arbors to obtain the following two children:

$$\begin{aligned}\mathbf{R}_{1'} &= (\mathbf{w}_2\mathbf{w}_3 w_4 w_3 w_3 w_2 w_5 w_4 w_1 w_1)\\ \mathbf{R}_{2'} &= (\mathbf{w}_2\mathbf{w}_1\mathbf{w}_1 w_4 w_1 w_5 w_5 w_1 w_1 w_3 w_2).\end{aligned}$$

(The above children will then undergo the repair operation before the next population is selected.)

Arbor Infeasibility

Notice that after the application of genetic operators and the repair algorithm, some of the arbors may not be feasible any more (i.e., one or more of the finished widths in the demand schedule may not be available anywhere on the arbor). In that case, the fitness values of those arbors are set to ∞. This essentially means that the infeasible arbors will be of the lowest rank when the next population is generated, and thus their chances to survive in the new population will be significantly reduced.

5.3.6 Elitist Model Implementation

Several advanced techniques have been proposed for improving the performance of *GAs* [11]. In our implementation, we employ a special technique called the "*elitist model*" in which the worst individual in the current population is replaced by the best individual of the previous population before the next population is selected. This way, the *GA* never gets worse in terms of best solution. It has been shown that *GAs* employing an elitist model perform better than those that do not [7].

5.3.7 Termination Criterion

After the initial population is created and evaluated, the genetic search continues from generation to generation, selecting and reproducing parents until a pre-determined termination criterion is met.

In our *GA* implementation, the algorithm is terminated after a specific number of generations, which is a frequently used stopping criterion in GAs. The maximum number of generations used as the termination criterion is presented in Table 5.1.

5.3.8 Parameter Values

To optimize the GA parameter values, we performed extensive computational experiments on numerous instances. Table 5.1 presents the values for the parameters that we

Parameter	*Value*
Population Size, N	50
Mutation Probability, P_M	0.05
Crossover Probability, P_C	0.25
Selectivity Coefficient, q	0.05
Maximum Number of Generations	300

Table 5.1: Genetic algorithm parameter values.

decided to use in our GA implementation.

Chapter 6

Computational Experiments

This chapter presents a comparison of the heuristic methods developed solving GRSADSP/1 on an empirical basis.

6.1 Introduction

In the last three chapters, several heuristic approaches were discussed for solving combinatorial problems: local improvement procedures (LIPs), simulated annealing (SA), and genetic algorithms (GAs). Specific implementation details of these heuristics were also described for solving $GRSADSP/1$[1].

Our objective in this chapter is to compare the performances of these heuristics in terms of solution quality and computational requirements by solving specific instances of the problem.

Note that as discussed in section 3.4, $CLIP/S$ performed better than other proposed $LIPs$ for the problem instances that were studied. So, we decided to compare the performances of SA and GA only with $CLIP/S$ in our computational experiments.

In the rest of this chapter, quality measures of an arbor are introduced, and the

[1] *The heuristics were implemented in C++, and the computational experiments were carried out on a 380 MHz personal computer.*

data sets used in the computational experiments are presented. Then the methodology of experimentation is described, and the observations are discussed.

6.2 Quality Measures of an Arbor

Recall that the objective in $GRSADSP/1$ is to minimize the total board feet of waste produced in the cutting job. So, the key quality measure of an arbor j is YP_j, i.e. the yield percentage corresponding to the arbor j. However, as discussed in section 1.2, another major concern in the problem is the overage, i.e. the production in any finished width above its corresponding demand, since it is not desirable to produce finished widths that significantly exceed their corresponding demands. Therefore, OP_j, i.e. the overage percentage corresponding to the arbor j, is another important quality measure.

Theoretical bounds on YP_j and OP_j can clearly be specified as: (1) the yield limit, defined in section 2.1, as an upper bound on YP_j, and (2) 0% as a lower bound on OP_j; i.e., theoretically, the best one can do is to produce no overage at all.

So, to determine the quality of an arbor j for a specific instance of $GRSADSP/1$, the expected total waste corresponding to the arbor j can be computed and compared to the optimal solution's value, if an optimal solution to the instance is already known. Otherwise, i.e. if the optimal solution is not known, the values of YP_j and OP_j can be used as benchmarks, comparing them to those of the theoretical bounds.

