# A BOOKKEEPING STRATEGY FOR MULTIPLE OBJECTIVE LINEAR PROGRAMS

ALOK AUROVILLIAN *, HONG ZHANG † AND MALGORZATA M. WIECEK ‡

**Scope and Purpose**– A fundamental problem in multiple objective linear programming is to determine the efficient set that often has a large or infinite number of solutions. The problem becomes even more complex if a multiple objective linear program (MOLP) is of a large size. Some well established sequential algorithms for MOLPs, such as that used by the software package ADBASE, are extensions of the simplex method for linear programming. The growing importance and enormous computational expense of MOLPs have motivated the development of parallel algorithms capable of solving very large problems. Based on ADBASE, a parallel ADBASE algorithm significantly accelerating the solution process of MOLPs on multiprocessors was developed.

However, the original ADBASE bookkeeping strategy of maintaining a large unsorted list of efficient solutions was found to be a major bottleneck for sequential as well as for parallel processing. This paper addresses the crucial issue of bookkeeping of efficient solutions. Three representative list creation schemes are compared and one of them, considered optimal, is incorporated into ADSBASE and the parallel ADBASE algorithm. Implementation results on sequential and parallel computers are demonstrated.

**Abstract**– This paper discusses bookkeeping strategies for solving large multiple objective linear programs (MOLPs) on ADBASE, a popular sequential software package, and on a parallel ADBASE algorithm. Three representative list creation schemes were first analyzed and tested. The best of them, Binary Search with Insertion Sort (BSIS), was incorporated into ADBASE and the parallel ADBASE algorithm. These revised versions were compared with their standard counterparts on a series of test problems, and their computation times were compared. The results show that the new bookkeeping strategy for maintaining a list of efficient solutions significantly speeds up the

* Alok Aurovillian is a computer consultant with Shamrock Computer Resources, LTD. He received his M.S. degree in Mathematics from the Department of Mathematical Sciences, Clemson University, and has a B.A. degree in Mathematics and Computer Science from Warren Wilson College, Asheville, NC. His research interests include digital image processing, simulation of queuing models, and parallel processing.(alok@auroville.org.in)

† Hong Zhang received a Ph.D. degree in Applied Mathematics from Michigan State University. She is an associate professor in the Department of Mathematical Sciences at Clemson University, Clemson, SC. Her research interests are in the area of computational mathematics and parallel processing. (hongsu@math.clemson.edu)

‡ Malgorzata M. Wiecek received her M.S. degree in Electrical Engineering and Ph.D. degree in Systems Engineering from the University of Mining and Metallurgy in Krakow, Poland. She is an associate professor in the Department of Mathematical Sciences at Clemson University, Clemson, SC and is currently on sabbatical leave in Fachbereich Mathematik, Universitaet Kaiserslautern, D-6750 Kaiserslautern, Germany. Her professional interests include theory and applications of mathematical programming with emphasis on multiple objective problems. (wmalgor@math.clemson.edu)

process of solving MOLPs, especially on parallel computers.

**1. INTRODUCTION.** Multiple objective programming is concerned with theory and methodology that can treat complex decision making problems encountered in economics, engineering, business, regional planning, and other areas of human activity. A decision making problem is characterized by multiple objectives such as distance, time, cost, reliability and safety. As in general, objectives are noncomparable and conflicting, the solution set of a multiple objective program usually includes a large or infinite number of points referred to as efficient solutions or decisions. Mathematical optimization has become the essential tool for generating efficient solutions; various solution methods have been developed by many researchers in the last two decades [3] [12].

The multiple objective linear programs (MOLPs) considered in this paper are formulated as:

$$max\{z = Cx \mid x \in S\},$$

where

$$S = \{x \geq 0 \mid A_{m_1}x \leq b_{m_1}, A_{m_2}x = b_{m_2}, A_{m_3}x \geq b_{m_3}\},$$

$C$ is an $k \times n$ matrix, $A_{m_i}$ are $m_i \times n$ matrices and $b_{m_i}$ are $m_i \times 1$ vectors with nonnegative components, $i = 1, 2, 3$. A point $x_o$ in $S$ is called an efficient solution of the MOLP if there is no other point $x$ in $S$ such that $Cx \geq Cx_o$, with strict inequality holding for at least one component.

The structure of the efficient set of MOLPs has several special features. Since this set is usually a subset of the convex polyhedral set $S$, it includes efficient extreme points (EEPs) and unbounded efficient edges. The efficient set is also "connected" in the sense that every EEP is connected to every other EEP by a series of efficient edges. Assigning EEPs to nodes and efficient edges to arcs, one can construct a solution graph along which the search for efficient points can be conducted.

