11-1991

Efficient Algorithm for Nonpoint Source Pollution Control Problems

Aziz Bouzaher
Iowa State University

John B. Braden
University of Illinois at Urbana-Champaign

Gary V. Johnson
University of Manitoba

Susan E. Murley
University of Illinois at Urbana-Champaign

Efficient Algorithm for Nonpoint Source Pollution Control Problems

Abstract
A dynamic programming algorithm is proposed for a class of nonpoint source pollution control problems. The inherently combinatorial nature of these problems--stemming from the discrete nature of the decision variables, which are production and conservation practices--gives them a special knapsack structure with multiple right hand sides and additional multiple choice constraints.

This paper focuses on the computer implementation of this algorithm and its numerical testing and behavior compared with standard integer programming codes. The results show the robustness and relative efficiency of the approach.

Furthermore, this paper demonstrates that dynamic programming can be used to generate sensitivity analysis information for multiple choice knapsack problems.

Keywords
Dynamic programming, integer programming, microcomputers, environmental studies, agriculture

Disciplines
Agricultural and Resource Economics | Agricultural Economics | Economics | Natural Resource Economics

An Efficient Algorithm for Nonpoint Source Pollution Control Problems

Aziz Bouzaher,
John B. Braden, Gary V. Johnson, and Susan E. Murley

Working Paper 91-WP 78
Novernber 1991

Center for Agricultural and Rural Development
Iowa State University
Ames, Iowa 50011

Aziz Bouzaher is a visiting professor of economics, Iowa State University; John Braden is a professor of agricultural economics, University of Illinois, Urbana-Champaign; Gary Johnson is an associate professor of economics and farm management, University of Manitoba, Winnipeg; and Susan Murley is a former graduate assistant in economics, University of Illinois.

This research has been supported in part by grants from the Illinois Department of Energy and Natural Resources (No. WR-7) and the Illinois Agricultural Experiment Station (No. 05-334). Parts of the study were conducted at the University of Illinois. Valuable assistance was provided by Joeng Lim.

Abstract

A dynamic programming algorithm is proposed for a class of nonpoint source pollution control problems. The inherently combinatorial nature of these problems--stemming from the discrete nature of the decision variables, which are production and conservation practices--gives them a special knapsack structure with multiple right hand sides and additional multiple choice constraints.

This paper focuses on the computer implementation of this algorithm and its numerical testing and behavior compared with standard integer programming codes. The results show the robustness and relative efficiency of the approach.

Furthermore, this paper demonstrates that dynamic programming can be used to generate sensitivity analysis information for multiple choice knapsack problems.

Key Words: Dynamic programming, integer programming, microcomputers, environmental studies, agriculture.

AN EFFICIENT ALGORITHM FOR NONPOINT SOURCE POLLUTION CONTROL PROBLEMS

Introduction

In Bouzaher et al.[1] a model for efficient control of sediment pollution in surface waters was presented. The approach combines (i) a spatially dynamic hydrologic model that captures the transport process through which eroded soil (due to the weather, the geography of the land, and the management practices applied to it) becomes sediment, pollutes water systems, and thus affects water quality; and (ii) an economic optimization model that evaluates the management alternatives (including production and conservation practices) on each land management unit (LMU).

The model operates over a water basin or watershed area (also called a pollution sink) that is divided into hydrologically independent catchments, each of which is represented by a characteristic runoff path, termed *transect*, to give direction and convergence to sediment. Each LMU adds sediment in a nonlinear way (due to the delivery process through intervening LMUs) along only one transect; thus the set of transacts forms a complete partition of the set of LMUs. In addition, each transect contributes sediment additively to the pollution sink. Furthermore, for each LMU along a transect, only a finite number of nondominated management practices are considered (the process of determining these nondominated practices is described in Bouzaher et al.)[1,2] A *choice set* is defined as a combination of management practices, one from each LMU along a transect. Clearly, the number of these choice sets is combinatorial and it is required to find an *optimal set of choice sets* for any given level of sediment deposition allowed in the pollution sink.

The theoretical foundations and the economic implications of the model can be found in Braden et al.[3-5] The details of the modeling effort can be found in Bouzaher et al.[1,2] and Braden et

al.[6] Other areas of possible application of the model are discussed in Braden et al.[4,7]

The overall sediment control problem turns out to be a special class of 0-1 knapsack problem with additional multiple choice [or Generalized Upper Bounding (GUB) constraints]. However, what makes this class of problems even more special is the types of solutions required and the uses they are intended for. In particular, for policy evaluation, families of solutions, giving the trade-off costs (benefits) of tightening sediment depositions (say, from 50 to 1000 tons), are required to design decentralized control measures, and are summarized in a *sediment abatement cost frontier*, which is essentially a value function of the right hand side of the knapsack constraint (see Braden et al.[3,5]). Thus, the class of problem at hand is termed *multiple choice, multiple right hand side knapsack problems.*

This paper focuses on the computer implementation of the dynamic programming algorithm presented in Bouzaher et al.[1] and its computational testing compared to standard integer programming codes. The results show the robustness and relative efficiency of the algorithm. The approach is useful for (i) practitioners and managers of nonpoint source pollution control, (ii) other areas where solutions of knapsack problems with multiple right hand sides are required, and (iii) sensitivity or parametric analysis of multiple choice knapsack problems.

First, the mathematical model of the problem and its dynamic programming formulation are presented. The second section discusses microcomputer implementation. The first and second parts draw all the background material needed for the computational study of the algorithm from Bouzaher et al.[1]. The final section presents extensive computational results, both on real data and on random problems. Some computational complexity issues are addressed empirically.

The Mathematical Formulation

The Knapsack Model

The control problem, described previously in general terms, requires complete enumeration to find an optimal choice set for each transect, and thus an optimal management

practice for each LMU. It can be seen as a critical path problem with the addition of the sediment constraint. It is therefore formulated as a zero-one integer programming problem.

Recalling that a choice set was defined to be a set of management alternatives, one for each LMU in a transect, we let:

j = 1, ..., J the number of transects in the system;

p = 1, ..., P_j the number of feasible and nondominated choice sets in transect j;

r_{pj} = incremental cost, over baseline practices (corresponding to the unconstrained case), generated by the adoption of choice set p in transect j;

s_{pj} = amount (tons) of sediment delivered into the pollution sink by choice set p for transect j (These coefficients are computed using a sediment transport model described in Bouzaher et al.[1] and Braden et al.[5]);

$\bar{S}$ = maximum allowable sediment load into the sink; and

x_{pj} $\begin{cases} 1, \text{ if choice set p is selected from transect j} \\ 0, \text{ otherwise.} \end{cases}$

We then choose x_{pj} (j = 1, ...J; p = 1, ..., P_j) to

$$\text{minimize} \quad f = \sum_{j=1}^{J} \sum_{p=1}^{P_j} r_{pj} x_{pj}, \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{J} \sum_{p=1}^{P_j} s_{pj} x_{pj} \leq \bar{S}, \tag{2}$$

$$\sum_{p=1}^{P_j} x_{pj} = 1 \; ; \; j = 1, \ldots, J, \tag{3}$$

$$x_{pj} \in [0,1], \; j = 1,\ldots,J; \; p = 1,\ldots, P_j \tag{4}$$

where f is the total incremental cost to be disbursed by the controlling agency in order to achieve a sediment load $\bar{S}$ in the pollution sink. We note that if the incremental cost is negative, r_{pj} is set to zero and the corresponding profit goes to the landowner. This is the basis for an incentive system to induce owners to change their management practices to correspond with the policies

resulting from the solution of problem (1) through (4). Expression (2) is a knapsack constraint limiting the sediment generation, and constraints (3), the multiple choice (or GUB) constraints, insure that only one choice set from each transect is selected. This problem has similarities with the constrained assignment problem (Agarwal[8]) and the multiple choice programming problem (Healy[9], Sinha and Zoltners[10], Armstrong et al.[11], and Chang and Tcha[12]).