6.3 The Data Sets

The selection of the data sets that are used in comparing the performance of a heuristic procedure with those of exact algorithms and also with those of other heuristics is extremely important, since incorrectly selected data sets may lead to generalizations that are inconsistent with reality and thus to erroneous conclusions. For this reason, we obtained several instances of the problem that have actually been encountered in the

industry and generated other data sets randomly that have the same characteristics as these instances.

As stated earlier, a typical instance of the problem consists of (1) a set of required finished widths and their corresponding board feet demands (demand schedule), (2) a frequency distribution of incoming lumber boards, (3) a shaft length, and (4) a blade width.

In the computational experiments, two different incoming lumber distributions were used. These distributions are shown in Tables 6.5 and 6.6. Notice that the first distribution contains 34 distinct incoming board widths ranging from $3''$ to $8\frac{1}{2}''$, and the second distribution contains 36 distinct incoming board widths ranging from $2\frac{3}{8}''$ to $14\frac{1}{2}''$.

The demand schedules, on the other hand, are presented in Tables 6.1, 6.2, 6.3, and 6.4. Each of these tables contains 2 different demand schedules with the same number of distinct finished widths and a name corresponding to each demand schedule. Note that the number of distinct finished widths in the schedules varies from 4 to 7.

Two different values for the shaft length were considered: $24''$ and $36''$. For the blade width, only the value of $1/8''$ was considered. Hence, a total of $8 \times 2 \times 2 = 32$ different instances of the problem were used in the computational experiments.

6.4 Methodology

The comparison of the heuristics were essentially made for *equal running times* because the most important criterion for the company performing the cutting job is the quality of the best arbor that they can get in a fixed amount of computational time.

Furthermore, a solution to the problem needs to be obtained in a relatively short time (typically, several cutting jobs are performed in any given day). Therefore, multiple runs of each of the three heuristics were performed for 1, 3, 5, and 10 minutes, respectively, for each data set. The values of the best solution obtained by each heuristic (if a solution

was at all obtained within the given time limit) within each of these four time periods were then reported. Running the heuristics for different time periods allowed observing the improvement of the best solutions obtained by each heuristic over time, and also how each heuristic compared within a given amount of time.

The total enumeration procedure was also executed for the small instances to obtain the corresponding optimal solutions. The best solutions obtained by the heuristics were then compared to the optimal solution for these instances.

Furthermore, the yield and overage percentages of the best solution found (if any) by each of the heuristics for all of the data sets within the first 10 minutes were reported. This way, we were able to draw conclusions about the quality of the best solutions obtained for the data sets for which the optimal solutions were not known.

Finally, for each data set, the best solution value obtained by multiple runs of $CLIP/S$ was compared to the solution value obtained by a software package which is currently used in the industry for solving $GRSADSP$. This package is called $GRADS$ (Gang-Rip Arbor Design System) and is a commercial implementation of the column generation procedure of linear programming proposed by Fathi et al. [3]. In this comparison, both $GRADS$ and $CLIP/S$ were run for *one* minute.

Tables 6.7, 6.9, 6.8, and 6.10 compare the heuristics for $L = 24''$ and $L = 36''$, and the two incoming lumber distributions shown in tables 6.5 and 6.6. Each of the eight columns in these tables correspond to the demand schedules shown in tables 6.1 through 6.4.

The second row in each of these tables presents the optimal solutions for the data sets for which we were able to complete the total enumeration procedure within 2 hours.

The next three rows show the expected total board feet of waste corresponding to the best arbor found (if any) by multiple runs of each heuristic *within the first* 1 minute (notice that for all of the data sets, SA could not give any solution within the first minute).

The following nine rows show the expected total board feet of waste corresponding

to the best arbor found (if any) by multiple runs of each heuristic *within the first* 3, 5, and 10 minutes, respectively.

The next row presents the yield limit for the corresponding data sets, and the following six rows show the yield and overage percentages of the best arbor found by multiple runs of each heuristic within the first 10 minutes.

Tables 6.11, 6.13, 6.12, and 6.14 compare $GRADS$ and $CLIP/S$ for $L = 24''$ and $L = 36''$, and the two incoming lumber distributions shown in tables 6.5 and 6.6.