The process of generating and storing EEPs and unbounded efficient edges was studied by Zeleny [18], Ecker and Kouda [5], Steuer [12], and many others. Theoretical studies accompanied by implementation efforts resulted in several computer programs for solving MOLPs. In 1976 Zeleny published a Fortran program for computing all EEPs [19] and in 1980 Fotso developed a program for computing all efficient faces [7]. Isermann and Naujoks released EFFACET, a Fortran package also for computing all efficient faces [9]. More recently, Strijbosch et al. presented a simplified MOLP algorithm which is a variant of ADBASE and is more logically transparent to the non-expert users [15]. Armand and Malivert developed the package VERTICES for computing all EEPs and all unbounded efficient edges and the package FACE for computing all efficient faces of an MOLP [1]. ADBASE was developed by Steuer in the 1970's and its revised versions have been released every few years since then [14]. The current version of ADBASE is a stable, efficient and well-written Fortran package for computing all EEPs and all unbounded efficient edges. Among all the available computer programs for

MOLPs, ADBASE is the most widely distributed package and has been used by many researchers. It is in fact the only well established package for solving MOLPs. For a detailed comparison of some well-known algorithms for solving MOLPs, see Strijbosch et al. [15].

Since solving large multiple objective programs often requires intensive computation and large storage space, several studies on generating efficient solutions by parallel processing have also been undertaken. Evtushenko et al. were perhaps first to recognize the need of solving multiple objective programs on parallel computers [6]. In the area of interactive decision making, Costa and Climaco developed a multiple reference point approach to solving MOLPs [4]. In the field of engineering, Chang presented a parallel implementation of power systems optimization with multiple objectives [2]. Wiecek and Zhang initiated studies on finding efficient solutions of MOLPs on parallel computers and developed a parallel ADBASE algorithm, which will be referred to as *Parallel ADBASE* in this paper [16] [17].

Following upon the recent developments in sequential and parallel computation for multiple objective programming, this paper discusses bookkeeping strategies specifically designed for ADBASE and Parallel ADBASE.

ADBASE performs several tasks related to solving MOLPs and analyzing their efficient sets. The part of ADBASE responsible for generating all EEPs consists of three main phases. In Phase I, an initial extreme point is found or the process terminates if such a point does not exist. An initial efficient basis and the corresponding initial EEP (IEEP) are found in Phase II or the process is terminated if no efficient basis exists. In Phase III, given an IEEP, all other EEPs and unbounded efficient edges are found in a systematic manner by pivoting and applying the *bookkeeping/master-list/crashing* strategy. The nonbasic variables of the IEEP are checked for feasibility and efficiency, and all the other EEPs directly connected to the initial one are found. These EEPs are stored in a list, and their corresponding bases, coded appropriately, are stored in a list called LISTB. After the IEEP has been completely examined for neighboring EEPs and the EEPs put on the list, the program moves down the list ("crashes") to the next stored EEP. From this solution, all its neighboring solutions are again examined for feasibility and efficiency. When an EEP is found, it is checked against the list to see if it has been put in the list. If it is not already in the list, it is placed in the list. This process is repeated until all the EEPs have been examined for possible neighboring EEPs.

Parallel ADBASE significantly accelerated the solution process of MOLPs and demonstrated the potential feasibility of ADBASE for solving very large MOLPs on advanced computers [16] [17]. However, it also revealed that the current bookkeeping scheme in ADBASE is a severe bottleneck for sequential processing as well as for parallel processing. It should be noted that ADBASE was initially developed as a sequential software for solving MOLPs of a small to medium size [13], for which bookkeeping strategy was not a concern to the efficiency of the package. However, when solving large MOLPs, especially on parallel computers, bookkeeping becomes the crux of the efficiency of the algorithm. Therefore it appeared worthwhile to devise a new bookkeeping scheme to improve ADBASE and Parallel ADBASE for solving large MOLPs.

The main feature of the current bookkeeping scheme in Parallel ADBASE is to maintain a global updated list of EEPs on all processors, so that the computation for searching and examining new EEPs can be distributed evenly among processors. While this global list reduces the communication overhead for job balance, it introduces redundant computation and storage. In [17], the pros and cons of replacing the global bookkeeping with local lists were considered and discussed. Preliminary analysis and investigation have been conducted since then and no convincing evidence demonstrating a favorable local list scheme has been found. Therefore, the research work reported here focuses on list creation schemes in bookkeeping for both ADBASE and Parallel ADBASE, while Parallel ADBASE still adopts the same global bookkeeping strategy.

This paper is organized as follows. First, three representative list creation schemes are examined. Then, having decided upon the most suitable scheme for maintaining the list of efficient bases, this optimal scheme is incorporated into ADBASE and comparisons are made with the original ADBASE. Finally, the resulting new bookkeeping strategy is implemented into Parallel ADBASE and the importance of the new strategy is discussed.

**2. LIST CREATION SCHEMES.** List creation is one of the most common operations performed by a computer. Many list creation schemes have been well developed and widely used as building blocks in complex numerical algorithms. Based upon the nature of MOLPs and ADBASE methodology, we selected three representative list creation schemes for comparison, each with different structural and algorithmic representations:

1. Unsorted List (UL),
2. Binary Search with Insertion Sort (BSIS),
3. Linked List Sort (LLS).

In these schemes, two important processes were distinguished: the search for the number on the list and the insertion of the number into the list if it is not already on the list.

