IMPROVEMENTS OF THE DYNAMIC PROGRAMMING ALGORITHM FOR TREE BUCKING

François Grondin

Mathematician
Forintek Canada Corp.
319 rue Franquet
Québec City
Québec, Canada G1P 4R4

(Received June 1996)

ABSTRACT

Log bucking is one of the most important operations in the transformation of trees into lumber. A bad decision at this stage can jeopardize the optimal recovery in volume or in value. The problem of optimizing the recovery during the bucking process has been solved using, among other things, dynamic programming. This article describes the main approaches and suggests some improvements to the dynamic programming approach. By introducing certain assumptions into the dynamic programming algorithm formulation, this approach becomes both more realistic and more efficient. The algorithm defined here is used in an integrated bucking-breakdown model. Example simulations demonstrate the computational speed improvements that result from the introduction of the assumptions.

Keywords: Optimized bucking, sawmilling simulation, dynamic programming.

INTRODUCTION

Log bucking is the operation that consists of cutting trees or stems into smaller logs of predefined lengths. This operation is necessary to transform a tree into valuable lumber. Because log bucking is one of the first operations of a sawmill, the decisions made at this stage of the transformation process have profound influence on the recovery performance of the mill. The search for a better way to crosscut stems allows us to increase this recovery.

The problem of subdividing a tree into smaller logs has been solved using well-known operational research techniques. One of these techniques, dynamic programming, has been used by many researchers during the last thirty years. However, program developers have made simplifying assumptions in order to narrow the search field so that computational efficiency could be achieved.

In this article, the first log bucking models will be presented. Emphasis will be placed on the major difficulties that occur in each case. Then we will suggest some basic assumptions that will be used in the mathematical formulation of this problem. A modified dynamic programming algorithm and computer time comparisons for some examples will complete this article.

LOG BUCKING MODELS

Models for bucking optimization only

One of the first models involving dynamic programming was defined by Pnevmaticos and Mann (1972). The goal of this model was to maximize the value of a stem by evaluating the number of logs to cut, their length, their diameters, and the location of the logs along the stem. The constraints set by Pnevmaticos and Mann for their model were:

a) The total length of the logs must be equal to or less than the initial stem length.

b) The diameter of any log must be within the limits of the diameters of the remaining stem.

c) Both log length and diameter must be within the limits specified by management.

This model uses some simplifying assumptions. First, the stem is defined using a truncated cone. Instead, they could have incorporated real taper formulas for the stem. Then,

Wood and Fiber Science, 30(1), 1998, pp. 91–104

the step length used for dynamic programming is the shortest log length that could be produced; this means that all other lengths have to be a multiple of this shortest length.

The major shortcoming of this model appears in the evaluation of log quality, which is based on probabilities, and in the estimation of log value, which incorporates only the length as a parameter. This model does not consider that a log should have overlength that is removed during the trimming operation.

Briggs (1977) improves Pnevmaticos and Mann's model. In this model, the step length is defined as the greatest common divisor of every length that could be produced, which offers the possibility to take the trimming overlength into account. The model does not evaluate log quality using probabilities, but instead uses specifications given by sawmillers for maximum and minimum diameters for each log class. Log value estimation is calculated based on the volume for each log dimension using Smalian's formula.

In his model, Briggs assumes that the stem is rectilinear. He does not consider defects like curves in his representation. Also, Briggs' model optimizes only log or lumber volume instead of lumber and subproducts value. If residual or alternate products are highly valued, this model will not prove sufficient.

Briggs (1980) improves his 1977 model to mitigate these problems. His approach is to create a general dynamic programming formulation, flexible enough to include any combination of log specifications and constraints that employs realistic stem quality features. Log value is evaluated by estimating the value of the lumber produced. This improves his 1977 model because, in that model, two logs with the same volume but different shapes were considered identical. In this model, Briggs analyzes the impact of defects like sweep and crook on the optimal bucking solution.

Estimation procedures used by Briggs (1980) are imprecise. To evaluate log volume, Briggs uses log scaling techniques that are quite inaccurate. A mean price is used for the lumber—for a given grade a 2 in. × 4 in. and a 2 in. × 6 in. are worth the same. A correction factor is applied to reduce the optimal solution value, to account for some defects like curve and crook. For example, the impact of curve is reduced using the following factor:

$$\text{Curve Reducing Factor} = \frac{\text{Maximal Deviation} - 2}{\text{Small End Diameter}}. \quad (1)$$

Please note, this factor does not consider log length as a parameter. For the same maximal deviation, the impact of the curve on recovery is usually greater for shorter logs than for longer logs.

Eaton (1977), McPhalen (1978), and Geerts (1979) developed similar models. Bobrowski (1994) employs another operational research technique, branch-and-bound, for log bucking optimization, and compares accuracy and computer time between this model and another model using dynamic programming. All those models are similar in one aspect—they consider the bucking operation independently of the log breakdown process.

Integrated models including bucking and log breakdown optimization

An integrated model including bucking and log breakdown optimization is built in a way such that the bucking operation must consider the result of the log breakdown simulation. Knowing the characteristics of a given log, it could be worthwhile to evaluate lumber recovery using those characteristics. Faaland and Briggs (1984) developed a model in which bucking and log breakdown process are closely related.

In their model, the stem is divided into segments (steps for the dynamic programming algorithm). Those segments are used to find the best way to buck the stem in order to maximize log value. This value is calculated by another dynamic programming algorithm that optimizes lumber value for the maximum cylinder included inside the log.