The third row in each of these tables presents the number of arbors in the solution obtained by $GRADS$ after running the software for one minute. The next two rows show the expected total board feet of waste and the overage percentage in the cutting job, respectively, if the operation is performed by employing the set of arbors obtained by $GRADS$.

The last two rows, on the other hand, present the expected total board feet of waste and the overage percentage, respectively, of the best arbor obtained by multiple runs of $CLIP/S$ in one minute.

6.5 Observations

We have made the following observations about our experimentation with the data sets:

1. The time required by the heuristics for one complete run ranged from (1) $1/10th$ of a second to 2 seconds for $CLIP/S$, (2) 5 to 15 seconds for GA, and (3) 2 to 35 minutes for SA (note that for SA, only the solutions obtained within the first 10 minutes were reported).

2. Across all the four time periods considered, the best solution values obtained by $CLIP/S$ were *at least as good as* the best solution values obtained (if any) by the other two heuristics for at least 30 data sets out of the 32 data sets. For the data sets for which either of the other two heuristics found a better solution than

$CLIP/S$, that solution's value was *never better by more than 1%* of the solution's value obtained by $CLIP/S$.

3. For the data sets for which the optimal solution is known, $CLIP/S$ found the optimal solution in at most 3 minutes.

4. For these data sets for which the optimal solutions are known, the yield percentages of the corresponding optimal solutions were below the theoretical yield limit up to *2%*, and the overage percentages were as high as *29%* (the yield and overage percentages of the optimal solutions are not reported separately, since in each case, $CLIP/S$ was able to find the optimal solution within the first 10 minutes).

5. For the data sets for which the optimal solutions are not known, the best solutions obtained by $CLIP/S$ within the first 10 minutes had yield percentages which were *always at most 2% less than* the theoretical yield limit and very low overage in most cases.

6. For all of the 32 data sets, the best solution value obtained by $CLIP/S$ within the first 1 minute *never improved by more than 2%* of that value within the next 9 minutes.

7. Across all the four time periods, the best solution values obtained by GA and SA (if any) were *never more than 6% above* of those obtained by $CLIP/S$.

8. Although the number of arbors in the solutions obtained by $GRADS$ were high (as many as the number of finished widths in the demand schedule in some cases), the expected total board feet of waste and the overage percentage of these solutions were only *slightly better* than those of the solutions obtained by $CLIP/S$.

Demand Schedule	Finished width (inches)	Demand (board feet)
4A	1	20
	$1\frac{7}{8}$	600
	$2\frac{1}{8}$	150
	$3\frac{5}{8}$	80
4B	$1\frac{1}{4}$	250
	2	200
	$2\frac{1}{2}$	175
	3	145

Table 6.1: Demand schedules with four distinct cut widths.

Demand Schedule	Finished width (inches)	Demand (board feet)
5A	1	200
	$1\frac{3}{8}$	500
	$1\frac{7}{8}$	400
	2	100
	3	150
5B	1	20
	$1\frac{7}{8}$	600
	$2\frac{1}{8}$	150
	$2\frac{5}{8}$	30
	$3\frac{5}{8}$	50

Table 6.2: Demand schedules with five distinct cut widths.

Demand Schedule	Finished width (inches)	Demand (board feet)
6A	$1\frac{3}{4}$	104
	2	142
	$2\frac{1}{4}$	867
	$2\frac{1}{2}$	149
	3	80
	$3\frac{1}{2}$	208
6B	$1\frac{1}{4}$	210
	$1\frac{1}{2}$	185
	$1\frac{3}{4}$	175
	2	160
	$2\frac{1}{2}$	155
	3	170

Table 6.3: Demand schedules with six distinct cut widths.

Demand Schedule	Finished width (inches)	Demand (board feet)
7A	1	110
	$1\frac{1}{2}$	125
	2	120
	$2\frac{1}{4}$	145
	3	135
	$3\frac{1}{4}$	170
	$3\frac{1}{2}$	122
7B	$1\frac{1}{4}$	230
	$1\frac{1}{2}$	210
	2	245
	$2\frac{1}{4}$	235
	$2\frac{1}{2}$	220
	3	210
	$3\frac{1}{4}$	215

Table 6.4: Demand schedules with seven distinct cut widths.

