Extensions of and Uses for the Differencing Algorithm for Number Partitioning

Robert H. Storer
Lehigh University

Report No. 99T-09

Extensions of and Uses for the Differencing Algorithm for Number Partitioning

Robert H. Storer
Department of Industrial and Manufacturing Systems Engineering
The Manufacturing Logistics Institute
Harold S. Mohler Laboratory
200 West Packer Avenue
Lehigh University
Bethlehem, PA 18015-1582
rhs2@Lehigh.edu

The Number Partitioning problem entails dividing a list of numbers into two sets so as to minimize the absolute difference of the sums of numbers in each. Number Partitioning is one of the six basic NP-complete problems identified by Garey and Johnson (1967), and is often used to prove other problems difficult by reduction. Few if any other uses for Number Partitioning have been reported. In this paper we present uses for, and extensions of the basic differencing algorithm (Karmarkar and Karp, 1982) for Number Partitioning. First we demonstrate that randomized versions of the differencing algorithm can be very effective for number partitioning. Then we employ these algorithms to build more complex heuristics for problems with embedded number partitioning. Specifically we address: 1) the problem of partitioning items into multiple groups (also known as "Multi-Processor" or "Parallel Machine" scheduling), 2) partitioning problems with certain cardinality constraints and subset sum problems, 3) stochastic number partitioning, and 4) balancing weights around axes of symmetry (including the well-studied turbine rotor blade balancing problem).

Number partitioning is a well known, fundamental combinatorial optimization problem. Indeed it is classified as one of the six basic NP-complete problems by Garey and Johnson (1979), and has been used extensively to prove other problems NP-complete through reduction. Johnson et al. (1991) rekindled interest in number partitioning in one of a series of three papers. These papers tested simulated annealing algorithms on a diverse set of problems one of which was number partitioning. Their results showed that long runs of carefully devised simulated annealing algorithms could rarely do better than the simple differencing algorithm of Karmarkar and Karp (1982). Two conclusions stood out from the paper of Johnson et al.:

1) The differencing algorithm obtains *miniscule objective function values* in O(NlogN) time.
2) The paper's statement that there were *no known applications for number partitioning* with the possible exception of coding theory.

The power of this simple algorithm led us to explore its use as an integral part of heuristics for solving more complex problems.

The number partitioning problem is particularly easy to state; partition a set of numbers into two mutually exclusive sets minimizing the absolute difference of the sum of the two sets. Equivalently, we seek to minimize the maximum (over the two sets) of the set sums. Letting a_i for i=1,...,N represent N numbers to be partitioned, and S and S′ represent the two sets after partitioning, then the problem can be stated as:

$$\text{Min} \left| \sum_{i \in S'} a_i - \sum_{i \in S} a_i \right|$$

There is no restriction on the cardinality of S and S′ so that more numbers may be in one set than in the other.

The number partitioning problem has several properties worthy of note. First, standard integer programming approaches appear completely unsuited for this problem. Indeed even relatively small instances will cause difficulty for algorithms such as CPlex 6.5 as we have verified experimentally. One can attempt to explain the reason for this as follows. Define T as the sum of the numbers:

$$T = \sum_{i=1}^{n} a_i$$

Then the number partitioning problem can be formulated as a zero-one integer program by recognizing that the sum of one of the two sets must be greater than or equal to T/2, leading to the following formulation:

$$\text{Min} \sum_{i=1}^{n} a_i(0.5-x_i)$$

subject to:

$$\sum_{i=1}^{n} a_i(0.5-x_i) \geq 0$$

$$x_i \in \{0,1\}$$

To providing insight into the difficulty of this problem we attempt to apply Lagrangian relaxation on the first constraint. The relaxed problem is:

$$\text{Min } (1 - \lambda) \sum_{i=1}^{n} a_i(0.5-x_i)$$

subject to:

$$x_i \in \{0,1\}$$

For $\lambda > 1$ the solution to the relaxed problem sets all $x_i = 1$, for $\lambda < 1$ all $x_i = 0$, while if $\lambda=1$, *all* solutions have objective function zero. Thus for any value of λ except 0, the solution to the relaxed problem actually *maximizes* the difference in the sums of the two partitioned sets. When $\lambda=0$, any solution is optimal for the relaxed problem. The point of this exercise is to show that there is no useful information provided by duality theory, and thus standard math programming approaches are ineffective. While we do not preclude the possibility that a more clever formulation or approach may work well, we believe it to be unlikely.

To illustrate the ineffectiveness of standard integer programming methods, we generated a problem with N=100 numbers randomly distributed as *IntegerUniform*(1,999999999), and used Cplex 6.5 to solve it. The highlights of this exercise appear in Table I. We were actually surprised that Cplex was able to solve this problem, although it took over 64 million branch and bound nodes and several hours on a PC running at 400 MHz.