The evaluation of the maximum lumber val-

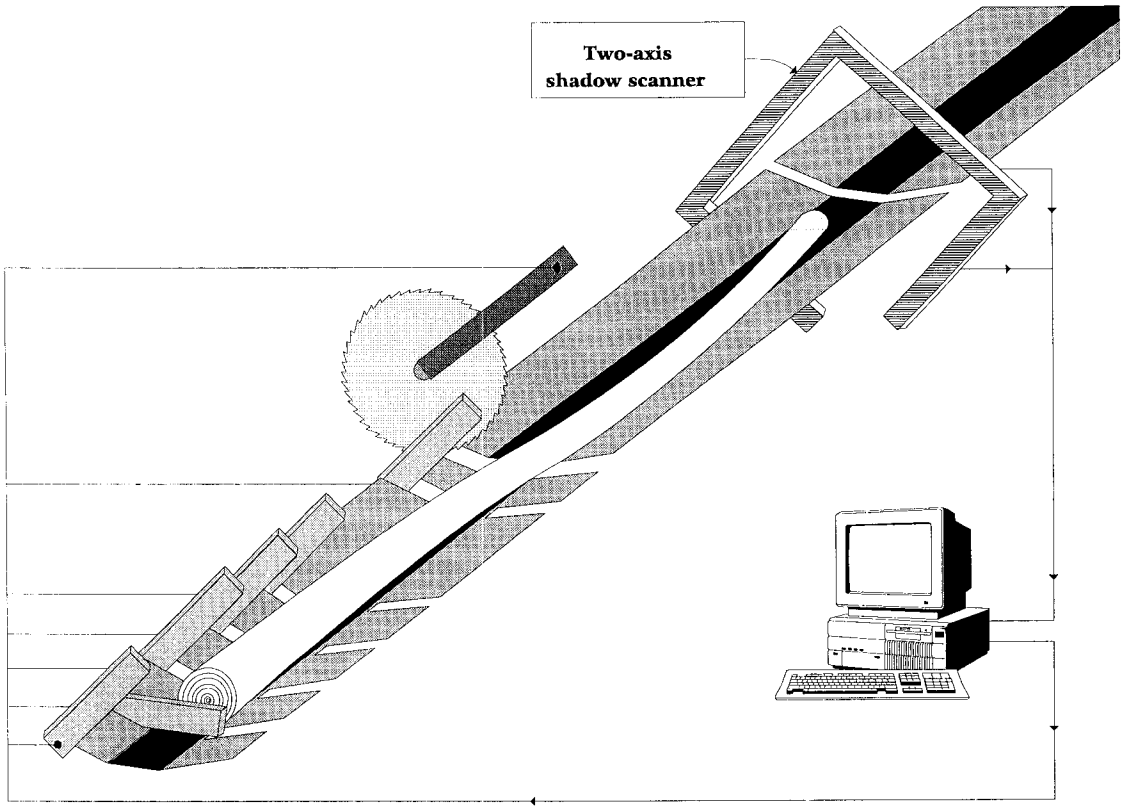

FIG. 1. Longitudinal bucking system with scanner and drop gates for optimized bucking on sawmill log deck.

ue does not consider that we could produce pieces shorter than the log length. Furthermore, using a cylinder to estimate optimal lumber value recovery may not be accurate.

Maness (1989) created a model in which bucking is optimized by dynamic programming. In his search procedure, the step was fixed at 24 inches plus one inch of overlength to reduce calculation time. His model uses a truncated cone to represent the stem, and the wane rules for the different grades are not used during calculations.

THE 1996 FORINTEK LOG BUCKING MODEL

In this section, the bucking model using dynamic programming developed by Forintek Canada Corp. (Mongeau and Grondin 1992 and Grondin and Drouin 1995) will be introduced. Bucking and log breakdown process are integrated: each log created during the bucking process is transformed by sawmill simulation. Stems and logs are modeled using elliptical cross-section representation.

The bucking problem we will define here is different from the one defined by Pnevmaticos and Mann (1972). The approach is closer to the one presented by Légaré (1994). This approach was chosen to represent longitudinal and transverse optimized bucking systems (Figs. 1 and 2). We suggest, based on sawmill observations, the following three assumptions:

1) *A bucking system, optimized or not, generally produces logs with fixed lengths, except for the small end of the stem where the last (topmost) log may be a different length.*

2) *A mill can convert shorter logs if they are at least as long as a given minimum length. Short logs usually come from the small end of the stem.*

3) *For an optimized bucking system, we*

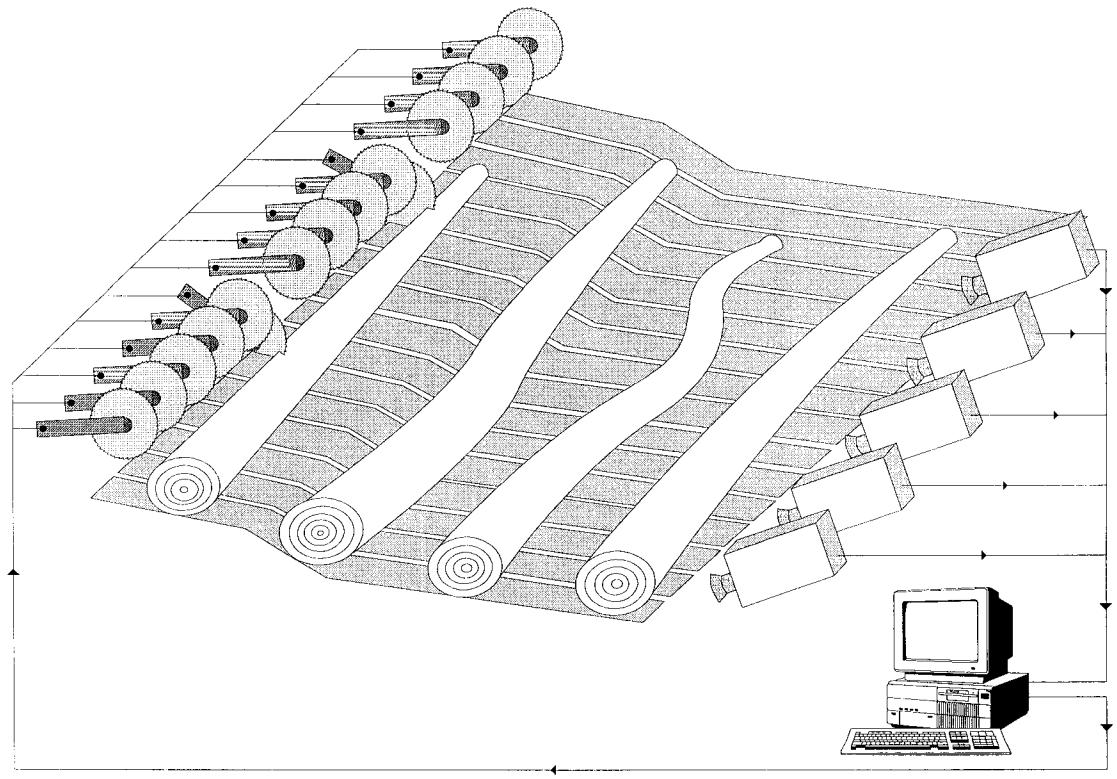

FIG. 2. Transversal bucking system with scanners and drop saws for optimized bucking on sawmill log deck.

must distinguish between the (search) step length and the shortest length that could be removed. The step length may be shorter than the shortest length.

