A Progressive Heuristic Search Algorithm for the Cutting Stock Problem

Eva Onaindia, Federico Barber, Vicente Botti,
Carlos Carrascosa, Miguel A. Hernandez, Miguel Rebollo

Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia
Camino de Vera s/n 46071 Valencia (SPAIN)
email: onaindia@dsic.upv.es

Abstract. This paper presents a new variant of the $A^\star$ class algorithms for solving the known 'one-dimensional cutting stock problem' in a cardboard factory where the objective is to minimise the useless remaining of a continuous cardboard surface. The algorithm limits the number of nodes in the tree search by using a bound criterion based on the problem restrictions. The process of computing the solution is achieved at several stages, obtaining a complete non-optimal solution at each stage and improving the response as long as new stages are executed. The final stage returns the optimal solution. The proposed approach allows for a solution at anytime in the process resolution and also the refinement of the solution as more time is given to the algorithm. In this way, if a non-optimal solution is satisfactory enough for the factory, the process can be interrupted at that time. The computational performance of the algorithm indicates the effectiveness of the algorithm for solving practical one-dimensional cutting stock problems. Additionally, the experimental results show the important money savings achieved for the factory.

1 Introduction

The one-dimensional cutting stock problem consists in cutting a continuous sheet of raw material of a given width so as to satisfy a certain amount of demands of strips of different widths [4]. The objective is to minimise the amount of waste of the sheet. This problem can be found in many industries as steel construction, clothing, cardboard, metal sheets, paper or flat glass, etc.

Most of the formulations of the cutting stock problem assume very general or unrealistic conditions as no limitations in the number of guillotines or reducer chippers, no restrictions on the stock material orientation or the possibility of permitting trimming. The general way of solving this problem has been tackled as a linear programming problem thus leading to a computational difficulty due to the great number of combinations (columns) in the problem [4]. Other approaches applied to a similar problem, the two-dimensional cutting stock problem, use dynamic programming for solving unconstrained [1] or constrained [2] guillotine cutting problems.

The problem which we are concerned to falls into the category of the non-guillotine cutting problems as the optimal cutting patterns of rectangles are not restricted to those with the guillotine property. Solutions to non-guillotine problems as the bin packing or pallet loading problems mainly propose heuristic methods with worst-case and average-case performance bounds to solve various special cases of the problem [3] or integer programming formulations with tighter upper bounds to limit the size of the tree search [5].

In general, solutions for the cutting stock problem hardly use heuristic techniques and the few existing ones are not applicable to real problem environments. In this way, from a practical point of view, industries are more interested in a rapid but acceptable solution, in terms of wastage, to carry out its daily stock planning than obtaining long time consuming optimal solutions. It becomes then necessary a trade-off between the cost and the quality of the solution.

We show in this paper the advantages reported by the application of a heuristic method (a variant of the $A^\star$ algorithm) to solve a problem which has been traditionally solved by operation research methods. The advantages can be summarised in the following points:

- Achieving of the optimal solution.
- The algorithm has been designed as a progressive procedure which returns an answer each time a better solution is reached.
- The operator can interrupt the program when the wastage percentage achieved by the algorithm is satisfactory enough for the factory planning policy. This is a very important feature due to the high cost in obtaining the optimal solution in many cases.
- Appropriateness to solve a real problem in a cardboard industry. The application of the algorithm is currently reporting important money savings in the factory.

2 Problem Description

The problem consists in obtaining a set of demands $D = D_1, D_2, ..., D_n$ from a rectangular stock roll with no length end and a fixed width W. Each demand is viewed as a smaller stock piece which is cut up into a number of *sheets* each having a predetermined *orientation*, *length* and *width*. The total stock area covered by a demand is given by the following formulae:

$$Area(D_i) = S_i \times w_i \times l_i$$

where S_i represents the total number of sheets necessary to complete the demand and w_i and l_i represent the sheet width and length respectively.

The total area covered by a demand may result in a rectangular or non-rectangular stock piece. For example, in Fig. 1, the area for demand D_1 is a rectangular piece while the total surface covered by D_4 has a non-rectangular shape. This depends on the number of sheets, how many of them can be cut in parallel along the roll width and also on the stock cutting planning.