The Dynamic Programming Formulation

The abatement cost frontier required for policy evaluation is given, for all $S \in \{$ feasible depositions $\}$, by the value function:

$$\Phi(S) = \underset{x}{\text{Min}} \{ f \mid (2),(3),(4) \}. \qquad (5)$$

This information requirement (see detailed discussion in Bouzaher et al.[1]) directly affects the type of approach for the solution of problem (1) through (4). While it is clear that the special zero-one knapsack formulation with GUB constraints is amenable to specialized integer programming methods such as those of Sinha and Zoltners[10], Balas[13], Murphy[14], and others, the number of solutions required would make these approaches inefficient, and more so as the problem size increases. These difficulties are compounded by the absence of computationally useful sensitivity analysis results in integer programming.

In this paper, an efficient dynamic programming (DP) approach is proposed that generates all the solutions of the knapsack problem (1) through (4), thus providing the frontier of $\Phi(S)$ containing the needed policy trade-off information.

The formulation takes advantage of the special structure of the knapsack problem. Basing the stages of the DP on the GUB constraints (3), the problem can be formulated as a multistage decision process where the multistage nature of the model coincides with the spatial decomposition of the transects. This formulation requires only J stages, one state variable, and P_j decision variables per stage, where P_j varies with each stage.

Let the multistage decision process be given by:

$$\{ \bar{S} , \underline{x} , T(\bar{S} , \underline{x}) , g(\bar{S} , \underline{x}) , j \}$$

where

j = stage number corresponding to transacts $j = 1, ..., J$.

$\bar{S}^j$ = state variable giving the maximum total sediment to allocate among the remaining transacts: $j, j+1, ..., J$;

$\underline{x}^j$ = vector of binary decision variables at stage j, and $\underline{x}^j \in R^{P_j}$ corresponds to the number of choice sets available within transect j.

$T(\bar{S}^j , \underline{x}^j)$ = transition function describing the sediment movement process at stage j such that:

$$\bar{S}^{j-1} = T(\bar{S}^j , \underline{x}^j) = \bar{S}^j - \sum_{p=1}^{P_j} s_{pj} x_{pj} = \bar{S}^j - x_{p^*_j} S_{p_j}$$

where $x_{p^*_j} = 1$ and $x_{pj} = 0$, for $p \neq p^*$ (the index of the selected choice set); and

$g(\bar{S}^j , \underline{x}^j)$ = the immediate cost incurred at stage j, given that the maximum limit on sediment generation is $\bar{S}^j$ and that option $\underline{x}^j$ is chosen.

The optimal objective function value for a j-stage process is given by:

$$\Phi^*(\bar{S}^j) = \min_{\underline{x}^k} \left[\sum_{k=1}^{j} (\bar{S}^k , \underline{x}^k) \right] = \min_{\underline{x}^k} \left[\sum_{k=1}^{j} \sum_{p=1}^{P_k} r_{pk} \cdot x_{pk} \right]$$

$k = 1,..., j \qquad k = 1,..., j$

An optimal policy consists of an optimal choice set for each transect and is given by the solution of the following DP functional equations:

for $j = 1, ..., J - 1$:

$$\Phi^*_j(\bar{S}^j) = \min_{\underline{x}^j} \{ g(\bar{S}^j , \underline{x}^j) + \Phi^*_{j+1}[T(\bar{S}^j , \underline{x}^j)] \}$$

$$= \min_{\underline{x}^j} \{ \sum_{p=1}^{P_j} r_{pj} \cdot x_{pj} + \Phi^*_{j+1}(\bar{S}^j - \sum_{p=1}^{P_j} s_{pj} \cdot x_{pj}) \},$$

$$\text{subject to:} \quad \sum_{p=1}^{P_j} x_{pj} = 1 \qquad (6)$$

$$x_{pj} \in [0,1] \,,\; p=1,\ldots P_j$$

$$\bar{S}^j \text{ feasible;}$$

for j = J:

$$\Phi^*_J(\bar{S}^J) = \min_{\underline{x}^J} \{ g(\bar{S}^J, \underline{x}^J) \}$$

$$= \min \{ \sum_{p=1}^{P_J} r_{pJ}.x_{pJ} \}$$

$$\text{subject to:} \quad \sum_{p=1}^{P_J} x_{pJ} = 1 \qquad (7)$$

$$x_{pJ} \in [0,1],\; p=1, \ldots, P_J.$$

Microcomputer Implementation

The DP algorithm was implemented on a Zenith PC/AT microcomputer using an Intel 80286 microprocessor running an 8 MHZ clock speed and zero wait states. At the time this program was being developed, 80386 PC-based computers with clock speeds of 20 MHZ and more were making their first appearance in the marketplace. It is expected that the implementation results reported here will be even more pronounced on 80386 and 80486 machines. The program was coded in Microsoft Pascal and optimized within the 64 kilobytes limitation on memory segments of the PC/AT. The following is description is provided before the flowchart of the algorithm is given.

The program starts by reading the number of stages (J), the number of variables for each stage j (P_j), and the objective function and knapsack constraint coefficients for each variable for each stage, ($r_{1j} >, r_{2j} >, \ldots, r_{P_j j}$) and ($s_{1j} \geq, s_{2j} \geq, \ldots \geq, s_P 12j_j$), respectively. The recursive equations (6) and (7) are then solved in "tabular" form by discretizing the state variable $\bar{S}$ over its feasible range; a subscript k is used to refer to this discretization as indicated below. In addition, let

$$\rho^J_{min} = S_{1J} \text{ and } \rho^J_{max} = S_{P_J J}$$

$$\rho^j_{min} = \sum_{l=j}^{J} s_{1l} \text{ and } \rho^j_{max} = \sum_{l=j}^{J} s_{P_l l} \text{ for } j = 1,\ldots, J-2.$$

We then have

for stage J:

for each k:

$$\Phi^*_J(\bar{S}^J_k) = \text{Minimum } \{r_{pJ}\} \qquad (8)$$
$$p = 1,\ldots,P_J$$

$$\rho^j_{min} \leq \bar{S}^j_k \leq \rho^j_{max} \qquad (9)$$

$\bar{S}^J_k$ discrete and finite;

for stages j = J - 1, ..., 1:

for each k:

$$\Phi^*_j(\bar{S}^j_k) = \text{minimum } \{r_{pj} + \Phi^*_{j+1}(\bar{S}^j_k - s_{pj})\} \qquad (10)$$
$$p = 1,\ldots,P_j$$

$$\rho^k_{min} \leq \bar{S}^j_k \leq \rho^j_{max} \qquad (11)$$

$\bar{S}^j_k$ discrete and finite.