For example, consider the bucking possibilities for a 31-foot stem (Fig. 3). The bucking system is designed to produce logs 8, 10, 12, and 16 feet long. The operator may decide to recover three 10-foot logs or to recover only two 10-foot logs and one 11-foot log. Or the operator may choose to keep a 16-foot log and a 15-foot log if the last one has only a few defects.

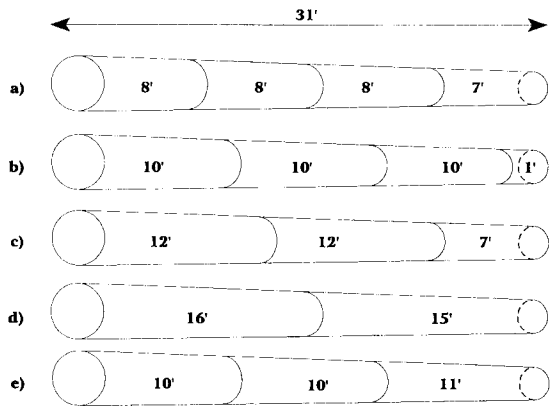

FIG. 3. Several alternative bucking possibilities for a 31-foot stem.

A bucking system should be flexible enough to handle variable log lengths at the small end of the stem that usually do not reach a predefined length. According to Fig. 3, if the mill has a minimum length specification of 8 feet, alternatives (a) and (c) are not viable. But, if the same mill admits logs 6 feet long or more, (a) and (c) should be considered as possible solutions. Of course, the optimal solution is closely related to the shape of the stem, to the lumber that could be produced, and to the optimization criterion (volume or value).

The last assumption refers to mechanical

limits of any optimized bucking system. Why should we try to estimate the stem each inch when the shortest length that is cut to remove defects is 24 inches? For a longitudinal bucking system (Fig. 1) where gates drop to a coming stem, we would not expect to have a gate placed 1 inch from the saw. We would also not expect to see two saws 1 inch apart in a transverse bucking system (Fig. 2).

This does not mean that we cannot choose a 1-inch step. Usually the step length is the greatest common divisor of the bucking length or any other length that is a divisor of those lengths. For example, if the bucking lengths are 99, 124, 149, and 200 inches (8, 10, 12, and 16 feet plus an overlength), the greatest step length is 1 inch. If we choose a 1-inch step, we would evaluate stem value for predefined lengths including a shortest bucking length used to remove defects rather than for each step.

MATHEMATICAL FORMULATION

Assumptions (1), (2) and (3) are the roots of the dynamic programming algorithm described in this article. Using these assumptions, we can formulate the recursive mathematical equations for the dynamic program. First, we must define our variables and our functions:

L = length of the stem
p = step size used in the dynamic program
$\log_{\min}$ = minimum log length that can be processed in the mill
l_0 = shortest bucking length; this is the shortest length that can be produced by the bucking system
B = $\{l_0, l_1, \ldots, l_N\}$ = set of all predefined bucking lengths; those lengths are listed from the shortest length l_0 to the longest l_N and must be divisible by p; they correspond to the log lengths the bucking station can produce
j = value of the current stage of dynamic program (in number of steps)

Now, let a and b ($a < b$) be two integer values describing the starting and ending positions on the stem of the log segment produced (in number of steps). Then:

chips(a, b) = chips value of the log segment with length $(b - a)p \le l_N$ starting at ap;
lumber(a, b) = breakdown (lumber + subproducts) value of the log segment with length $(b - a)p \le l_N$ starting at ap;
log_value(a, b) = value of a log segment with length $(b - a)p \le l_N$ starting at ap generated by the bucking system. This value, evaluated using log breakdown simulation, is defined as:

$$\textbf{log\_value}(a, b) = \begin{cases} \textbf{chips}(a, b), \\ \quad \text{if } b - a < \dfrac{\log_{\min}}{p}; \\ \quad \text{(if a segment is shorter than minimum length that can be processed in mill)} \\ \max[\textbf{lumber}(a, b), \textbf{chips}(a, b)], \\ \quad \text{if } b - a \ge \dfrac{\log_{\min}}{p}. \\ \quad \text{(if a segment is equal to or longer than minimum feasible length)} \end{cases} \tag{2}$$

We also define:

stem_value(a, b) = value of any segment of the stem with length $(b - a)p \le l_N$ starting at ap. The position $a = 0$ corresponds to the big end of the stem.

According to the assumptions (1), (2) and (3), and using the definitions of **log_value**(a, b) and **stem_value**(a, b), we can build the fol-

lowing recursive dynamic programming algorithm for the stem value:

$$\textbf{stem\_value}(0, j) = \begin{cases} 0 \\ \quad \text{if } j = 0; \\ \quad \text{(initial condition)} \\ -\infty \\ \quad \text{if } 0 < j < \dfrac{l_0}{p}; \\ \quad \text{(while shortest bucking length is not reached)} \\ \max\limits_{i \in I(j)} \{\textbf{stem\_value}(0, j - i) + \textbf{log\_value}(j - i, j)\} \\ \quad \text{if } \dfrac{l_0}{p} \le j < \left\lceil \dfrac{L}{p} \right\rceil; \\ \quad \text{(while end of stem is not reached)} \\ \max\limits_{k \in K(j)} \{\textbf{stem\_value}(0, j - k) + \textbf{log\_value}(j - k, j)\} \\ \quad \text{if } j = \left\lceil \dfrac{L}{p} \right\rceil; \\ \quad \text{(at the end of stem)} \end{cases} \tag{3}$$

where $I(j) = \{i \in \mathbf{N} \mid ip \in B \text{ and } i \le j\}$ is the set of predefined bucking lengths (in number of steps) shorter than jp, $K(j) = \{k \in \mathbf{N} \mid 1 \le k \le \min(l_N/p, j)\}$ is the set of available lengths (in number of steps) shorter than the minimum between the greatest bucking length and the length of the stem, and $[x]$ is the smallest integer greater than or equal to x.