Demands are specified in terms of a weight quantity of cardboard (q_i), the size of the sheets ($w_i \times l_i$) and the cardboard quality. The quality determines the density (d) of the cardboard type (this measure is given in gr./m^2). Due to technological reasons, dimensions of the sheets can not be exchanged so it is necessary to keep the specified orientation ($w_i \times l_i$) in the roll cardboard. The daily set of demands to be processed are selected according to the delivering date of each demand.
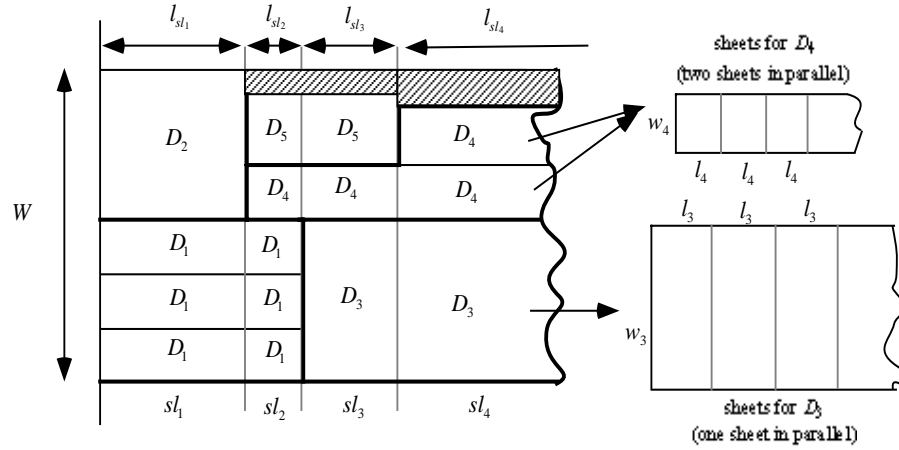

Fig. 1. Demands distribution over a cardboard bar

A slitter machine (Fig. 2) disposes of a series of longitudinal guillotines or reducer chippers (C) and several transversal guillotines or simply guillotines (R). The number (R) of guillotines determine the maximum number of demands which can be cut up simultaneously as each of the guillotines slit a piece of material which is sent to a different pallet. The number of reducer chippers (C) state the maximum number of sheets (for the same or different demands) to be cut in parallel along the roll width (Fig. 2) [1].

It is not necessary all the guillotines or reducer chippers work simultaneously. This will depend on the number of demands which are being processed in parallel and the number of sheets for each demand at one particular moment. Once one of the demands is completed, the number of guillotines and reducer chippers will be adjusted again to attain a new demand combination. This new combination must consider the remaining surface to cover from the pending demands.

The objective is to optimise the total cardboard surface, that is, to cover the flat surface with a number of sheets such that the less extension of the

---

[1] In our case two of the reducer chippers are used to eliminate a small trim along the bar and the rest are used to slit the sheets.
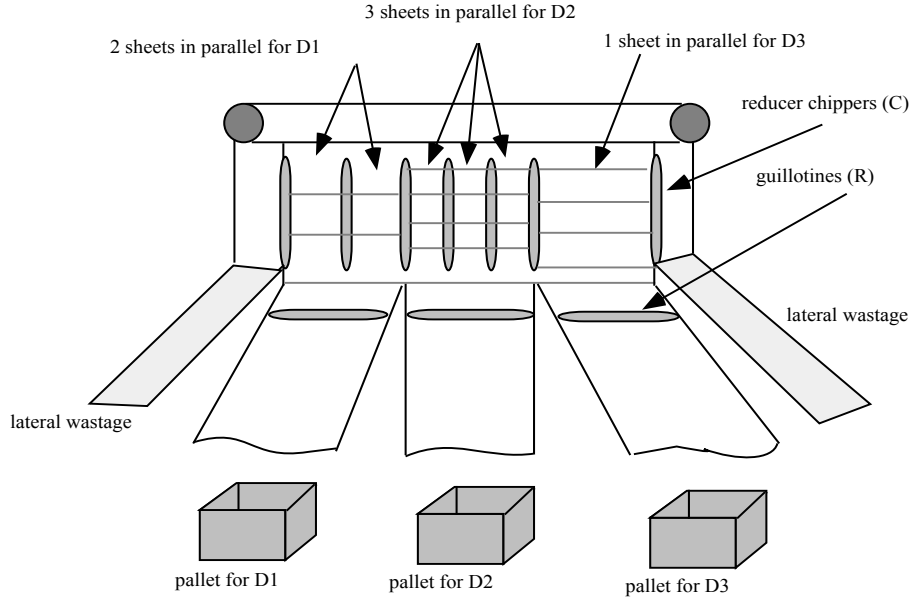
Fig. 2. A slitter machine with 3 guillotines and 7 reducer chippers

surface remains uncovered. We will only consider the lateral wastage along the bar length.