The first scheme, referred to as UL, simulates the ADBASE package, where the list of coded bases, LISTB, is maintained in an unsorted manner. In this scheme, every time a new number is generated, it is compared to each entry in the list. If the number is not already on the list, it is then inserted at the bottom of the list. Thus, the whole list needs to be scanned each time and the insertion occurs at the bottom.

In general, sorted data are easier and faster to manipulate than randomly ordered data. It was conjectured that maintaining a sorted list of numbers would speed up the process of determining whether a number was already on the list. This led to choosing two most representative schemes that maintain a sorted list but differ in the search techniques and the storage of the numbers. Scheme 2, referred to as BSIS, is a well-known computational scheme for searching and maintaining a list of numbers as an array, while scheme 3, referred to as LLS, is known for maintaining a sorted linked list of nodes which contain the numbers [8].

The computational complexities of the three schemes are given in Table 1. Even though all the schemes considered have the same overall complexities, it is important to note the differences in their search and insertion times. A careful observation indicates that BSIS and LLS outperform the UL scheme in general, and BSIS is the best scheme for very large lists. This is because (1) for a newly found number, $O(n^2)$ is the actual search complexity for UL, an upper bound of search complexity for LLS, and an upper bound of insertion complexity for BSIS attainable only when the new number appears at the very top of the list; (2) for a redundant number, insertion is not performed; (3) for BSIS, when the list is large, its search time is negligible compared to its insertion time. The insertion could be time consuming but its time is at most comparable with and is always bounded by the actual search time taken by the LLS and UL scheme respectively.

TABLE 1
*Complexity for List Creation Schemes*

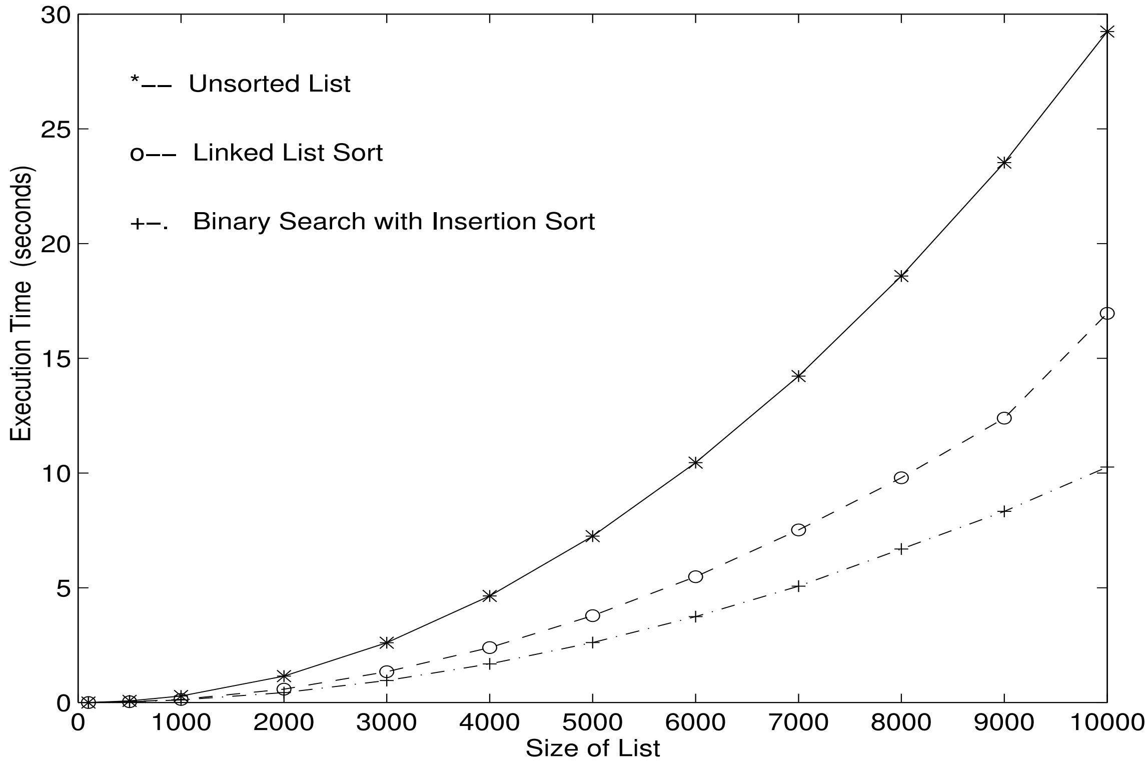
| Algorithm | Complexity | | |
|---|---|---|---|
| | Search | Insertion | Overall |
| UL | $O(n^2)$ | $O(1)$ | $O(n^2)$ |
| BSIS | $O(n \log_2 n)$ | $O(n^2)$ | $O(n^2)$ |
| LLS | $O(n^2)$ | $O(1)$ | $O(n^2)$ |

The schemes were tested on large lists of up to 10,000 numbers and the results are summarized in Figure 1, where each data point represents an average of 10 executions whose execution times were measured in seconds. The computer used for these simulations was a Sun SPARCserver 1000 workstation running Solaris 2.4 as the operating system. The figure confirms that maintaining a sorted list with BSIS and LLS does improve upon ADBASE's method of maintaining an unsorted list of numbers. As predicted by the analysis, the UL scheme with an unsorted list of numbers was the slowest while BSIS gave the fastest times.