Assumptions (1) and (2) occur during the evaluation of **log_value**(a, b). In fact, as long as dynamic programming does not reach the small end of the stem, the only lengths that can be produced are $l_0, l_1, \ldots, l_N$. When the small end is reached, the length of the log segment can take any value between 0 and l_N. If the log segment length is shorter than $\log_{min}$, the segment is sent to a chipper and its value is based on chip revenues. Of course, if length l_0 is shorter than $\log_{min}$, all segments with length l_0 are automatically converted into chips. If l_0 is greater than $\log_{min}$, every segment will be processed. For each segment, the algorithm will compare breakdown value and chip value to determine which is higher for optimization.

The third assumption (3) occurs in the equation **stem_value**$(0, j) = -\infty$. Knowing that l_0 is the shortest length that could be removed from the stem, it is unnecessary to evaluate a segment that is not at least equal to l_0. This last assumption is the one that enables us to significantly increase calculation speed for reaching the optimal bucking solution.

PERFORMANCE COMPARISONS

To demonstrate the benefits of using this algorithm, we will compare calculation times needed to optimize the bucking process for different step sizes (p), different shortest bucking lengths (l_0), and different minimal log lengths ($\log_{min}$). Remember that this algorithm is employed in a model where bucking and log breakdown are integrated. To realize this performance test, a fictitious mill and a set of lumber products were defined. A short description of the mill is given in Table 1. Table 2 gives a list of all lumber products that can be produced. Lumber volume optimization was tested during the simulation. The stems used for this test came from northeastern Quebec. Log diameters and log centroid positions were collected at every 3 feet along the stem for each stem. Elliptical cross-sectional representations of the logs that consider sweep and crook were used. Cross-sections were added using interpolation at every 6 inches to increase log representation quality.

Forty-six different bucking optimization scenarios were simulated using a 486-DX 33Mhz IBM-compatible computer. Because of

TABLE 1. *Description of the fictitious mill used to demonstrate the tree bucking optimization model.*

Machine center	Position of the piece	Cutting
Optimized bucker	end of piece at origin	dynamic programming lengths: 195″, 147″, 123″, 99″ (test on l_0) 192″, 144″, 120″, 96″ (test on p) 195″, 147″, 123″, 99″ (test on $\log_{min}$)
Diameter scanner	—	sends logs with at least 3.937″ at small end to line 1
Line 1		
Twin band saw	sweep up, split taper	opening: 3.75″ sawkerf: 0.145″
Centered bulledger	left face down, split taper	number of saws: 3 opening: 1.69″ sawkerf: 0.145″
Line 2		
Chipper canter	sweep up, split taper	opening: 2.75″
Centered bulledger	preceding face down, split taper	number of saws: 2 opening: 1.69″ sawkerf: 0.145″
Horizontal resaw	preceding face down, split taper	openings: 1.69″, 0.875″ sawkerf: 0.145″
Edger	preceding position	openings: 5.75″, 3.75″ sawkerf: 0.145″
Trimmer	end of piece at origin	positions of the saws: 0″–192″, 0″–168″, 0″–144″, 0″–120″, 0″–96″, 0″–84″, 0″–72″, 0″–60″, 0″–48″

the long time needed for each simulation, 45 stems of different lengths were randomly chosen. Calculation times for the simulations varied between 3 and 30 hours, depending on the step size (p), length of the minimum bucking segment (l_0), and minimum log length to be processed ($\log_{min}$).

The simulation software used here (see Mongeau and Grondin 1992 and Grondin and Drouin 1995) compares each breakdown possibility and keeps the best. According to the number of breakdown possibilities, the simulation may become time-consuming. In this simulation software, cutting a log corresponds to finding the intersection between a cutting plane and the log geometrical representation. When a log is cut by one saw during the simulation, it generates two distinct pieces and sawdust. To simulate the whole sawmilling process, the software simulates each machine in the mill. The piece to cut is first positioned (for example, a log is placed "horns down" in front of a twin band saw). Then, the piece is cut and the resulting pieces are sent to the next machine. This process goes on until all pieces reach the sorting table or the chipper where they are evaluated. If there are many positioning or cutting possibilities for a machine, every possibility will be tried and the best one will be kept. Figure 4 gives an idea of the fictitious mill as defined in the software.

Our goal here is to show with a few examples the benefits of this improved dynamic programming algorithm. Lumber volume optimization is used in the simulation in order to avoid building a fictitious price table for the products and to have more chances to generate lumber instead of chips (**log_value**(a, b) = **lumber**(a, b) if $b - a \geq \log_{min}/p$).

TABLE 2. *Lumber sizes and wane rules used in example to demonstrate the tree bucking optimization model.*

Thickness (real/nominal)			Widths (real/nominal)	Lengths (real/nominal)
0.875″/1″			2.75″/3″	48″/4′
1.690″/2″			3.75″/4″	60″/5′
			5.75″/6″	72″/6′
	Wane rules			84″/7′
	plank			96″/8′
Grade	thickness (%)	width (%)	length (%)	108″/9′
0	50	50	100	120″/10′
1	75	75	100	132″/11′
	2 × 3 & 2 × 4 & 2 × 6			144″/12′
Grade	thickness (%)	width (%)	length (%)	156″/13′
0	25	25	100	168″/14′
1	33	50	100	180″/15′
2	50	50	100	192″/16′
3	75	25	100	
4	99	75	100	

Calculation time and recovery rate comparisons for a variable shortest bucking length (l_0)

In this section, we want to compare the impact of the shortest bucking length (l_0) on calculation time. We set the step value to 3 inches, the minimum length to 72 inches, and the predefined lengths to 195, 147, 123, and 99. Those lengths were chosen so that the step value is the greatest common divisor. We then compare calculation times for values of l_0 from 3 to 99 inches by 3-inch step. While $l_0 < 72$, any segment of length l_0 is automatically sent to the chipper. Otherwise, this segment would be considered as a log and would be processed into the mill. For this reason, the variation of l_0 may have a direct impact on the recovery rate. This necessitates that we also compare the lumber recovery and chips percentages for the simulations.