Central to the efficient implementation of this DP algorithm are three main tasks performed in the process of table generation:

1. Discretization of the state variable. Because the real knapsack coefficient data (sediment) is given in a decimal form, computations involved in table generation are carried out in decimal precision. However, because the state variable $\bar{S}^j$ is discretized in steps of size one, a minimal amount of rounding is performed only when necessary. This results in solution values that are very accurate. Steps of size intervals different from one are allowed if one is willing to make trade-offs among accuracy, table size, and computational speed, as discussed below.

2. The use of constraints (9) and (11) reduces the size of the DP tables considerably. These constraints are problem dependent and arise naturally from the fact that, for l = j, ..., J:

$$s_{1l} = \text{minimum } (s_{pl}) \text{ and } s_{P_l l} = \text{maximum } (s_{pl}).$$
$$p = 1, ..., P_l \qquad p = 1, ..., P_l$$

We also note that in some situations subjective lower and upper bounds can be specified to further reduce the table size.

3. Table generation. Stage tables are not generated and stored explicitly as such, but as linked lists, which saves storage and allows backtracking for optimal solutions. A key feature is the use of the monotonicity of the objective function value to reduce both computations and storage. Instead of evaluating Φ_j^* at all discrete values in the state variable range $[\rho_{min}^j, \rho_{max}^j]$, a "divide and conquer" procedure is used to search for the boundary points of the step function. This is essentially a *binary search procedure*.

Dropping the stage index j for clarity of notation, the table generation procedure is summarized below

Initialization: Given ρ_{min}, ρ_{max}, insert a median point ρ_{med}. Compute $\Phi_1 = \Phi(\rho_{min})$, $\Phi_2 = \Phi(\rho_{med})$, $\Phi_3 = \Phi(\rho_{max})$. Let L be the interval $[\rho_{min}, \rho_{max}]$; add L to the list of search regions R. Go to the main step.

Main Step:

1. If R = Φ, stop with the stage table at hand; otherwise pick a search region from the list R and go to 2.
2. If $\Phi_1 = \Phi_2 = \Phi_3$, add one entry to the stage table; go back to 1.
3. If $\Phi_1 \neq \Phi_2 \neq \Phi_3$, create two new search regions: $L_1 = [\rho_{min}, \rho_{max} = \rho_{med}]$, $L_2 = [\rho_{min} = \rho_{med}, \rho_{max}]$. For each region insert a median point and compute the corresponding Φ_1, Φ_2, Φ_3. Add L_1 and L_2 to R; go back to 1.

4. If $\Phi_1 \neq \Phi_2 = \Phi_3$, $\Phi(\overline{S}_k) = \rho_{min} < \overline{S}_k < \rho_{med}$. Add one entry to the stage table corresponding to ρ_{min}; go back to 1.

5. If $\Phi_1 \neq \Phi_2 = \Phi_3$, $\Phi(\overline{S}_k) = \Phi_2$ for $\rho_{med} \leq \overline{S}_k \leq \rho_{max}$. Add one entry to the stage table corresponding to ρ_{med}; go back to 1.

We note that this procedure's efficiency is related to the number of segments in the step function $\Phi_j^*(\cdot)$ (Figure 1). The worst case bound is attained when the step function degenerates into single points corresponding to $\{\overline{S}_k^j$, for all k$\}$.

Computational Testing and Results

To assess the efficiency of the DP algorithm, two types of data sets were used for the knapsack problem's coefficients $\{r_{ij}, s_{ij}\}$: randomly generated problems from a pool of real transect data and pure randomly generated problems.

In the first case a number of data sets from actual application sites (see Bouzaher et al.[1] and Braden et al.[5]) were pooled and an experiment was designed for problem generation. The main purpose was to test the behavior of the algorithm in the face of real data magnitudes both in absolute and differential terms.

In the second case, coefficient distributions with certain characteristics were specified and another set of problems was experimentally designed.

The idea was to test sensitivity of the algorithm's robustness and efficiency to a number of parameters:

1. real versus randomly generated data;
2. coefficient ranges for random data;
3. coefficient ranges for real data (random generation from real data was also used because there was apparent structure or pattern to take advantage of);
4. computation time;

5. accuracy of the solutions (as it correlates with the discretization level of the state variable);
6. storage requirements for table generation (both overall and for individual tables); and
7. comparative performance with standard integer programming codes.

The performance of the DP algorithm (DPOPT) was compared with that of APEX[15], a well-known mixed integer commercial code. Two other PC-based mixed-integer softwares were tested, but had very limited capabilities: LINDO[16] and MILP88[17]. Both DPOPT and APEX were run on the same data sets. However, it is important to note that in the case of DPOPT an entire family of solutions, given by equation (5) above, was generated, while APEX solutions are obtained for single values of the RHS of the knapsack constraint. An *average solution point* will thus be computed for DPOPT. For example, if for a given problem, DPOPT uses 10 CPU seconds to generate 20 points on the optimal frontier $\Phi(S)$ for all $S \in [10,100]$, then 0.5 CPU seconds is used as the computation time for an *average solution point*. This point is then compared to APEX's solution time for solving a problem corresponding to a *single value* of S (e.g. S = 50). to give a basis for comparison.

The Test Problems

One hundred forty-four test problems, solved using both DPOPT and APEX, were constructed as follows:

1. Random problems with real data coefficients: From a pool of real transect data four replications of each of eight different size problems (50,100,200,300,400,500,600,800 variables) were generated, giving 32 test problems.

2. Random problems with random coefficients:

   a. From a uniform distribution for each of the ranges, [0,1], [0,10], [0,100], [0,1000], four replications for each of four different size problems (50,100,200,300 variables) were generated, giving 64 test problems.

b. From a uniform distribution for each of the ranges, [0,1], [0,10], [0,100], four replications for each of four different size problems (400,500,600,800 variables) were generated, giving 48 test problems.

We note that in all cases the number of multiple choice constraints (corresponding to the number of transects) was kept constant and not parametrized. This was essentially used as a blocking factor.

The Results

Tables 1 and 2 summarize results for DPOPT, averaged over the four replications. These tables contain information on:

1. Table size, representing the number of discrete feasible values at which the state variable is actually evaluated (entries of the last final DP table); it is also the best proxy for storage requirement, a complexity parameter. In addition to information on the smallest and largest tables built by the DP algorithm, it is to be noted that table size increases monotonically with the number of stages.

2. Table spread, another complexity parameter, gives the number of potential state variable values that need to be evaluated (the final number of discrete levels of the state variables), and is used to evaluate the efficiency of the binary search procedure.

3. CPU time gives an indication of the computational effort needed to generate not only all the solutions corresponding to the entire abatement-cost frontier, but also the "average" single solution to be compared to other approaches.