3   Problem Formulation

In this one-dimensional cutting stock problem we are only interested in the roll width (W). The underlying idea is to obtain the best combination of demands in such a way that this combination covers the maximum width of the bar as possible. This statement introduces the concept of *stock lines* (Fig. 1). A stock line (sl_i) is a set of sheets cut up in parallel each corresponding to one of the simultaneous demands being processed. There are two different approximations for a stock line:

a) to consider only one demand D_i in parallel a number of N_i times (each time corresponds to a sheet) along the width of the bar, such that $N_i \times w_i \leq W$.

b) to consider several demands $D_i, D_{i+1}, \ldots, D_j$ in parallel where each demand D_k is cut N_k times along the width of the bar and hence

$$\sum_{k=i}^{j} N_k \times w_k \leq W \quad .$$

The number of reducer chippers determine the maximum number of sheets which can be slit in parallel (either from one or several demands). In general, a slitter machine with C reducer chippers allow for $C - 1$ parallel sheets along the roll width such that $\sum_{i=1}^{C-1} w_i \leq W$.

Once the stock line is planned, this combination is maintained until one of the demands is finished up, that is, the pallet reaches the requested quantity of the demand. A line is then defined as the piece of stock material which covers a set of constant demands. The total surface covered by a specific stock line sl_j is given by $w_{sl_j} \times l_{sl_j}$ where w_{sl_j} is the stock line width and l_{sl_j} is the stock line length such that $w_{sl_j} \leq W$. The total amount of lateral wastage produced by the stock line is $(W - w_{sl_j}) \times l_{sl_j}$.

Let C be the maximum number of sheets to be cut in parallel in one stock line and N the number of demands to be served. For a demand D_i there exists a maximum number m_i of combinations in parallel where $m_i \leq C$. These combinations may generate the demand alone, twice in parallel (two sheets of the demand in a stock line), three times in parallel (three sheets of the demand in a stock line),..., or m_i times in parallel (depending on the demand size).

In this way, a maximum of $\sum_{i=1}^{N} m_i$ demand combinations may arise in a problem thus leading to a $O(Nm)$ cost where $m = max_i(m_i)$.

Let R be the maximum number of demands to be served in parallel where $R \leq C$. The total number of stock lines is calculated as the number of possible arrangements among all possible demand combinations into groups of R demands:

$$\binom{Nm}{R} = \frac{Nm!}{R! \times (Nm - R)!} = \frac{Nm \times (Nm - 1) \times \cdots \times (Nm - R + 1)}{R!}$$
$$= \frac{1}{R!} \prod_{i=0}^{R-1} (Nm - i) \leq \frac{1}{R!} \prod_{i=0}^{R-1} Nm \in O((Nm)^R) \ .$$

Temporal cost for the worst case is $O((Nm)^R)$. This complexity also represents the spatial cost. However, the latter one can be drastically reduced since many of the combinations can be removed for the following reasons:

- those combinations which exceed the bar width (W)
- those combinations where the same demands come up in different order. From a practical point of view, different lines as the ones in Fig. 3 are equivalent.

These and other considerations enable to apply a great reduction in the solution search space by cutting off those branches where this type of combination arise. On the other hand, this computed cost is for a general case. In the case of the factory where the application is running the value of R is two and m and C is six since the slitter machine disposes of two guillotines and seven reducer chippers.
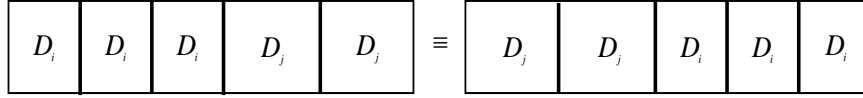
Fig. 3. Equivalent stock lines with demands distributed in a different order

4 Algorithm design

In this section we discuss the algorithm design in order to solve the problem optimally. The aim at using a $A^{\star}$ is twofold:

a) although an heuristic algorithm could report the same performance as an Operations Research method in the worst case, it is likely it would show a better performance in the average case as it always expands the most promising solution at hand, what could lead directly to the optimal solution.
b) the main objective is, however, to tackle the problem as a process which progressively refines an existing solution thus achieving better results as more time is given to the algorithm. This allows to stop the process when the obtained wastage percentage is satisfactory enough instead of waiting until the optimal solution is found.