These results indicate that BSIS is the most suitable scheme of those examined as a bookkeeping strategy for ADBASE, especially when the nature of MOLPs and ADBASE's technique are considered. The reason is that the solution graph of an MOLP is often cyclic and ADBASE searches the graph dynamically. While searching for all EEPs, previously examined EEPs can be encountered again. Thus, the searching of the list of EEPs becomes most significant while the insertion is less important. BSIS is clearly the fastest search technique. The computational results of Figure 1 also justify the use of this scheme especially in the case of large lists of numbers, where it takes approximately one third of the time taken by the UL scheme which simulates ADBASE's bookkeeping technique. Thus, the BSIS scheme was chosen to be implemented into ADBASE and Parallel ADBASE.

**3. SEQUENTIAL ADBASE.** In this section we present the results of implementing the new bookkeeping strategy of the BSIS scheme into the sequential ADBASE. In ADBASE, an efficient basis and an EEP are first coded and then stored in a vector whose each component represents up to 30 variables. While implementing the BSIS scheme into ADBASE, it was observed that sorting a list of vectors took a longer time than sorting a list of scalars. This led to the formation and use of an index table such

FIG. 1. *Times for Three List Creation Schemes*




that the coded efficient bases and EEPs stored in vector format were unchanged while their sorted order was recorded in the index table. Thus, whenever a new basis was generated, it was compared to the other bases in LISTB by applying the BSIS scheme through the index table. If the basis was not found in LISTB, it was stored at the bottom of the list and its sorted position was inserted into the index table. The ADBASE equipped with BSIS through this index table is referred to as *ADBASE-BSIS*.

Since the primary goal of this work was to improve the bookkeeping strategy, computational results for the times spent on bookkeeping were compared between the original ADBASE and ADBASE-BSIS. Moreover, it was necessary to determine the percentage of the total time spent on bookkeeping. As all bookkeeping in ADBASE occurs in the subroutine CODE, it was convenient to time each call to this subroutine in order to get the approximate time spent on bookkeeping. The total time spent in the subroutine CODE in comparison to the total time for the whole program is an approximate indicator of the percent of time spent on bookkeeping.

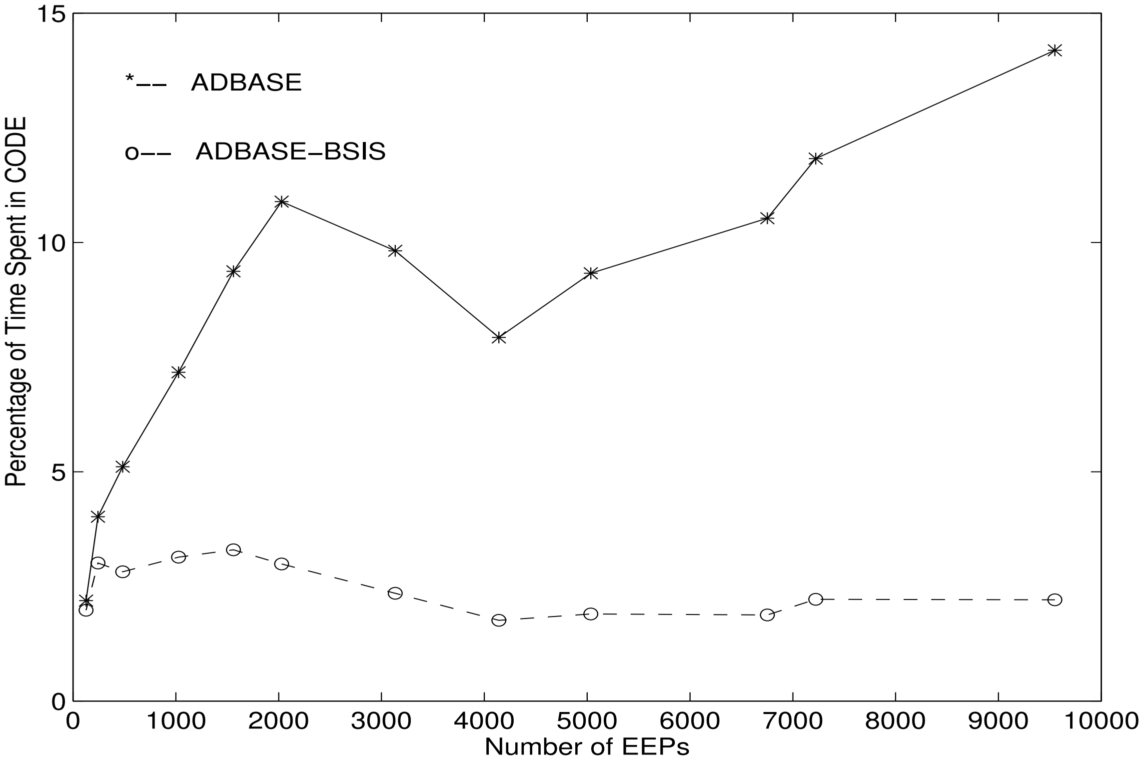
Table 2 and Figure 2 give the execution times of ADBASE and ADBASE-BSIS for different size MOLPs generated by the ADBASE random problem generator [13] [14]. From these results, the following is observed:

1. For ADBASE, as the size of the problem increases, the percentage of time spent in the subroutine CODE increases. This result justifies the primary focus of this work: to find a more efficient method to do the bookkeeping.
2. The results obtained by ADBASE-BSIS give a very significant improvement.

TABLE 2
*Execution Times for Subroutine CODE (secs)*

| Number of EEPs | CODE/ Total | |
|---|---|---|
| | ADBASE | ADBASE-BSIS |
| 126 | 0.06 / 2.88 | 0.05 / 2.69 |
| 243 | 0.19 / 4.81 | 0.14 / 4.52 |
| 483 | 0.63 / 12.38 | 0.32 / 11.51 |
| 1027 | 2.48 / 34.53 | 0.99 / 31.55 |
| 1561 | 5.54 / 59.11 | 1.72 / 52.23 |
| 2029 | 8.89 / 81.58 | 2.08 / 69.38 |
| 3133 | 21.62 / 220.00 | 4.45 / 189.22 |
| 4142 | 36.48 / 460.32 | 7.11 / 404.67 |
| 5035 | 54.74 / 586.66 | 9.68 / 509.47 |
| 6752 | 94.35 / 895.96 | 14.47 / 769.74 |
| 7222 | 108.98 / 921.42 | 17.37 / 783.32 |
| 9549 | 185.84 / 1309.98 | 23.86 / 1079.35 |

FIG. 2. *Percent of Total Time Spent in CODE*




As the size of the problem increases, the percentage of time spent in CODE remains roughly unchanged and is a small portion of the total computation. For very large problems, the bookkeeping with the BSIS scheme takes only 2% of the total computational time while it takes about 14% for the original ADBASE. This is a very good improvement.

The list creation scheme that occurs in the subroutine CODE is the dominating part of the ADBASE program that cannot be efficiently divided into independent subtasks. Therefore, the improvement in the bookkeeping by the BSIS scheme becomes even more significant when MOLPs are solved on multicomputers.

**4. PARALLEL ADBASE.** The new list creation scheme was incorporated into Parallel ADBASE which had been developed and implemented by Wiecek and Zhang [17]. The computer program for the original Parallel ADBASE and the one for Parallel ADBASE with the BSIS list creation scheme will be referred to as *PADBASE* and *PADBASE-BSIS*, respectively.

In developing a parallel algorithm, one tries to achieve the shortest execution time by dividing the problem into several sub-tasks to be executed concurrently on multicomputers [10] [11]. The computational time of any program can be split into a global time $T_g$, that cannot be distributed among processors, and a local time $T_l$, that can be executed simultaneously on all processors. Thus, the total execution time using $p$ processors, $T_p$, can be represented as:

$$T_p = T_g + \frac{T_l}{p}.$$

For a particular problem, the improvement of PADBASE-BSIS over PADBASE will be due to the improvement in bookkeeping, which reduces $T_g$. And, as the number of processors $p$ increases, the global time, $T_g$, becomes the dominating factor of the total computation time. Reducing $T_g$ is therefore likely to be significant for the parallel implementation of ADBASE.

In order to compare the parallel performance of PADBASE and PADBASE-BSIS, the percentage savings of the new bookkeeping scheme over the previous method is used, which is defined as:

$$\text{(1)} \qquad \textit{Percent Savings} = \frac{T_p \text{ for PADBASE } - T_p \text{ for PADBASE-BSIS}}{T_p \text{ for PADBASE}}.$$

Since PADBASE and PADBASE-BSIS differ only by their bookkeeping strategies which take place in global computations, the percent savings (1) can be also expressed as

$$\text{(2)} \qquad \textit{Percent Savings} = 1 - \frac{T_g \text{ for PADBASE-BSIS } + T_l/p}{T_g \text{ for PADBASE } + T_l/p}.$$

When MOLPs are well-scaled with the number of processors used, i.e., $T_l/p$ remains constant as the size of the MOLPs and the number of processors increase simultaneously, (2) implies that the percent savings of PADBASE-BSIS over PADBASE is inversely proportional to the ratio of $T_g$'s for PADBASE-BSIS and PADBASE. Since $T_g$ is approximately equal to the time spent in CODE, and the time spent in CODE increases as the size of MOLPs becomes larger (see Figure 2), the percent savings of PADBASE-BSIS over PADBASE are expected to increase with the size of MOLPs.