Figure 5 gives an idea about calculation time for the different values of l_0. Here, we compare to the case where $l_0 = 3$ inches, i.e., where every step is evaluated (time unit = 100). The curve on Fig. 5 represents calculation time needed to reach an optimal solution. We see that calculation times are significantly different from the case $l_0 = 3$ inches. Time decreases from 77.9% ($l_0 = 6$) to 10.2% ($l_0 = 99$). This result makes sense because the algorithm does not lose time to evaluate useless segments.

The bars on Fig. 5 represent the number of segments that were evaluated during simulations for each value of l_0. As the reader could notice, calculation time and this number of segments are closely related. The more segments a simulation compares, the greater the calculation time needed to reach an optimal solution. Some values of l_0 (l_0 = 12, 24, 33 . . .) seem to compare fewer segments than their neighbors. While the small end of the stem is not reached, the dynamic programming algorithm tries to find an integer combination of every bucking length at the current position. So, consider the following equation derived from this specific example:

$$l_0 a + 99b + 123c + 147d + 195e = \lambda,$$
$$0 \leq \lambda \leq L \quad \text{and} \quad \lambda \equiv 0 \bmod 3. \qquad (4)$$

In this equation, 99, 123, 147, and 195 are the bucking lengths, l_0 is the minimum bucking length, and λ is the current position on the stem between 0 and L. Because the step value is 3 inches, λ must be divisible by 3. The number of evaluated segments during the simulations depends on the existence of a positive

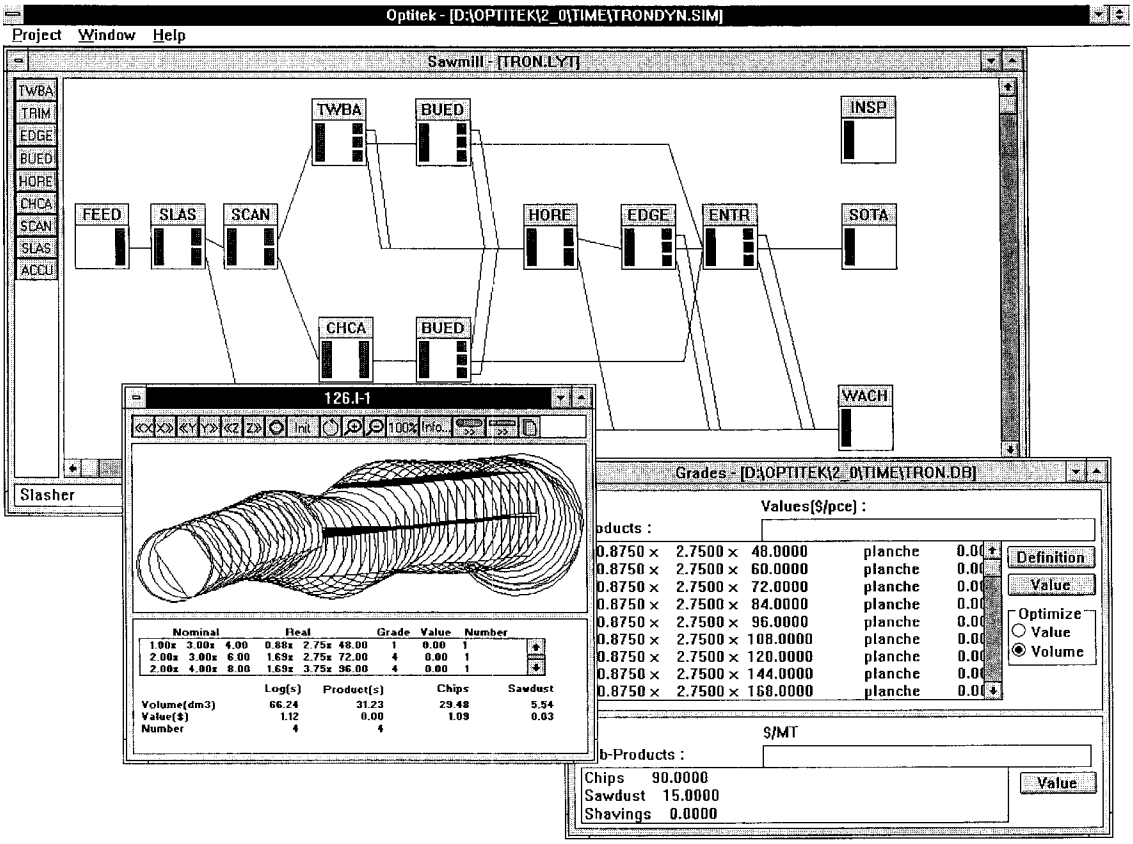

FIG. 4. Fictitious mill as defined in the simulation software.

integer solution to this equation, which is called a diophantine equation. The reader should consult Niven and Zuckerman (1980) for more details on the subject.

The values for l_0 and λ have a direct impact on the existence of any positive integer solution. It could be shown (Niven and Zuckerman 1980, p.135) that if $l_0 = 3$ inches, there always exists a positive integer solution. Things are different when $l_0 > 3$. For this example, it could be shown using the theory of diophantine equations that $l_0 = 24$ inches would generate fewer positive solutions for any λ than $l_0 = 21$ or $l_0 = 27$ inches.

We see in Fig. 6 that there is a 1.7% difference between the highest and lowest lumber recovery rates between $l_0 = 0$ and $l_0 = 69$ inches. Since any segment of length $l_0 \geq 72$ is processed as a log, the lumber recovery rate increases dramatically when $l_0 \geq 72$ inches. Recovery rate decreases between 0 and 69, 72 and 81, 84 and 93, and 96 and 99. In this example, the simulation may produce 48- to 96-inch lumber. When $l_0 = 84$, it becomes possible to produce 84-inch lumber (it was not possible when $l_0 = 81$). This explains the jump in recovery rate and the cyclic recovery pattern.