Board width (inches)	Linear feet in stock	Board width (inches)	Linear feet in stock
3	432	$3\frac{1}{2}$	576
$3\frac{5}{8}$	144	$3\frac{3}{4}$	720
$3\frac{7}{8}$	288	4	288
$4\frac{1}{8}$	576	$4\frac{1}{4}$	144
$4\frac{3}{8}$	144	$4\frac{1}{2}$	288
$4\frac{5}{8}$	144	$4\frac{3}{4}$	288
$4\frac{7}{8}$	432	5	720
$5\frac{1}{8}$	1152	$5\frac{1}{4}$	720
$5\frac{3}{8}$	576	$5\frac{1}{2}$	1152
$5\frac{5}{8}$	1152	$5\frac{3}{4}$	1152
$6\frac{1}{8}$	144	$6\frac{3}{8}$	432
$6\frac{1}{2}$	144	$6\frac{3}{4}$	144
$6\frac{7}{8}$	576	7	432
$7\frac{1}{8}$	144	$7\frac{1}{4}$	288
$7\frac{1}{2}$	144	$7\frac{5}{8}$	288
8	144	$8\frac{1}{8}$	144
$8\frac{1}{4}$	144	$8\frac{1}{2}$	144

Table 6.5: Incoming lumber distribution no. 1.

Board width (inches)	Linear feet in stock	Board width (inches)	Linear feet in stock
$2\frac{3}{8}$	102	3	102
$3\frac{3}{8}$	264	$3\frac{1}{2}$	306
$3\frac{5}{8}$	102	$3\frac{3}{4}$	580
4	439	$4\frac{1}{4}$	287
$4\frac{3}{8}$	102	$4\frac{1}{2}$	408
$4\frac{5}{8}$	204	$4\frac{3}{4}$	391
5	560	$5\frac{1}{4}$	376
$5\frac{3}{8}$	256	$5\frac{1}{2}$	102
$5\frac{3}{4}$	102	6	408
$6\frac{1}{4}$	102	$6\frac{1}{2}$	204
$6\frac{5}{8}$	102	$6\frac{3}{4}$	510
7	672	$7\frac{1}{4}$	612
$7\frac{3}{8}$	102	$7\frac{1}{2}$	612
$7\frac{3}{4}$	257	8	460
$8\frac{1}{4}$	306	$8\frac{5}{8}$	60
$8\frac{3}{4}$	102	10	102
11	102	$11\frac{1}{2}$	102
$11\frac{5}{8}$	102	$14\frac{1}{2}$	102

Table 6.6: Incoming lumber distribution no. 2.

Demand schedule	4A	4B	5A	5B	6A	6B	7A	7B
Optimal solution	151	91	-	135	205	-	-	-
Best sol. values within 1 min								
CLIP/S	151	92	166	136	205	122	102	175
GA	151	94	176	136	209	127	107	175
SA	-	-	-	-	-	-	-	-
Best sol. values within 3 min								
CLIP/S	151	91	164	135	205	122	101	171
GA	151	94	165	135	205	126	107	175
SA	151	91	-	135	205	-	-	-
Best sol. values within 5 min								
CLIP/S	151	91	164	135	205	122	101	171
GA	151	92	164	135	205	125	107	175
SA	151	91	-	135	205	-	-	-
Best sol. values within 10 min								
CLIP/S	151	91	164	135	205	120	101	171
GA	151	91	164	135	205	124	102	171
SA	151	91	162	135	205	122	106	171
YP of best sol. within 10 min								
(Yield Limit)	90.4	90.1	90.9	91.4	91.7	90.9	92	91.7
CLIP/S	89.7	89.7	89.4	89.2	90.5	90.1	90.9	90.9
GA	89.7	89.7	89.4	89.2	90.5	89.9	90.9	90.9
SA	89.7	89.7	89.4	89.2	90.5	90.0	90.8	90.9
OP of best sol. within 10 min								
CLIP/S	28.6	3.4	2.6	28.4	25.8	4.1	9.3	9.1
GA	28.6	3.4	2.6	28.4	25.8	5.1	10.1	9.1
SA	28.6	3.4	1.8	28.4	25.8	4.0	13.0	9.1

Table 6.7: Comparison of the heuristics for L=24" with lumber distribution no. 1.