Using the data obtained from the sequential implementation of ADBASE, one can predict the parallel performance of PADBASE and PADBASE-BSIS, and further quantify the above discussion. For example, assume PADBASE takes $T_p = 100$ seconds to solve a given MOLP on $p = 1$ processor. Assume the global bookkeeping takes $T_g = 10$ seconds for PADBASE and $T_g = 2$ seconds for PADBASE-BSIS; that is to say, times that approximate the average times spent in the subroutine CODE as seen in Figure 2. The parallel execution times can be predicted for this hypothetical MOLP as solved on a multicomputer with $p$ processors. Table 3 lists, for this example, the predicted values, the percent savings expected from PADBASE-BSIS instead of PADBASE, and ratios representing the predicted percentage of time spent on bookkeeping.

TABLE 3
*Predicted Parallel Performance*

| $p$ | Execution Time $T_p$ (seconds) | | Percent Savings (%) | $T_g/T_p$ (%) | |
|---|---|---|---|---|---|
| | *PADBASE* | *PADBASE −BSIS* | | *PADBASE* | *PADBASE −BSIS* |
| 1 | 100.00 | 92.00 | 8.0 | 10.00 | 2.17 |
| 2 | 55.00 | 47.00 | 14.5 | 18.18 | 4.26 |
| 4 | 32.50 | 24.50 | 24.6 | 30.77 | 8.16 |
| 8 | 21.25 | 13.25 | 37.7 | 47.06 | 15.09 |

The last two columns in Table 3, corresponding to the percentage of time spent on bookkeeping in PADBASE and PADBASE-BSIS respectively, indicate that the bookkeeping, which was a relatively small global operation, 10.00 % for PADBASE and 2.17 % for PADBASE-BSIS when the programs were sequentially executed, could dominate the computational time as the number of processors increases. Comparing these two columns, we anticipate that the new list creation scheme will significantly reduce the percent of total time spent on bookkeeping. This observation further justifies the need to reduce the global time by using the new list creation scheme, especially in the case when MOLPs are solved on advanced parallel computers.

Parallel ADBASE was run on an Intel Paragon XPS/4, an MIMD computer, located at the South Carolina Supercomputing Network Facilities and Services. All the runs were done on a fixed compute partition of eight nodes. Care was taken to ensure that runs with the same number of processors were always done on the same node partition. This precaution helped eliminate the possible instabilities of the Paragon in choosing the nodes on which the programs were executed.

PADBASE and PADBASE-BSIS were tested on the same test problems used on sequential ADBASE. Table 4 gives the execution times of the two programs and percent savings of PADBASE-BSIS over PADBASE, while Figure 3 gives the percent savings for the different problems on the different numbers of processors. In all instances, the new bookkeeping scheme improves upon the previous one. For each of the MOLPs considered, the percent savings increase as more processors are used.

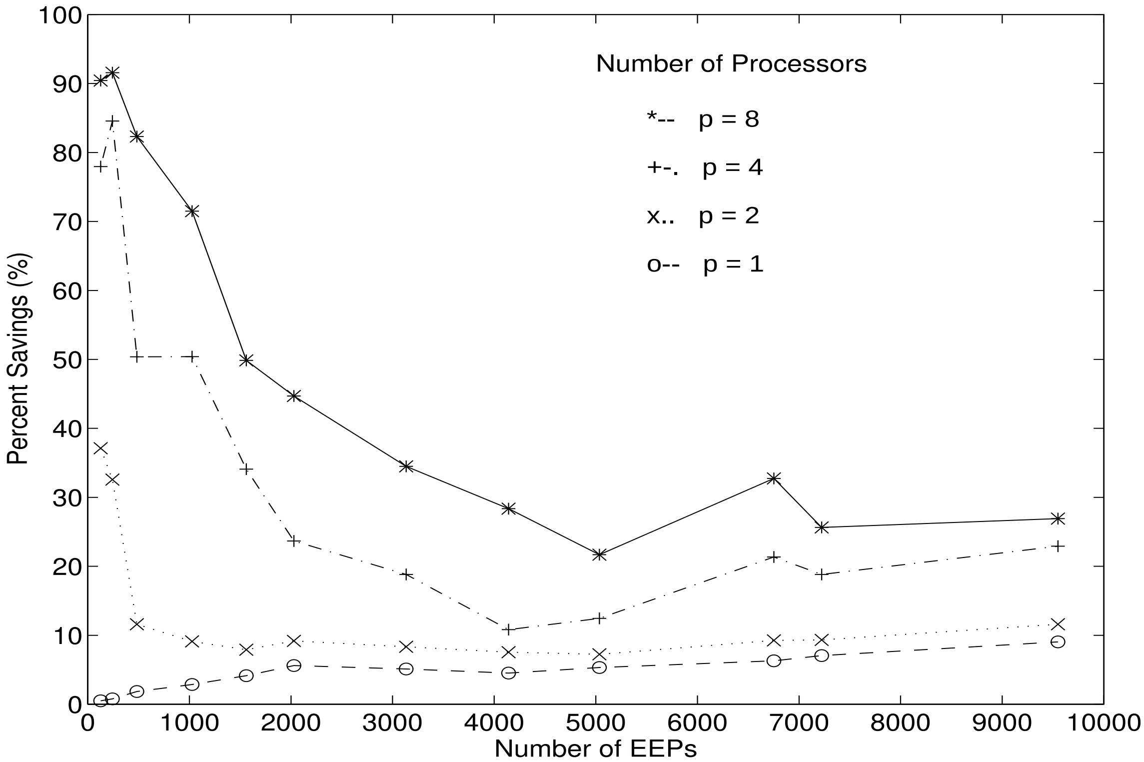
It may be noted from Table 4 that for all but the few largest problems, the execution times increase (for both PADBASE and PADBASE-BSIS) in going from four to eight processors, and the behavior of the percentage improvement of PADBASE-BSIS over PADBASE as the problem size increases seems inconsistent with the expectation. Previous research has revealed that as the number of processors doubles, in order to