The same but reversed phenomenon occurred for the chips percentages. As noted, lumber recovery rates generally decrease and chips percentages generally increase for increasing values of l_0 between 0 and 69. Removing a longer segment and sending it to a chipper offers less wood to transform into lumber. Even if smaller values of l_0 give better results theoretically, higher values are more realistic due to mechanical constraints of the

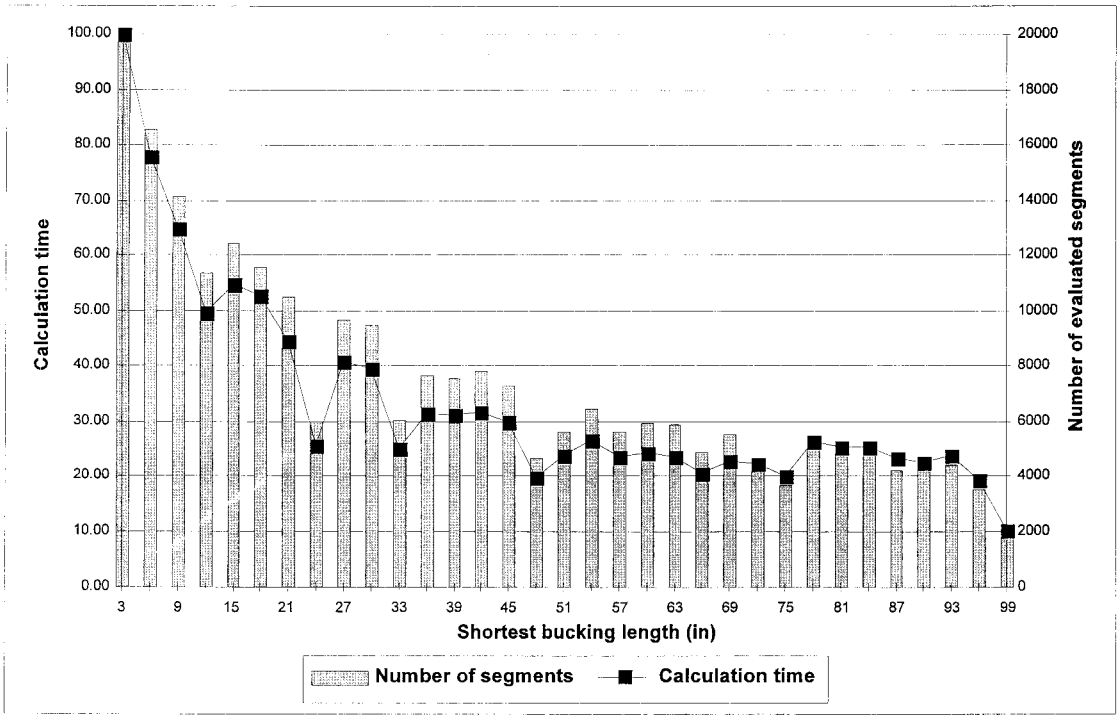

FIG. 5. Calculation time and number of evaluated segments vs shortest bucking length (l_0).

bucking system. Values of l_0 between 48 and 96 inches are unusual in sawmills. The only reason they were considered for simulation was to give the reader a complete view of the impact of l_0 on calculation time and recovery rates.

Calculation time comparisons for a variable step size (p)

The goal of this section is to show that the step size p does not significantly influence calculation speed of this improved dynamic programming algorithm when the other parameters are set. Here, we simulate optimized bucking where we set the minimum length to 72 inches, the shortest bucking length to l_0 = 24 inches. The set of predefined bucking lengths is modified: 192, 144, 120, and 96 inches are chosen to allow a larger step size. Because the step size must divide all predefined bucking lengths, the only possible integer values for the step are 1, 2, 3, 4, 6, 8, 12, and 24 inches. We compare calculation times for each case to the case where the step is 1 inch. Since the algorithm evaluates only those segment lengths included in list B (except for the small end of the stem where shorter segments may be evaluated), the optimal solution stays the same for all steps. Our objective here is to see if there is a significant calculation time difference between simulations using shorter steps with the same bucking lengths. According to Fig. 7, mean calculation time for step values lower than 24 inches is slightly higher. For this sample, the worst case (p = 8) and the best case (p = 24) show a 1% difference in computer processing time. These differences may be explained by the fact that dividing the stem into shorter segments will generate more segments to handle. Calculation time is relatively stable for p = 1 through p = 8 but decreases for p = 12 and p = 24 inches, where fewer segments are generated. The processing time differences for different step