The solution design uses an $A^{\star}$ algorithm combined with an upper bound in a tree search procedure where the bound is progressively updated as better solutions are derived.

Each node in the tree represents a combination of stock lines. Nodes already expanded represent the demands already delivered and nodes generated but not expanded (alive nodes) is a sequence of stock lines where not every demand has been completed. The procedure performs a best-first search iteratively by depth levels selecting the node with the least cost among all the nodes to be expanded in the deepest level. In that way, the list of alive nodes is ordered by depth levels and by priority at each level. This strategy allows to rapidly obtain a solution to bound the search space before expanding useless branches in the tree search.

Heuristic evaluation

The objective is to minimise the lateral wastage in the roll, that is to maximise the width of the bar or equivalently to minimise the length of the roll. This means that the longest width of the bar is used the shortest length of the roll is needed. In this way, the evaluation function $(f(n) = g(n) + h(n))$ is calculated in terms of the roll length where:

- $g(n)$: is the total cost from the initial state to reach node. This represents the roll length consumed by the demands delivered until node is reached.
- $h(n)$: is an estimation of the roll length needed to reach an optimal goal state.

Let $D_i, D_{i+1}, \ldots, D_j$ be the pending demands at node n. The total cardboard surface (TS) necessary to complete all the pending demands is given by:

$$TS = \frac{q_i}{d} + \cdots + \frac{q_j}{d} \ .$$

where q_i are the demands quantities and d is the cardboard density, and the shortest roll length (SL) necessary to complete the process is $SL = \frac{TS}{W}$. Let sl_m be the stock line with the least wastage, that is, the line which covers the maximum width in the roll (w_{sl_m}). Function evaluation is computed as:

$$h(n) = \frac{TS}{W - w_{sl_m}} \leq h^\star(n) \ .$$

whose value is surely less or equal than the real cost of the problem (the equality is given in case all the remaining stock lines are of width w_{sl_m}). With this function we can ensure the optimality of the found solution.

Upper bound proposal

To find the optimal solution is a NP-complete problem whose resolution has a factorial cost. In order to make the problem tractable, the process of computing the solution is divided in four different stages or levels:

- *First level solution*: an initial solution is calculated as the combination of those stock lines with only one demand in parallel along the roll width. This first rough solution is the one required when only one demand is to be planned.
- *Second level solution*: this approximate solution is computed as the combination of those stock lines with minimum wastage which allow to deliver all the demands. The total length of this solution is used as an upper bound for the next level.
- *Third level solution*: this stage generates a tree search by expanding the best k-successors from a given node. Function evaluation is applied at each node; in case the resulting value exceeds the current upper bound the branch is cut off. The upper bound is progressively updated as follows: associated with each frontier node m at the search horizon is a value $f(m) = g(m)$ since every node m is a goal state; upper bound is then updated as $UB = min_m(UB, g(m))$. This is why this level solution performs a best-first search by deepening levels thus permitting to reach goal states more rapidly. The final value of the upper bound obtained from this level is passed to the next one. This idea is similar to the one developed in algorithm $IDA^\star$ [6] or in real-time searching [7].
- *Fourth level solution*: this stage performs a similar search for the rest of successors of each node. The upper bound used for this stage is the one calculated in the previous one. This level solution guarantees the optimal solution.

In most of the test cases evaluated in the experimental results it was noticed that the optimal solution was found at the third level. In other cases, the solution from the third level was satisfactory enough for the factory so as it was not necessary to run the application to completion.

5 Application interface

The proposed algorithm has been implemented in Visual C++. The input data is the set of demands to be planned. These are selected from all those available in the stock of the factory. Demands are chosen according to the required cardboard type and the latest date at which the demand must be delivered to the client. The selected set of demands are showed in a window (see Fig. 4) and the user can then withdraw some particular demands, oblige to take into account other different set of demands or let the application the decision to plan this additional information. Another input parameter the user can enter is the maximum acceptable wastage percentage.