Demand schedule	4A	4B	5A	5B	6A	6B	7A	7B
Optimal solution	-	-	-	-	-	-	-	-
Best sol. values within 1 min								
CLIP/S	151	88	162	135	190	112	91	156
GA	151	88	162	135	190	117	94	161
SA	-	-	-	-	-	-	-	-
Best sol. values within 3 min								
CLIP/S	151	88	162	135	190	112	89	156
GA	151	88	162	135	190	114	90	161
SA	-	-	-	-	-	-	-	-
Best sol. values within 5 min								
CLIP/S	151	88	162	135	190	112	89	156
GA	151	88	162	135	190	114	90	159
SA	-	-	-	-	-	-	-	-
Best sol. values within 10 min								
CLIP/S	151	88	162	135	187	112	89	156
GA	151	88	162	135	190	113	90	158
SA	151	88	-	-	190	-	-	-
YP of best sol. within 10 min								
(Yield Limit)	90.4	90.1	90.9	91.4	91.7	90.9	92	91.7
CLIP/S	89.7	89.8	89.4	89.2	90.6	90.6	91.4	91.4
GA	89.7	89.8	89.4	89.2	91.2	90.5	91.5	91.1
SA	89.7	89.8	-	-	91.2	-	-	-
OP of best sol. within 10 min								
CLIP/S	28.6	0.9	1.1	28.4	17.5	3.4	2.4	6.1
GA	28.6	0.9	1.1	28.4	28.2	2.0	6.0	3.5
SA	28.6	0.9	-	-	28.2	-	-	-

Table 6.8: Comparison of the heuristics for L=36" with lumber distribution no. 1.

Demand schedule	4A	4B	5A	5B	6A	6B	7A	7B
Optimal solution	124	86	-	119	179	-	-	-
Best sol. values within 1 min								
CLIP/S	124	86	168	119	179	115	96	171
GA	124	86	171	119	179	121	102	174
SA	-	-	-	-	-	-	-	-
Best sol. values within 3 min								
CLIP/S	124	86	168	119	179	115	96	170
GA	124	86	170	119	179	119	102	174
SA	124	86	-	119	179	-	-	-
Best sol. values within 5 min								
CLIP/S	124	86	168	119	179	115	96	170
GA	124	86	169	119	179	116	100	174
SA	124	86	-	119	179	-	-	-
Best sol. values within 10 min								
CLIP/S	124	86	168	119	179	115	96	170
GA	124	86	168	119	179	116	100	172
SA	124	86	171	119	179	116	96	174
YP of best sol. within 10 min								
(Yield Limit)	91.3	91.2	91.4	91.6	92.1	91.7	92.3	92.0
CLIP/S	89.5	90.2	89.2	89.7	91	90.6	91	90.7
GA	89.5	90.2	89.2	89.7	91	90.3	91	90.6
SA	89.5	90.2	89.1	89.7	91	90.4	91	90.4
OP of best sol. within 10 min								
CLIP/S	24.6	3.3	2.8	22.5	16.5	2.9	5.4	5.6
GA	24.6	3.3	2.8	22.5	16.5	2.9	9.6	6.3
SA	24.6	3.3	4.1	22.5	16.5	4.2	5.4	4.9

Table 6.9: Comparison of the heuristics for L=24" with lumber distribution no. 2.

Demand schedule	4A	4B	5A	5B	6A	6B	7A	7B
Optimal solution	-	-	-	-	-	-	-	-
Best sol. values within 1 min								
CLIP/S	122	83	168	116	151	108	89	155
GA	122	82	168	116	154	107	91	162
SA	-	-	-	-	-	-	-	-
Best sol. values within 3 min								
CLIP/S	122	82	168	116	151	108	87	155
GA	122	81	168	116	154	107	90	158
SA	-	-	-	-	-	-	-	-
Best sol. values within 5 min								
CLIP/S	122	81	168	116	151	108	87	152
GA	122	81	167	116	152	107	90	158
SA	-	-	-	-	-	-	-	-
Best sol. values within 10 min								
CLIP/S	122	81	166	116	150	108	87	152
GA	122	81	167	116	151	107	90	158
SA	-	82	-	-	151	-	-	-
YP of best sol. within 10 min								
(Yield Limit)	91.3	91.2	91.4	91.6	92.1	91.7	92.3	92.0
CLIP/S	89.6	90.4	89.1	89.6	91.6	90.8	91.7	91.5
GA	89.6	90.4	89.2	89.6	91.6	90.9	91.6	91.2
SA	-	90.4	-	-	91.6	-	-	-
OP of best sol. within 10 min								
CLIP/S	23.8	0.2	1.3	17.4	5.9	1.5	4.4	4.6
GA	23.8	0.2	2.5	17.4	5.9	2.3	5.5	5.4
SA	-	0.8	-	17.4	5.9	-	-	-

Table 6.10: Comparison of the heuristics for L=36" with lumber distribution no. 2.