Tables 3 and 4 give summarize results for APEX, averaged over the four replications. The relevant response variable is CPU time for solving the special problem (1) through (4). In every case the RHS of the knapsack constraint was set to the mid-point of the state variable range in the corresponding DP test problem.

The major computational findings are summarized in a series of graphs. Figure 2 shows that for an *average point* on the solution frontier, the dynamic programming algorithm, implemented on a microcomputer, outperforms APEX, a mainframe commercial package, for the special class of knapsack formulated environmental problems. In addition, the performance advantage of DPOPT increases with problem size. Furthermore, the behavior of DPOPT does not show expected exponential growth in CPU time as the size of the problem increases. We purposely confined the experimental range of problem size to 800 variables based on real size applications, as discussed in Bouzaher et al.[1]

Figure 3 shows another aspect of the efficiency of DPOPT--actual table size is always lower than its limiting case, table spread. Here again, the efficiency differential increases with problem size. In the random data case, Table 5 and Figure 4 shows that table size ratio increases very slowly with problem size but improves markedly with data range size, a complexity parameter.

Figure 5 shows that another important complexity feature of knapsack problems is also shared by our class of problems--computational effort with DPOPT (both CPU and storage) gets worse as the data range, from which the problem coefficients are drawn, increases. This finding seems to be independent of solution strategies, as evidenced by the work of Balas and Zimel[13]. In the case of APEX, Figure 6 shows a similar trend, even though some bias is introduced due to the *near-optimal* stopping criterion used by this package.

The computational effort feature of the knapsack problem is of course very important, but (fortunately) for the class of nonpoint source pollution control problems for which DPOPT was initially designed, real data coefficient (returns and sediment loads) are always in the the lower ranges, as evidenced from comparing Figures 2 and 4.

In addition, we note that DPOPT solved all the problems to optimality with an amazing accuracy, as compared with APEX. This feature shows the robustness of the procedure.

Finally, we note that the computational experiment designed at the outset enabled us to conduct analyses of variance and confirm that:

1. The DP approach is significantly more efficient than APEX;
2. There are no significant differences between the two approaches as far as the importance of problem size and coefficients range in determining computation times; and
3. The variance of CPU time increases with problem size, as revealed by residual plots from both approaches.

Conclusion

This paper presents the computer implementation and testing of a dynamic programming algorithm designed to solve a specially structured class of knapsack problems. These problems arise naturally in certain sediment-based environmental control problems.

Detailed controlled computational experimentation with the DP algorithm, DPOPT, has shown that the approach is both robust and efficient. In addition to its portability, DPOPT has two types of advantages relative to standard integer optimization codes.

1. Computational Advantages. Besides the apparent efficiency advantage over APEX, using a coarser step size in the discretization of the state variable can cut solution time dramatically if the analyst is willing to give up accuracy (that may entail accepting a more costly solution and differences in policy recommendations). Also, it is clear that the step size option enables the decision maker to attempt even larger problems (entire regions or states in the case of sediment control), especially if the aim is to obtain general behavioral guidelines,

2. Information Advantages. Because of the multistage-recursive nature of DPOPT, it not only generates all solutions for all possible values of the state variable, but more important, it generates all optimal solutions for all intermediate stages, that correspond to subregions of the area under study. This can also be very helpful in sensitivity analysis. For the case of sediment control this information advantage is fundamental in identifying site specific control measures and targeting more polluting land management units (see Bouzaher et al.[1] for more discussion of this point).

In addition, this paper demonstrates the usefulness of dynamic programming for knapsack problems with multiple choice constraints and multiple right hand sides; as such, it should be of interest to practitioners and researchers alike in that the algorithm presented here can provide sensitivity analysis information. Finally, this paper shows the importance of computers and Operations Research methodology in both modeling and solving complex environmental problems, a nonconventional applications area.

Table 1. DPOPT solutions with real data
(averages over four replications)

Problem size	Table size min	Table size max	Table spread min	Table spread max	CPU time (seconds) all solutions[a]	CPU time (seconds) one solution[b]
50	1.5	17.8	1.7	210.0	1.425	0.084
100	2.5	112.0	25.7	626.9	21.000	0.188
200	3.0	257.0	36.5	1243.2	123.874	0.482
300	2.0	571.0	64.2	2148.3	381.428	0.668
400	1.0	692.5	0.0	2475.4	549.300	1.115
500	1.8	769.8	9.4	2845.7	1070.600	1.391
600	1.0	807.0	1.7	3276.1	1522.900	1.887
800	2.8	1584.5	28.2	4662.2	3491.100	2.203

[a]This represents the time to compute the optimal frontier $\Phi(S)$ for all S.
[b]This represents the average time to compute a single point on the optimal frontier.

Table 2. DPOPT solutions with random data
(averages over four replications)

Pb. size	Coef. spread	Table size min / max		Table spread min / max		CPU time (seconds) all sol.[a]	one sol.[b]
50	[0,1]	1.25	2.25	0.23	0.87	0.15	0.067
50	[0,10]	2.75	14.25	3.60	42.10	0.93	0.065
50	[0,100]	3.75	26.50	55.33	270.00	3.73	0.141
50	[0,1000]	1.75	26.00	105.95	2873.00	5.55	0.213
100	[0,1]	1.25	3.25	0.30	2.13	0.40	0.123
100	[0,10]	1.75	31.80	4.23	53.50	3.73	0.117
100	[0,100]	2.50	91.00	45.80	576.70	20.35	0.223
100	[0,1000]	4.25	123.50	595.23	6848.10	66.43	0.540
200	[0,1]	1.00	5.00	0.00	4.60	1.08	0.196
200	[0,10]	2.25	54.30	2.33	90.90	16.00	0.295
200	[0,100]	2.00	225.50	22.13	1158.90	120.00	0.532
200	[0,1000]	1.75	250.80	547.13	12320.30	288.90	1.152
300	[0,1]	1.00	7.00	0.00	5.60	2.10	0.300
300	[0,10]	2.00	90.50	2.93	159.60	45.85	0.507
300	[0,100]	4.50	275.50	54.58	1614.40	292.13	1.060
300[c]	[0,1000]	4.00	402.00	598.20	16251.30	611.70	1.522
400	[0,1]	1.00	4.70	0.00	2.57	2.03	0.432
400	[0,10]	3.25	134.80	5.30	232.80	98.40	0.734
400	[0,100]	3.25	490.30	33.45	2190.50	616.60	1.258
500	[0,1]	1.25	8.50	0.38	7.90	4.35	0.512
500	[0,10]	2.25	181.80	3.93	251.60	139.20	0.766
500	[0,100]	1.50	640.00	31.75	2657.70	1020.58	1.595
600	[0,1]	1.00	9.30	0.00	8.70	4.83	0.519
600	[0,10]	3.00	223.00	5.10	290.70	197.33	0.885
600	[0,100]	3.75	924.00	36.25	3363.70	1802.48	1.951

[a]This represents the time to compute the optimal frontier $\Phi(S)$ for all S.
[b]This represents the average time to compute a single point on the optimal frontier.
[c]Table size exceeded the DPOPT limit. Thus, only the first three data coefficient spreads will be considered for subsequent problem sizes.