Fig. 4. Selected demands to be cut

While the process is being resolved the user can interact with the application through a window where it can be seen the new best computed solution at the

current moment and also the reaming time to finish up the calculation of the current solution level. Additionally, the user can interrupt the process at any time in case it is considered the solution in course is good enough.

Once the computation process is finished, either because the user has interrupted it or the application has stopped itself, a new window is returned with the information about the planned demands and the total wastage percentage obtained. It is also shown the computed solution in detail, either graphically (Fig. 5) or in a high detailed textual format.
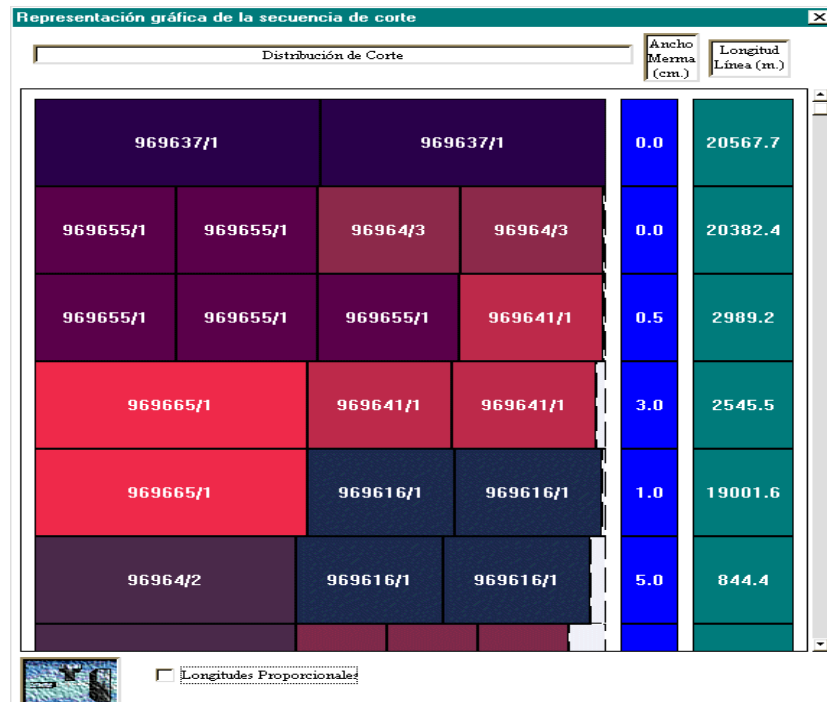

Fig. 5. Graphical solution

6 Experimental results

The computational tests for evaluating the performance of the algorithm are examples taken from a historical file of real stock demands in the factory.

Fig.6 shows the average wastage percentage obtained in the several proofs performed with different amount of demands according to its delivering date. Notice that, the total wastage percentage decreases in proportion as the number of demands. However, monotonicity is not preserved since it can not be ensured

a smaller wastage with one more demand though the tendency is to decrease in proportion as the number of demands.
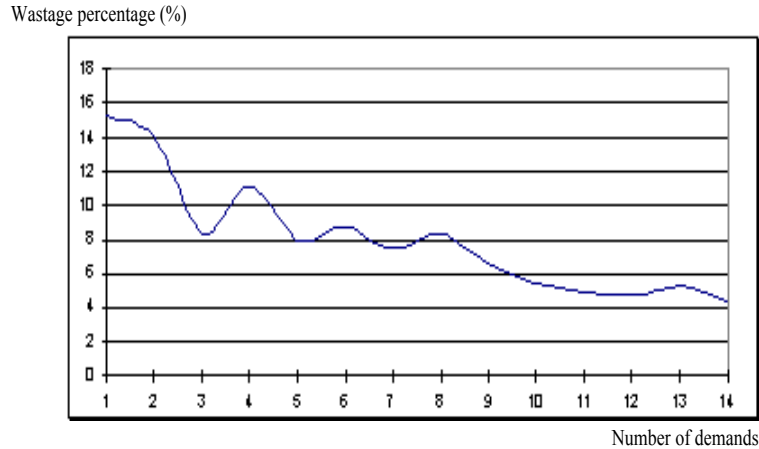
Wastage percentage (%)

Fig. 6. Wastage distribution proportionally to the number of demands

Finally, we show the mean deviation in the solution computed with limited time executions (20 sec., 40 sec., 60 sec., 90 sec., and 120 sec.) comparatively with the optimal solution. Figure 7 shows the deflection between the wastage obtained at each different period execution and the optimal wastage. It can be noticed the deviation decreases as time execution given to the algorithm increases. The problem is to ensure at which moment a better solution will be found.