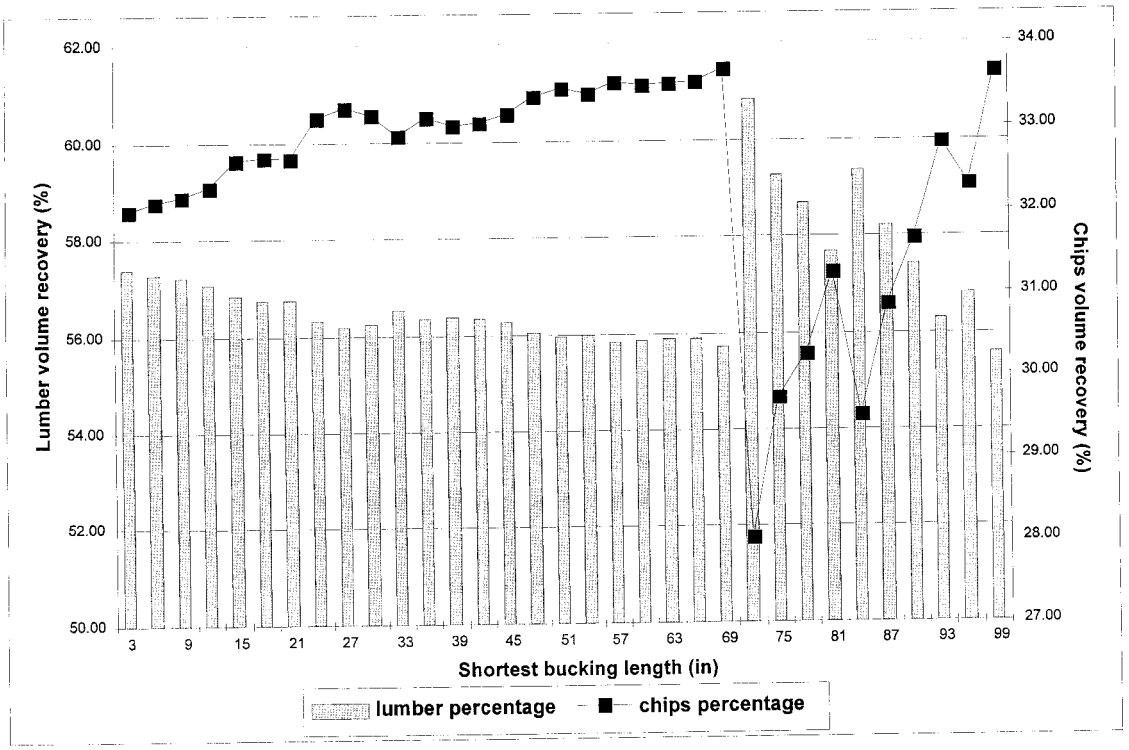

FIG. 6. Product volume recovery vs. shortest bucking length (l_0).

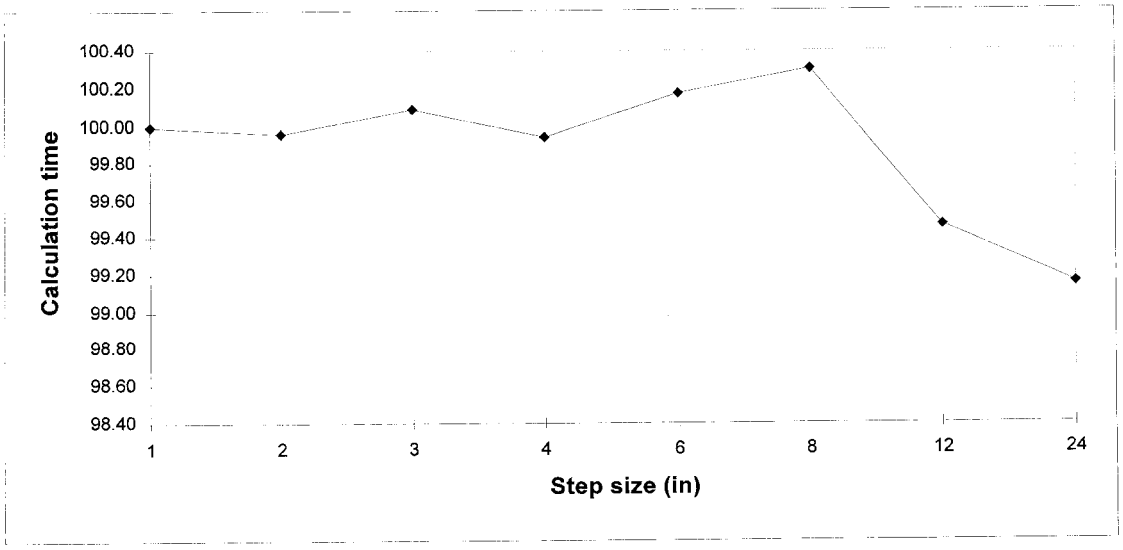

FIG. 7. Computational time increase vs. step size p when l_0 is set to 24 inches.

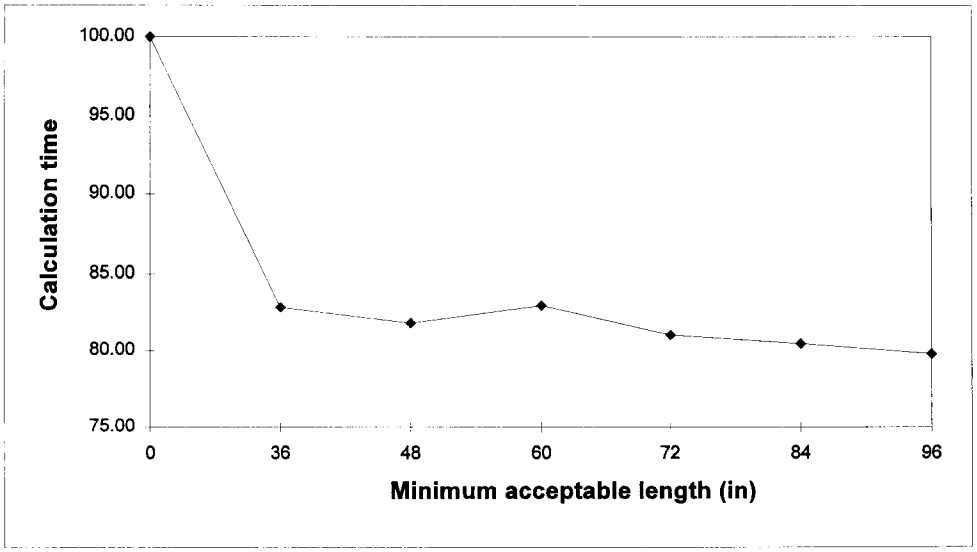

FIG. 8. Calculation time vs. minimum log length (log_{min}).

sizes are minimal so the failure to set p at the optimal level will not lead to gross inefficiencies.

Calculation time and recovery rates comparisons for a variable minimal log length (log_{min})

In this section, the influence of log_{min} on calculation time and recovery rates will be demonstrated. According to its definition, log_{min} is the length of the shortest log to be processed into the mill. This parameter mostly influences the decisions at the top of the stem, where the length of the resulting segment is not necessarily a standard bucking length, nor a multiple of the step size. Lowering log_{min} would increase the total length of the tree that can be processed. In that case, since the simulation saws every log with length higher than log_{min} and sends the others to a chipper, increasing the amount of logs would also increase calculation time. To demonstrate this fact, a simulation was run where log_{min} could take the values 0, 36, 48, 60, 72, 84, and 96 inches. The step value was set to 3 inches, and the bucking lengths were set to 195, 147, 123, 99, and 24 inches. Since the length of usable logs may change, we expect calculation times and recovery rates to be affected.