Demand schedule		4A	4B	5A	5B	6A	6B	7A	7B
GRADS	No. of arbors	2	4	5	3	5	6	6	7
	Total waste	151	90	160	133	200	116	93	170
	Overage	24%	0%	0%	25%	5%	0%	0%	0%
CLIP/S	Total waste	151	92	166	136	205	122	102	175
	Overage	29%	3%	3%	28%	26%	5%	9%	9%

Table 6.11: Comparison of CLIP/S and GRADS for L=24" with incoming lumber distribution no. 1.

Demand schedule		4A	4B	5A	5B	6A	6B	7A	7B
GRADS	No. of arbors	3	4	5	5	5	6	7	7
	Total waste	151	88	160	133	183	110	88	154
	Overage	10%	0%	0%	25%	9%	0%	0%	0%
CLIP/S	Total waste	151	88	162	135	190	112	91	156
	Overage	29%	1%	1%	28%	18%	4%	3%	6%

Table 6.12: Comparison of CLIP/S and GRADS for L=36" with incoming lumber distribution no. 1.

Demand schedule		4A	4B	5A	5B	6A	6B	7A	7B
GRADS	No. of arbors	2	4	5	3	5	6	7	7
	Total waste	123	84	165	117	182	113	96	165
	Overage	22%	0%	0%	16%	7%	0%	0%	0%
CLIP/S	Total waste	124	86	168	119	179	115	96	171
	Overage	25%	3%	3%	23%	17%	3%	5%	6%

Table 6.13: Comparison of CLIP/S and GRADS for L=24" with incoming lumber distribution no. 2.

Demand schedule		4A	4B	5A	5B	6A	6B	7A	7B
GRADS	No. of arbors	2	4	5	3	4	6	7	7
	Total waste	122	79	165	115	155	107	86	155
	Overage	23%	0%	0%	17%	1%	0%	0%	0%
CLIP/S	Total waste	122	83	168	116	151	108	89	155
	Overage	24%	0%	1%	17%	6%	2%	5%	5%

Table 6.14: Comparison of CLIP/S and GRADS for L=36" with incoming lumber distribution no. 2.

Chapter 7

Summary, Conclusions and Future Research

7.1 Summary of the Research

In this research, our major objectives were (1) to investigate the computational complexity of $GRSADSP/1$, (2) to develop a total enumeration procedure for solving $GRSADSP/1$, (2) to apply various heuristic methods such as local improvement procedures ($LIPs$), simulated annealing (SA) and genetic algorithms (GAs) to $GRSADSP/1$, and (3) to experiment with various data sets to compare the efficiency of these heuristics.

The summary of the research with respect to the above objectives is as follows:

First, we showed that verifying whether the yield limit (i.e., the theoretical upper bound on the yield percentage) in $GRSADSP/1$ can actually be achieved for a given shaft length and blade width regardless of the demand schedule is NP-complete.

Then we described a total enumeration procedure for solving $GRSADSP/1$ and discussed the complexity of the procedure.

We developed four heuristic methods based on local improvement procedures, and decided to choose $CLIP/S$ as our local search method, since it performed better than the other three $LIPs$ for the data sets we studied.

Next, we developed two other heuristics for the problem based on simulated annealing and genetic algorithms.

Then we applied $CLIP/S$, GA and SA to specific instances of the problem, which consist of several instances that have actually been encountered in practice and other randomly generated instances.

We compared the solutions obtained by the three heuristics to each other where the running times of the heuristics were equalized for four different time periods, namely, 1, 3, 5, and 10 minutes. We also analyzed the quality of the solutions obtained over these time periods. We compared the solutions obtained by the heuristics to the optimal solutions of the data sets for which we were able to run the total enumeration procedure within a reasonable time. For the data sets for which we could not obtain the optimal solution, we attempted to determine the solutions' quality by comparing the yield and overage percentages of the solutions to those of the theoretical lower bounds.