An important feature of the algorithm is that the relation between the number of stock lines in the initial situation and the number of nodes to be generated in the solution can be computed at the beginning of the process. This provides a global estimation of the necessary time to solve the problem.

7   Conclusions

The main contribution of this paper is the design of a progressive $A^\star$ algorithm to deal with a real cutting stock problem and the significant results by means of a heuristic search algorithm. The main underlying idea is the combination of a heuristic evaluation function together with an upper bound threshold in order to cut off those branches in the tree search which do not lead to a better goal state. With this formulation, it is not only possible to obtain the optimal solution at the last stage in the algorithm but experimental results show that the solution attained at previous stages is in most of the cases as good as the optimal one.

Solution is progressively refined at each stage by updating the value of the upper bound. In this way, each time a new upper bound value is reached a
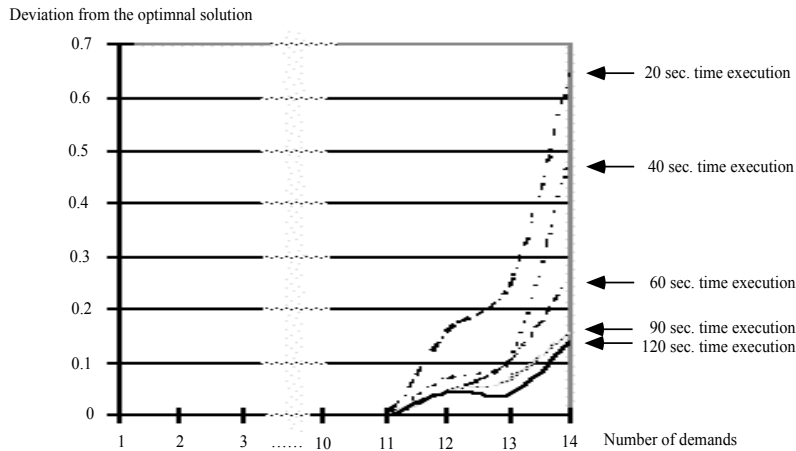
Deviation from the optimnal solution

20 sec. time execution

40 sec. time execution

60 sec. time execution

90 sec. time execution
120 sec. time execution

Number of demands

Fig. 7. Deviation from the optimal solution

new better solution is being obtained. This process allows to stop the algorithm process when the lateral wastage from the last obtained solution is satisfactory enough for the factory.

Future improvements include the design of new appropriate heuristic functions for the particular environment of the cardboard industry and more accurate estimations of the pending time execution to reach the optimal solution for a problem at a particular instant of the application execution.

References

1. Beasley J.E. "Algorithms for unconstrained two-dimensional guillotine cutting". Journal of the Operational Research Society, 36, pp. 297-306 (1985).
2. Christofides N., Hadjiconstantinou E. "An exact algorithm for orthogonal 2-D cutting stock problems using guillotine cuts". European Journal of Operational Research, Vol. 83, pp. 21-38 (1995).
3. Daniels J.J., Ghandforoush P. "An improved algorithm for the non-guillotine constrained cutting-stock problem". Journal of the Operational Research Society, 41, pp. 141-149 (1990)
4. Gilmore P.C., Gomory R.E. "Multistage Cutting Stock Problem of Two and More Dimensions". Operations Research, Vol. 3, pp. 94-120, (1965).
5. Hadjiconstantinou E., Christofides N. "An exact algorithm for general orthogonal, two dimensional knapsack problems". European Journals of Operational Research, Vol. 83, pp. 39-56 (1995).
6. Korf R.E. "Real-Time Heuristic Search". Artificial Intelligence Vol. 42, pp. 189-211 (1990).
7. Ishida T. "Real-Time Search for learning autonomous agents".Ed. Kluwer Academic Publishers. (1997).