TABLE 4
*Parallel Performance*

| Number of EEPs | $p$ | Execution Time (seconds) *PADBASE* | Execution Time (seconds) *PADBASE−BSIS* | Percent Savings (%) |
|---|---|---|---|---|
| 126 | 1 | 2.0 | 1.9 | 0.5 |
| | 2 | 1.8 | 1.1 | 37.1 |
| | 4 | 3.3 | 0.7 | 78.0 |
| | 8 | 7.9 | 0.8 | 90.4 |
| 243 | 1 | 3.4 | 3.4 | 0.9 |
| | 2 | 2.8 | 1.9 | 32.6 |
| | 4 | 7.6 | 1.2 | 84.6 |
| | 8 | 14.1 | 1.2 | 91.6 |
| 483 | 1 | 8.8 | 8.7 | 1.8 |
| | 2 | 5.3 | 4.6 | 11.6 |
| | 4 | 5.4 | 2.7 | 50.4 |
| | 8 | 15.3 | 2.7 | 82.3 |
| 1027 | 1 | 24.8 | 24.1 | 2.9 |
| | 2 | 13.8 | 12.5 | 9.1 |
| | 4 | 13.6 | 6.7 | 50.4 |
| | 8 | 24.0 | 6.8 | 71.5 |
| 1561 | 1 | 42.0 | 40.2 | 4.1 |
| | 2 | 22.1 | 20.4 | 7.9 |
| | 4 | 16.8 | 11.1 | 34.1 |
| | 8 | 22.4 | 11.2 | 49.9 |
| 2029 | 1 | 56.9 | 53.7 | 5.6 |
| | 2 | 30.2 | 27.5 | 9.2 |
| | 4 | 18.9 | 14.4 | 23.7 |
| | 8 | 25.1 | 13.9 | 44.7 |
| 3133 | 1 | 153.4 | 145.5 | 5.1 |
| | 2 | 80.9 | 74.2 | 8.3 |
| | 4 | 48.1 | 39.0 | 18.9 |
| | 8 | 58.5 | 38.3 | 34.5 |
| 4142 | 1 | 321.7 | 307.1 | 4.5 |
| | 2 | 167.6 | 155.0 | 7.5 |
| | 4 | 88.6 | 79.0 | 10.8 |
| | 8 | 109.5 | 78.4 | 28.4 |
| 5035 | 1 | 405.5 | 383.9 | 5.3 |
| | 2 | 205.5 | 190.6 | 7.3 |
| | 4 | 111.0 | 97.2 | 12.4 |
| | 8 | 124.5 | 97.5 | 21.7 |
| 6752 | 1 | 626.3 | 586.8 | 6.3 |
| | 2 | 321.0 | 291.2 | 9.3 |
| | 4 | 193.9 | 152.5 | 21.4 |
| | 8 | 184.9 | 124.3 | 32.8 |
| 7222 | 1 | 636.0 | 591.2 | 7.0 |
| | 2 | 323.2 | 293.1 | 9.3 |
| | 4 | 188.6 | 153.1 | 18.8 |
| | 8 | 182.1 | 135.4 | 25.6 |
| 9549 | 1 | 890.9 | 810.3 | 9.0 |
| | 2 | 456.5 | 403.5 | 11.6 |
| | 4 | 276.1 | 212.8 | 22.9 |
| | 8 | 237.1 | 173.2 | 27.0 |

maintain the efficiency and the scalability of PADBASE, the size of an MOLP (measured by its number of EEPs) needs to be increased by at least a factor of ten [17]. This is largely due to the nature of MOLPs whose inherent cyclic solution graphs prevent the adoption of local bookkeeping strategies that have been effectively used in tree search strategies for solving single objective linear programs [11]. The performance analysis and the predictions discussed above require that the problems be well-scaled, that is, MOLPs be of large size. The data collected in Table 4 show that for the three largest MOLPs, both PADBASE and PADBASE-BSIS start accelerating the solution process on eight processors. Figure 3 indicates that when the problem size is large enough, (e.g., when the number of EEPs is larger than 5,000), the percent savings on the different numbers of processors slowly increase and approach the predicted values given in Table 3, as the problem size grows larger. The asymptotic behavior of the percent savings for sufficiently large problems, as revealed by Figure 3, agrees with the predictions. This observation implies that the theoretical analysis matches the computational results for large size MOLPs.