Table 3. APEX solutions with real data
(averages over four replications)

Problem size	CPU seconds	% from optimum[a]
50	0.566	0.0
100	1.137	4.1
200	2.419	2.2
300	3.520	0.43
400	5.790	1.22
500	6.800	0.67
600	9.360	0.21

[a]Readers familiar with APEX know this stopping criterion feature of the code; it can be either specified by the user or, as in this case, endogenously determined.

Table 4. APEX solutions with random data (averages over four replications)

Problem size	Coefficient spread	CPU seconds	% from[a] optimum
50	[0,1]	1.166	2.67
	[0,10]	0.785	0.00
	[0,100]	1.094	4.73
	[0,1000]	1.059	3.51
100	[0,1]	3.712	4.92
	[0,10]	2.325	5.14
	[0,100]	2.111	1.69
	[0,1000]	1.848	2.64
200	[0,1]	4.686	1.72
	[0,10]	11.611	11.43
	[0,100]	8.405	0.75
	[0,1000]	9.057	3.24
300	[0,1]	11.627	3.69
	[0,10]	12.283	2.66
	[0,100]	15.272	1.53
	[0,1000]	33.123	2.61
400	[0,1]	38.893	0.92
	[0,10]	18.075	na
	[0,100]	21.854	na
500	[0,1]	18.621	3.31
	[0,10]	31.836	1.64
	[0,100]	31.902	6.44
600	[0,1]	29.319	3.44
	[0,10]	32.007	4.74
	[0,100]	27.504	2.44

[a]See the footnote for Table 3.

Table 5. Table size ratios (TSR)
(averages over four replications)

Pb. size	Real data TSR (%)	Random data coef. spread	Random data TSR (%)
50	8.3	[0,1]	a
		[0,10]	33.8
		[0,100]	9.8
		[0,1000]	0.9
100	17.8	[0,1]	a
		[0,10]	59.3
		[0,100]	15.8
		[0,1000]	1.8
200	20.7	[0,1]	a
		[0,10]	59.7
		[0,100]	19.5
		[0,1000]	2.1
300	26.6	[0,1]	a
		[0,10]	56.7
		[0,100]	17.1
		[0,1000]	2.4
400	19.9	[0,1]	a
		[0,10]	57.9
		[0,100]	22.4
500	27.1	[0,1]	a
		[0,10]	72.2
		[0,100]	24.1
600	24.6	[0,1]	a
		[0,10]	76.7
		[0,100]	27.5
800	33.9	na	na

Note: TRS measures the efficiency of the binary search for the DP tables generation.

[a]In all these cases TSR > 100%. However, this is not significant because the algorithm is designed to work with a state variable step size of one; rounding then makes it possible to have a table size larger than table spread.