According to Fig. 8, there is a 15% decrease in calculation time between 0 and 36 inches. Between 36 and 96 inches, calculation time continues a slight downward trend. This result can be explained by the fact that for log_{min} = 0 inches, every segment is considered as a log and has to be processed into the mill. At log_{min} = 60 inches, there is a small increase due to the amount of possible combinations of logs that could be cut from the stem sample. Figure 9 demonstrates that lumber recovery rates decrease and chips recovery rates increase for increasing values of log_{min}.

CONCLUSIONS

In this article, we found assumptions that help us to improve calculation speed of a dynamic programming algorithm that is used during bucking operation. When we used this algorithm in an integrated bucking and log breakdown model, we realized noticeable improvements in computational speed when the minimum bucking length is increased.

As mentioned earlier, the main advantage of this algorithm is the possibility to simulate op-

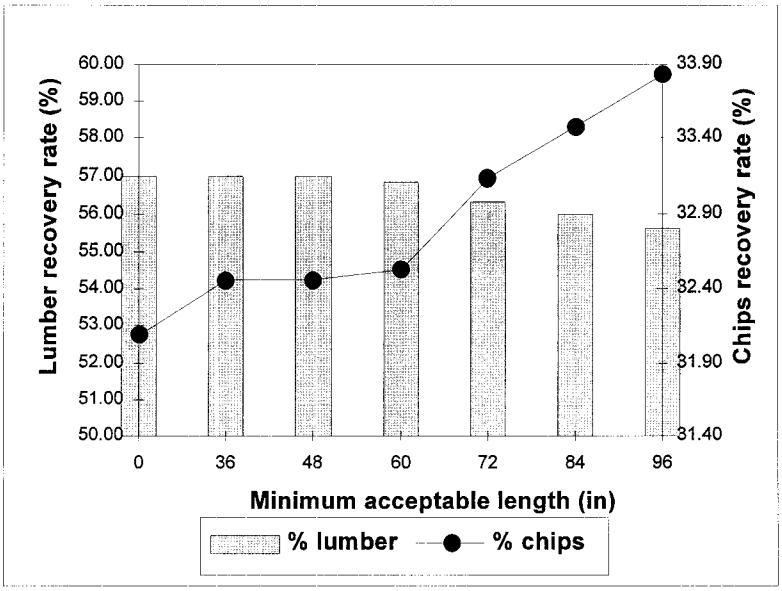

FIG. 9. Product volume recovery vs. minimum log length ($\log_{min}$).

timized bucking where the greatest common divisor is small compared with bucking lengths. For example, if our lengths are 99, 124, 149, and 200 inches, the greatest common divisor of these lengths is 1 inch. Current algorithms may spend a lot of time to evaluate useless segments. But if we add a shortest bucking length of l_0 = 24 inches and a minimum log length of 72 inches, calculation time decreases significantly (approximately 85% compared to the case where l_0 = 1 inch and the minimum length is 72 inches).

This algorithm can be adapted for other tasks. Optimized trimming, for example, may use a similar version of this algorithm. In future work, we can improve our algorithm by taking account of the sawdust produced during bucking and trimming operations. Also, instead of looking only at log length, we should see if we can include other parameters like diameters and sweep to determine if a log is to be rejected or not.

ACKNOWLEDGMENTS

The author wishes to thank Yves Lévesque, Nicol Drouin, and Frédéric Hébert for their support and their help, and Dr. Jan Wiedenbeck for valuable comments.

REFERENCES

BOBROWSKI, P. M. 1994. The effects of modelling on log bucking solution techniques. J. Opt. Res. Soc. 45(6): 624–634.

BRIGGS, D. G. 1997. A dynamic programming model for bucking tree stems into logs. Tropical Forests Utilization System. Contrib. No. 30 Institute of Forest Products, College of Forest Resources, University of Washington. Editions Kenneth J. Turnbull Center for International Studies. Seattle, WA. 12 pp.

———. 1980. A dynamic programming approach to optimize stem conversion. Ph.D. thesis, University of Washington, Seattle, WA. 393 pp.

EATON, N. J. 1977. LOGGON—A computer aid to optimize the cross-cutting of roundwood. CSIR Spec. Rep. HOUT 151. Council for Scientific and Industrial Research, Pretoria, South Africa.

FAALAND, B., AND D. G. BRIGGS. 1984. Log bucking and lumber manufacturing using dynamic programming. Mgmt. Science 30(2):245–257.

GEERTS, J. M. P. 1979. Optimal crosscutting of timber. Dept. of Forest Technique and Forest Products, Agricultural University Wageningen, The Netherlands. Pp. 218–229.

GRONDIN, F., AND N. DROUIN. 1995. Modèle de simulation du sciage. Tech. Rep. Proj. No. 3315K341, Forintek Canada Corp., Sainte-Foy, Qc.

LÉGARÉ, A. 1994. Modèle de tronconnage optimisé par ordinateur adapté aux bois de petites dimensions. Mémoire présenté pour l'obtention du grade de maître ès sciences. (M.Sc. thesis). Faculté des Études Supérieures, Université Laval, Québec, Canada. 165 pp.

MANESS, T. C. 1989. A technique for the combined optimization of log sawing and bucking strategies. Ph.D. thesis, University of Washington, Seattle, WA. 198 pp.

MCPHALEN, J. C. 1978. A method of evaluating bucking and sawing strategies for sawlogs. M.Sc. thesis, Faculty of Graduate Studies, University of British Columbia, Vancouver, BC. 81 pp.

MONGEAU, J. P., AND M. GRONDIN. 1992. Un nouveau modèle de simulation du débitage. Tech. Rep. Proj. No. 3343K388, Forintek Canada Corp., Sainte-Foy, Qc.

NIVEN, I., AND H. S. ZUCKERMAN. 1980. An introduction to the theory of numbers. 4th ed. John Wiley & Sons, New York, NY. 335 pp.

PNEVMATICOS, S. M., AND S. H. MANN. 1972. Dynamic programming in tree bucking. Forest Prod. J. 22(2):26–32.