7.2 Conclusions

Our conclusions with regard to the above work are:

1. $CLIP/S$ is a very fast and efficient heuristic for $GRSADSP/1$. Excellent solutions can be obtained by multiple runs of $CLIP/S$ even within very short execution times.

2. It is not necessary to run $CLIP/S$ for a long period of time, since the quality improvement of the best solution obtained after a certain amount of time is not worth the extra time spent.

3. GA is also an efficient alternative to $CLIP/S$, but the results obtained with $CLIP/S$ are slightly better than those obtained by GA.

4. Although SA also finds high quality solutions, its computational requirements are beyond acceptable in some cases; leaving it out of consideration as a general

purpose, practical heuristic.

7.3 Future Research

In this research, we focused on solving $GRSADSP$ using only one arbor. However, it is quite plausible that better results can be obtained using more than one arbor (especially two arbors) despite the time loss due to arbor exchange. Furthermore, for some demand schedules encountered in practice, the shaft length of the saw used in the cutting job is simply not long enough to put at least one cutting channel of each finished width in the schedule, making the employment of at least two arbors unavoidable.

The heuristics that we have developed for $GRSADSP/1$ can actually serve as a starting point for developing an efficient heuristic method for solving $GRSADSP/2$ in the following manner: the original demand schedule can be partitioned into two new demand schedules in some way, where each schedule can then be treated as a separate cutting job. The heuristics implemented for $GRSADSP/1$ can *then* be used to determine an arbor to be used in each of these cutting jobs.

Once a solution method is developed for $GRSADSP/2$, both $GRSADSP/1$ and $GRSADSP/2$ can be solved to obtain a solution for the data sets for which either number of arbors is feasible. These solutions can then be compared to determine which solution would actually be a better choice.

Bibliography

[1] Aartz E., Korst J. (1989) "*Simulated Annealing and Boltzmann Machines*", John Wiley & Sons Inc., New York.

[2] Collins N. E., Eglese R. W. and Golden B. L. (1988) "*Simulated annealing - an Annotated Bibliography*", AJMMS, Vol. 8, pp. 209-307.

[3] Fathi, Y., Kegler S. R. and Culbreth C. T. (1996) "*A Column Generation Procedure for Gang-rip Arbor Design and Scheduling*", International Journal of Production Research, Vol. 34, No. 2, pp. 313-327.

[4] Garey M. R., Johnson D. S. (1979) "*Computers and Intractability: A Guide to the Theory of NP-Completeness*", W. H. Freeman, San Francisco.

[5] Holland J. H. (1975) "*Adaptation in Natural and Artificial Systems*", University of Michigan Press, Ann Arbor, Michigan.

[6] Johnson D. S., Aragon C. R., McGeoch L. A. and Schevon C. (1989) "*Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning*", Journal of Operations Research, Vol. 37, No. 6, pp. 865-892.

[7] Joines J. A. (1996) "*Hybrid Genetic Search for Manufacturing Cell Design*", Ph.D. Thesis, North Carolina State University, Raleigh, North Carolina.

[8] Kirkpatrick S., Gellat C. D. and Vecchi M. P. (1983) "*Optimization by Simulated Annealing*", Science, Vol. 220, pp. 671-680.

[9] Kou, L. T. (1977) "*Polynomial Complete Consecutive Information Retrieval Problems*", SIAM Journal of Computing, Vol. 6, pp. 67-75.

[10] Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., and Teller E. (1953) "*Equation of State Calculation by fast Computing Machines*", Journal of Chemical Physics, Vol. 21, pp. 1087-1091.

[11] Michalewicz Z. (1992) "*Genetic Algorithms +Data Structures = Evolution Programs*", 3rd Edition, Springer-Verlag, New York.

[12] Reeves, C. R. (Ed.) (1993) "*Modern Heuristic Techniques for Combinatorial Problems*", John Wiley & Sons Inc., New York.

[13] Steiglitz K., Papadimitriou C. D. (1982) "*Combinatorial Optimization: Algorithms and Complexity*", Prentice Hall, Upper Saddle River, New Jersey.