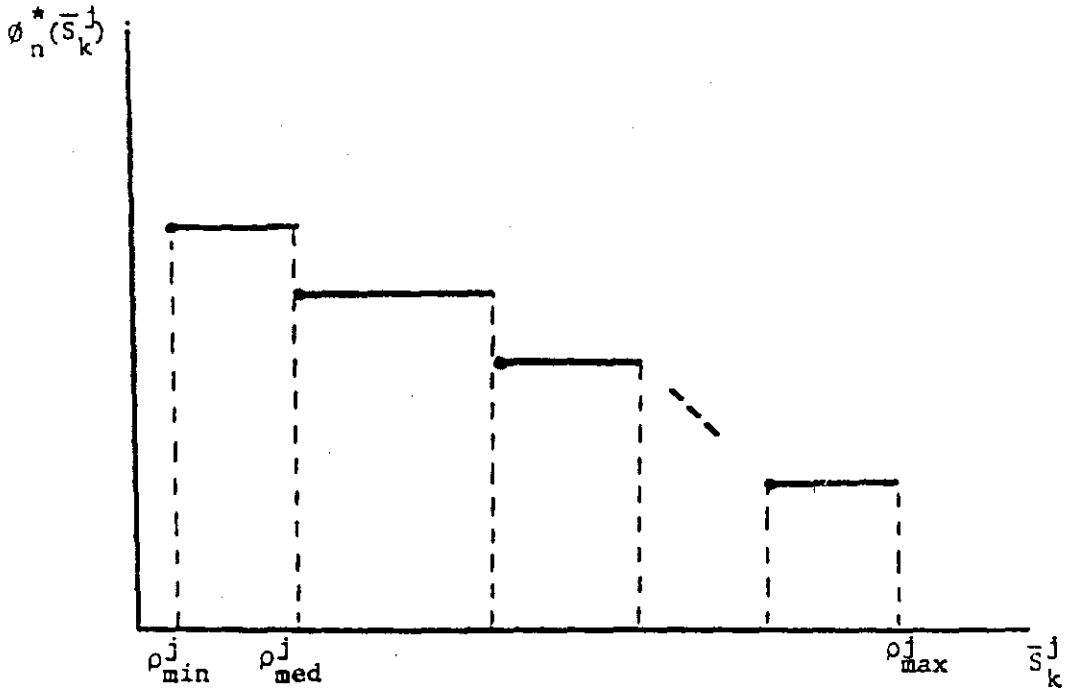

Figure 1. Table generation procedure

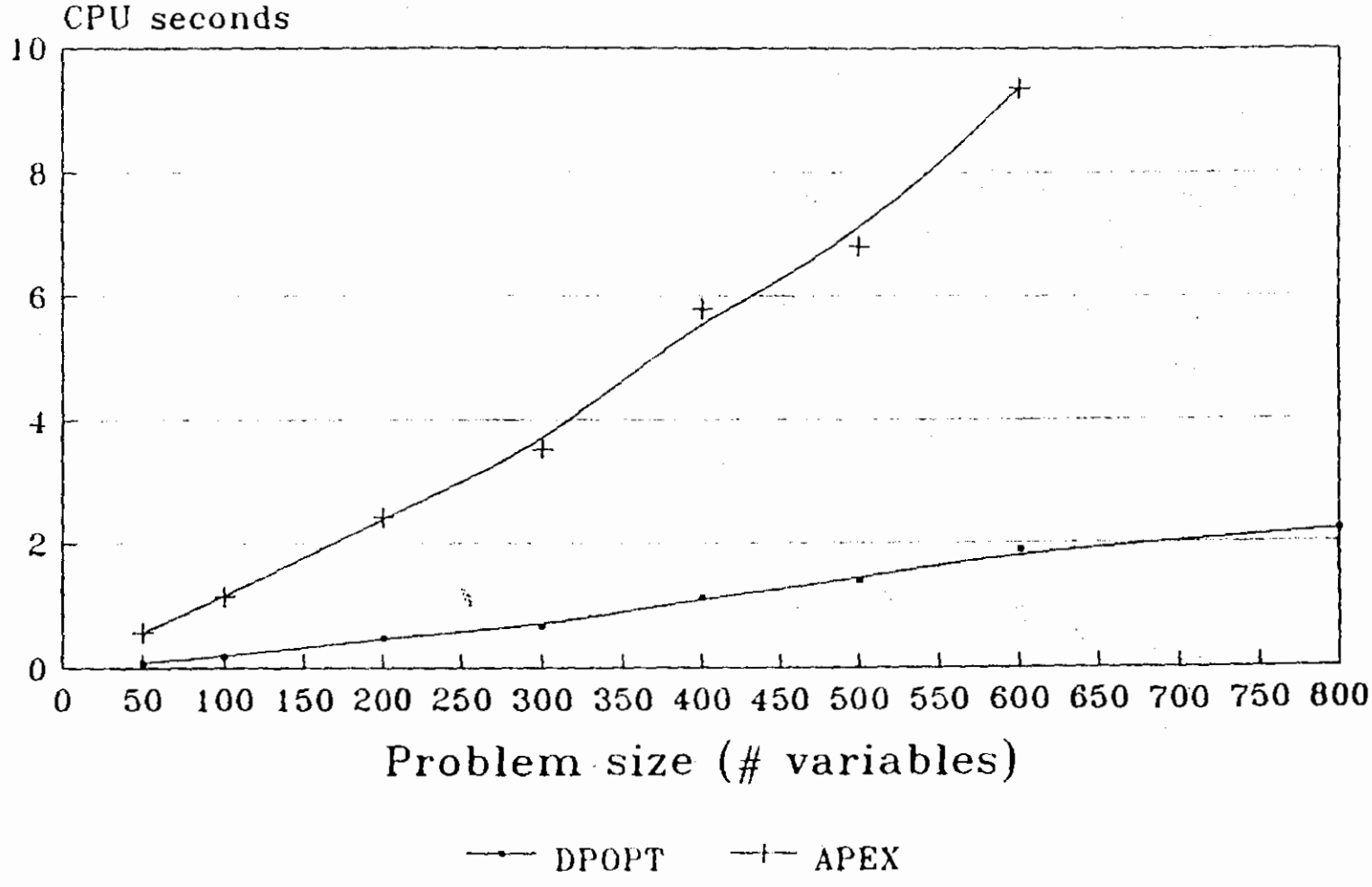

Figure 2. Computation times for DPOPT and APEX (random problems using real data)

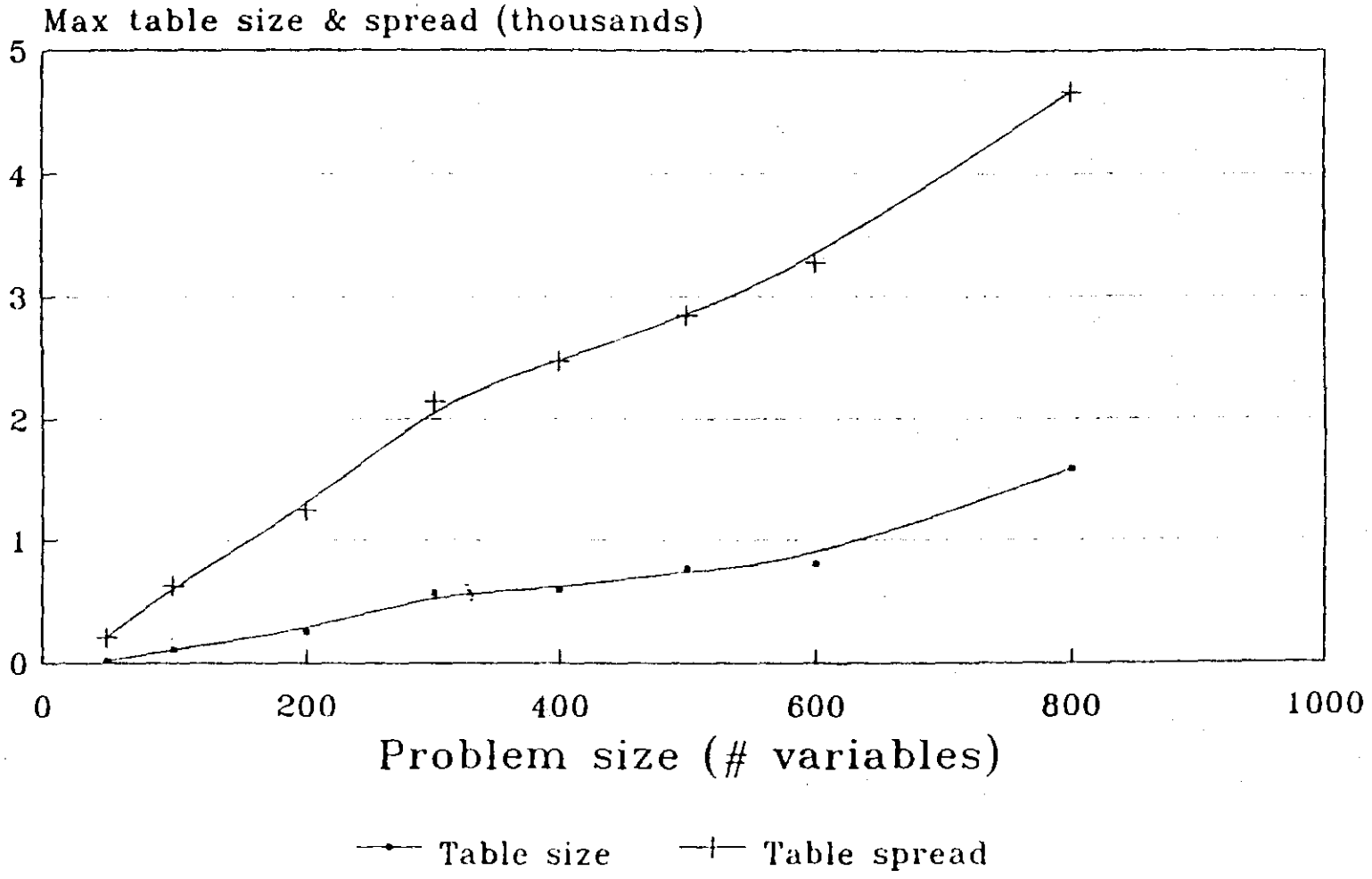

Figure 3. Maximum table size and spread for DPOPT (random problems using real data)