FIG. 3. *Percent Savings of PADBASE-BSIS over PADBASE*




Overall, the new bookkeeping scheme provides marked savings for all problems, over the range of number of processors. As predicted, the percentage improvement increases as either the number of processors increases or the problem size grows when the MOLPs are well scaled with the number of processors. These results establish the significance of improving the bookkeeping task, a global operation, particularly for large problems solved on multicomputers.

**5. CONCLUSIONS.** The need for a new bookkeeping strategy for ADBASE and Parallel ADBASE was the main motivation for this research. This led to a study of three representative list creation schemes. The computational complexity study, the numerical results and the nature of MOLPs revealed that the Binary Search with Insertion Sort (BSIS) is the most suitable list creation scheme for incorporation into the bookkeeping of ADBASE. The BSIS scheme was implemented in ADBASE as well as in Parallel ADBASE. The scheme considerably reduced the time spent on the bookkeeping, a global operation for Parallel ADBASE. The analytic study and experimental results reported in this work suggest that the resulting bookkeeping strategy significantly improves upon the previous one, and has potential for being even more effective for solving very large MOLPs on multicomputers.

**Acknowledgments.** We would like to thank Thomas W. Crockett of the Institute for Computer Applications in Science and Engineering at NASA Langley Research Center for his help in our implementation of the parallel programs. We also would like to thank an anonymous referee for helpful comments that significantly improved the presentation of our paper.

REFERENCES

[1] P. Armand and C. Malivert, *Determination of the efficient set in multiobjective linear programming*, Journal of Optimization Theory and Applications, 70 (1991), pp. 467–490.

[2] C. S. Chang, *Co-ordinated static and dynamic monitoring and optimization of power systems using a parallel architecture and pattern recognition techniques*, IEE Proceedings - C, 139 (1992), pp. 197–204.

[3] V. Chankong and Y. Y. Haimes, *Multiobjective Decision Making - Theory and Methodology*, North-Holland, New York, 1983.

[4] J. P. Costa and J. N. Climaco, *A multiple reference point parallel approach in* MCDM, Proceedings of the Tenth International Conference on Multiple Criteria Decision Making, Taipei, 3 (1992), pp. 265–272.

[5] J. G. Ecker and I. A. Kouda, *Finding all efficient extreme points for linear multiple objective programs*, Mathematical Programming, 14 (1978), pp. 249–261.

[6] Y. Evtushenko, V. Mazourik, and V. Ratkin, *Multicriteria optimization in the* DISO *system*, Optimization, Parallel Processing and Application, eds: A. Kurzhanski, K. Neumann and D. Pallaschke, Springer-Verlag, Berlin, (1988), pp. 94–102.

[7] L. Fotso, *Multiple objective programming*, Ph.D. Dissertation, Operations Research and Statistics Interdisciplinary Program, Rensselaer Polytechnic Institute, Troy, New York, (1981).

[8] E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, W. H. Freeman and Company, New York, NY, 1993.

[9] H. Isermann and G. Naujoks, *Operating manual for the EFFACET multiple objective linear programming package*, Fakultaet fuer Wirtschaftswissenschaften, University of Bielefeld, Bielefeld, Germany, (1984).

[10] L. Kronsjo and D. Shumsheruddin, eds., *Advances in Parallel Algorithms*, John Wiley & Sons., New York, 1992.

[11] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing : Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[12] R. E. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*, John

Wiley and Sons, New York, 1986.

[13] ———, *Random problem generation and the computation of efficient extreme points in multiple objective linear programming*, Computational Optimization and Applications, 3 (1994), pp. 333–347.

[14] ———, *Manual for the ADBASE Multiple Objective Linear Programming Package*, Department of Science and Information Technology, University of Georgia, Athens, Georgia, 1995.

[15] L. W. G. Strijbosch, A. G. M. van Doorne, and W. J. Selen, *A simplified MOLP algorithm: the MOLP-S procedure*, Computers and Operations Research, 18 (1991), pp. 709–716.

[16] M. M. Wiecek and H. Zhang, *Solving multiple objective linear programs on the Intel Paragon*, Proceedings of Mardi Gras '94 Conference: Toward Teraflop Computing and New Grand Challenge Applications, Baton Rouge, Louisiana, February 10-12, (1994), pp. 323–329.

[17] ———, *A parallel algorithm for multiple objective linear programs*, Computational Optimization and Applications, 8 (1997).

[18] M. Zeleny, *Lecture Notes in Economics and Mathematical Systems : Linear Multiobjective Programming*, Springer-Verlag, New York, 1974.

[19] ———, *Multicriteria simplex method: a Fortran routine*, Lecture Notes in Economics and Mathematical Systems, 123, Springer-Verlag, Berlin, (1976), pp. 323–345.