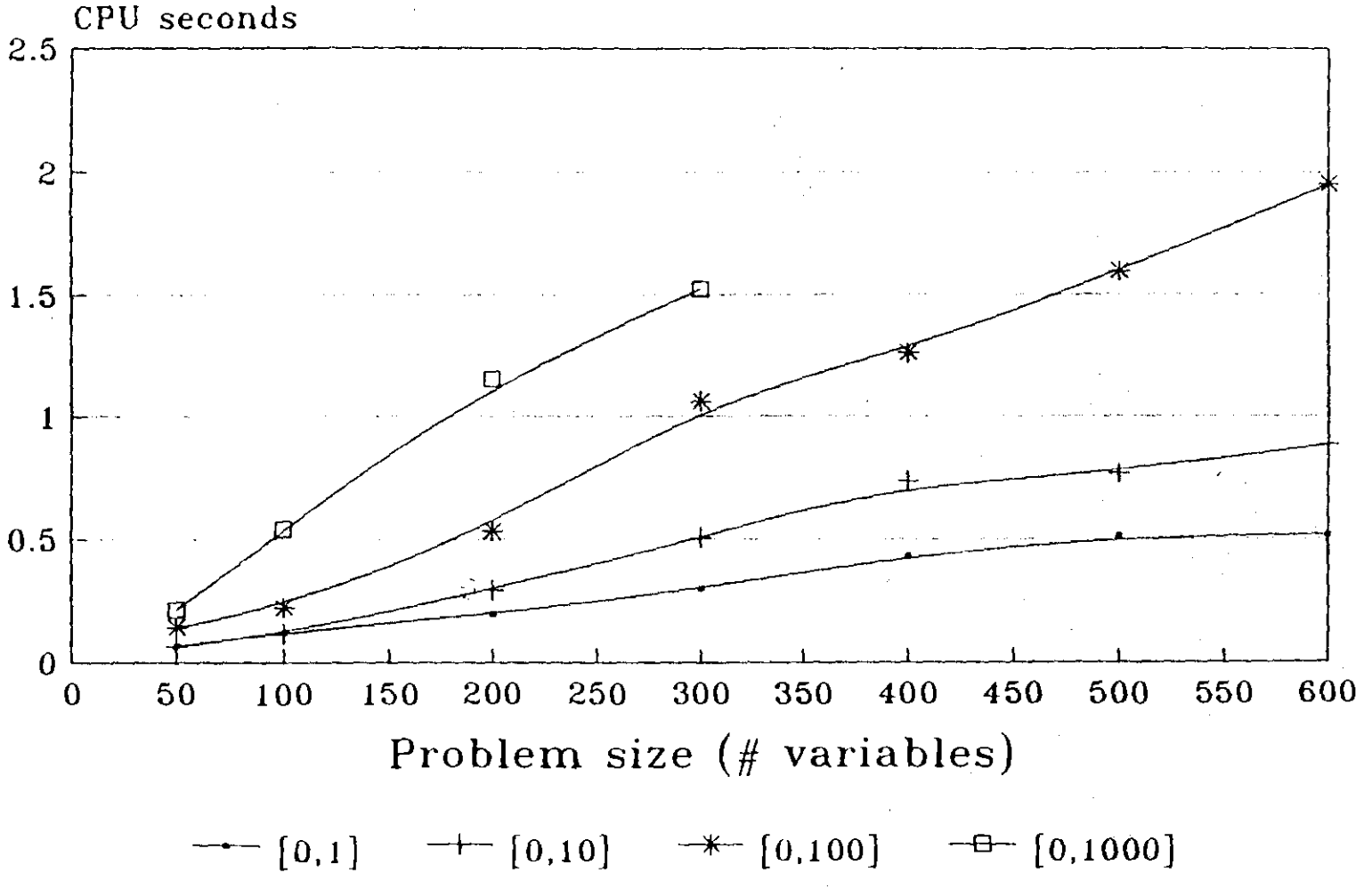

Figure 4. Computation times for DPOPT (randomly generated problems)

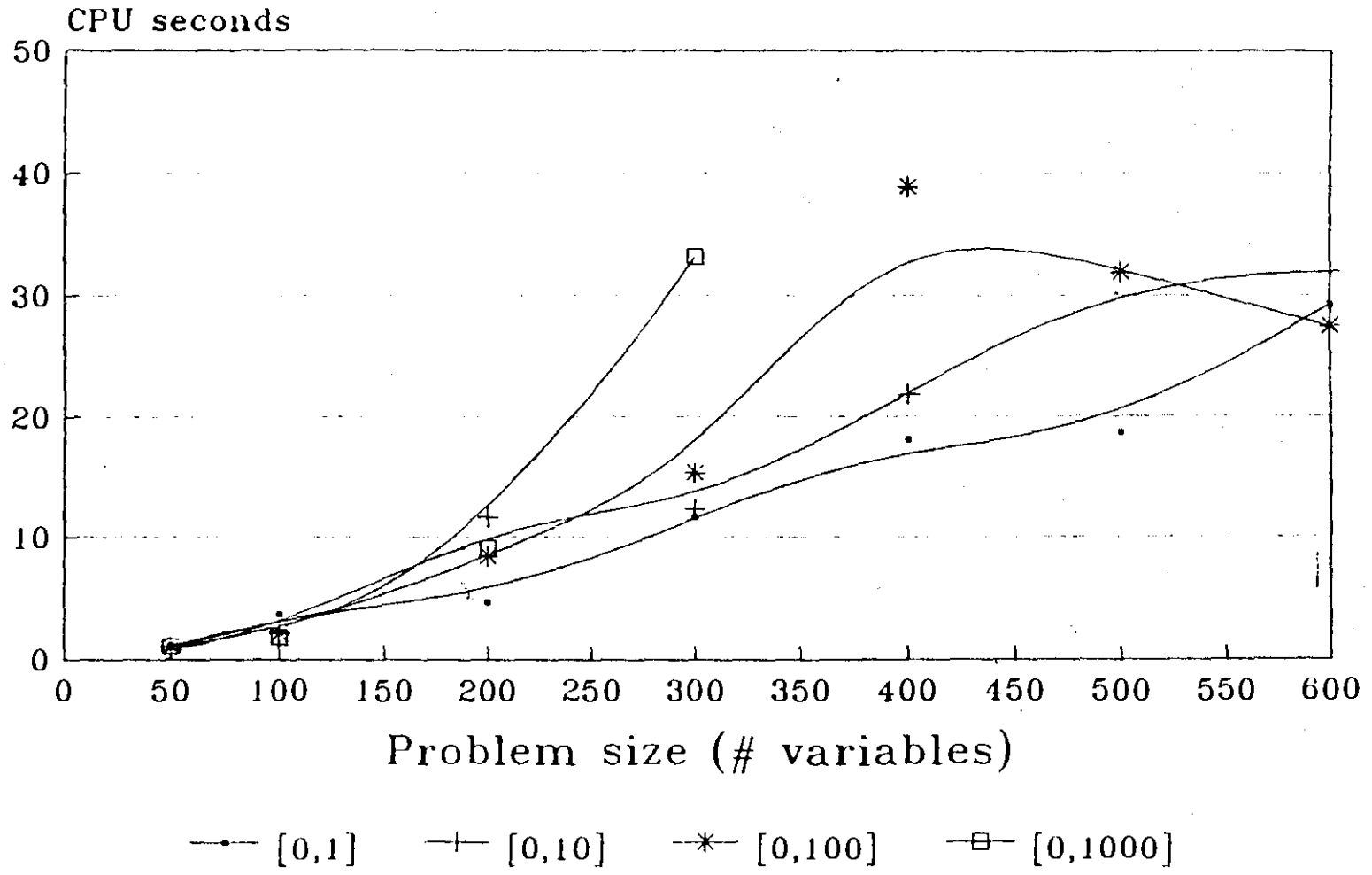

Figure 5. Computation times for APEX (randomly generated problems)

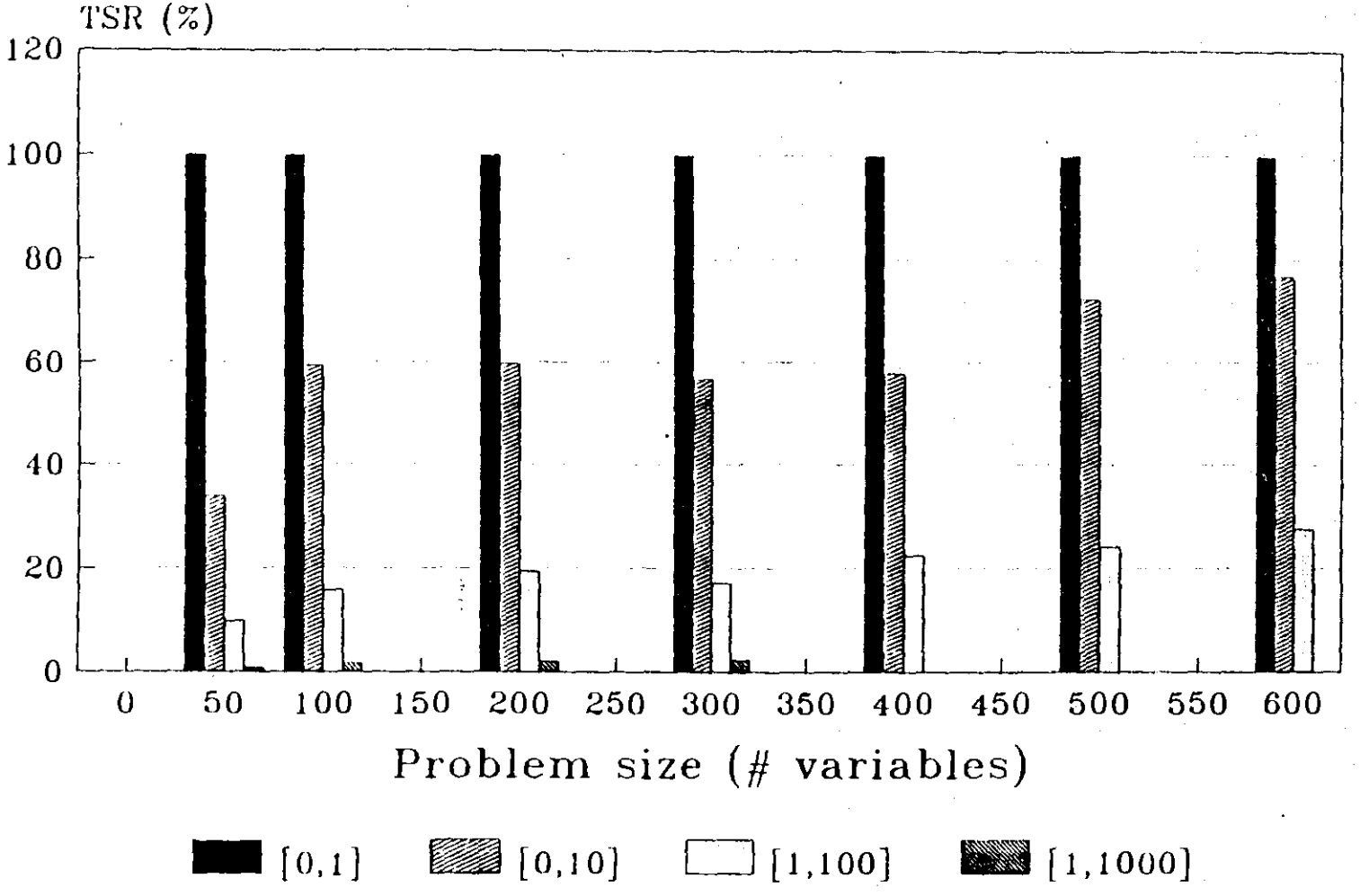

Figure 6. Table size ratio for TSR (randomly generated problems)

REFERENCES

1. A. Bouzaher, J.B. Braden and G.V. Johnson (1990) A Dynamic Programming Approach to a Class of Nonpoint Source Pollution Control Problems. Management Science, Vol. 36, No. 1, 1-15.

2. A. Bouzaher, S.E. Murley, G.V. Johnson and J.B. Braden (1988) SEDEC: A Sediment Economics Simulation Model. Computer Methods and Water Resources: Computer Aided Engineering in Water Resources, ed. D. Ouazar and C. Brebbia, 197-208, Southampton, Eng.: Computational Mechanics Publications, Springer-Verlag.

3. J.B. Braden, A. Bouzaher, G. V. Johnson and D. Miltz (1987) Separability, Recursion, and Targeting in Environmental Management. Unpublished Manuscript, University of Illinois at UC, DAE/IES, Urbana, Illinois.

4. J.B. Braden, E.E. Herricks, and R.S. Larson (1989) Economic Targeting of Nonpoint Pollution Abatement for Fish Habitat Protection. Water Reservoir Res. Vol. 25, No. 12, 2399-2405. December.

5. J.B. Braden, G.V. Johnson, A. Bouzaher and D. Miltz (1989) Optimal Spatial Management of Agricultural Pollution. American J. Agr. Economics, Vol. 71, No.2, 404-413.

6. J.B. Braden, G. V. Johnson and D. G. Martin (1985) Efficient control of Sediment Deposition in Water Courses. In Options for Reaching Water Quality Goals, T. M. Schad, Ed. Tech. Pub. Ser. 84-2, American Water Resources Association, Bethesda, MD., 69-76.

7. J.B. Braden, R.S. Larson, and E.E. Herricks (1991) Impact Targets versus Discharge Standards in Agricultural Pollution Management. Forthcoming, Amer.Jr. Agr. Econ. May.

8. V. Aggarwal (1985) A Lagrangian Relaxation Method for the Constrained Assignment Problem. Comput. Opns. Res. Vol. 12, No.1, pp. 97-106.

9. W.C. Healy (1964) Multiple Choice Programming. Oper. Res. Vol. 12, 122-138.

10. P. Sinha, and A.A. Zoltners (1979) The Multiple Choice Knapsack Problem. Oper. Res. Vol. 3, N0. 3, 503-515.

11. R.D. Armstrong, P. Sinha, and A.A. Zoltners (1982) The Multiple Choice Nested Knapsack Model. Mgt. Sc. Vol. 28, No. 1, 34-43.

12. S.G. Chang and D. W. Tcha (1985) A Heuristic for Multiple Choice Programming. Comput. Opns. Res. Vol. 12, No.1, 25-37.

13. E. Balas and E. Zemel (1980) An Algorithm for Large Zero-One Knapsack Problems. Oper. Res. Vol. 28, No. 5, 1130-1154.

14. R.A. Murphy (1986) Some Computational Results on Real 0-1 Knapsack Problems. OR Letters Vol. 5 No.2, 67-71.

15. APEX-III, Reference Manual Version 1.2, (1979) Control Data Corporation.

16. LINDO, (1987) User's Manual Version 2.0, LINDO Systems Inc.

17. MILP88 (1987) Mixed-Integer Linear Programming for the IBM PC, User's Manual Version 6.02, Eastern Software Products.