(Table I here)

A second interesting property is that commonly used and powerful local search heuristics such as simulated annealing and tabu search, which are based on interchange neighborhoods, are also completely unsuitable for number partitioning. This fact was demonstrated quite clearly in Johnson et al. (1991). An interesting conclusion of the paper was that, despite a quite sophisticated implementation of simulated annealing, and extensive computation time, the algorithm performed quite poorly relative to the performance of the simple differencing algorithm of Karmarkar and Karp (1982). Johnson et al. (1991) point out that any local search algorithm based on neighborhoods defined by swapping elements between sets can be expected to perform poorly due to the "mountainous" topology of the objective function over this neighborhood. Thus the two most powerful and commonly used approaches for combinatorial optimization problems (integer programming and local search) both appear unsuited for number partitioning.

A third interesting property that seems not to have been widely recognized is that, in *some sense*, number partitioning problems are easy. It is of course necessary for us to better define and then demonstrate what we mean by this point; one of the key points of this paper. The basis for this statement is that there are very effective constructive heuristics for number partitioning beginning with the Karmarkar-Karp differencing algorithm. In addition, researchers have recently proposed even more powerful heuristics which are all essentially randomized versions of the differencing algorithm. These algorithms can solve many number partitioning problems to optimality with high probability, a claim we justify empirically subsequently. Further, the larger the problem instance in terms of N, the easier it apparently is to find optimal solutions. (However, as the number of digits in each number increases, number partitioning problems become increasingly difficult). These effective algorithms can be used within more complex algorithms for solving problems with "embedded" number partitioning problems as we will demonstrate.

2.0 Background

The basis for all good known heuristics for number partitioning is the simple and elegant single pass "differencing algorithm" proposed by Karmarkar and Karp (1982). The basic differencing algorithm as described in Johnson et al. (1991) is as follows:

Differencing Algorithm

```
0.    Create N nodes each labeled with one of the numbers $a_i$.
      Place all nodes in the set "live".

1.    Let u and v be the nodes in "live" with the largest and second largest labels
      respectively.
            Add an arc between u and v.
            Remove v from the set "live".
            Relabel u with $a_u - a_v$.

Repeat step 1. N-1 times (until one node remains in "live").

2.    The resulting graph will form a tree. Two color the tree to determine the
      partition.
```

The label of the last remaining live node at the end of step 1 will be the objective function value of the partition (or "partition value"). At each iteration of step 1, placing an arc between nodes u and v puts them on opposite sides of the partition, although *which* side remains to be determined later once the entire tree is constructed. Also note that two coloring a tree is a trivial exercise.

2.1 Extensions to the Differencing Algorithm

Extensions to the basic differencing algorithm have been proposed by Storer, Flanders, and Wu (1996), and by Arguello, Feo, and Goldschmidt (1996). The algorithms in these papers are similar, and are essentially randomized versions of the differencing algorithm. In this section we review the algorithm proposed by Storer, Flanders, and Wu (1996), propose a modification, and show empirical results on algorithm performance. These empirical results are designed to substantiate the claim that number partitioning is in some sense "easy".

To describe the algorithm proposed in Storer, Flanders, and Wu (1996) we first present an alternative description of the differencing algorithm:

(Algorithm A1)

```
0.    Place the numbers in a list sorted from largest to smallest.

1.    Let u and v be the top two numbers on the list.
      Remove u and v from the list. Calculate a new number $|u-v|$.

2.    Merge $|u-v|$ back into the list as follows:
      Starting at the bottom of the sorted list of numbers $a_{[i]}$ and going up, compare $|u-v|$ to $a_{[i]}$
      until index d is found such that $|u-v| \le a_{[d]}$
```

Merge |u-v| just below $a_{[d]}$ in the list.

Repeat steps 1 and 2 until one number remains on the list.

The merge operation of step 2 has complexity on the order of O(N). Since it is repeated N times, the overall complexity of the algorithm is $O(N^2)$. Algorithm A1 will determine the objective function value, but additional computation is required to determine the actual partition (one would keep track of the tree, and perform the 2-coloring).

We have found that a small improvement in performance of the merge operation of step 2 is possible by starting at the bottom of the list and going up rather than going down from the top. Presumably a more sophisticated merge procedure could reduce the complexity even further. However, in order to develop a randomized version of the algorithm, we rely on this naive merge operation.

In Storer, Flanders, and Wu (1996) algorithm A1 was randomized by first scrambling the sorted list a bit. It was implemented as follows:

(Algorithm A2)

```
1.    Create "perturbed numbers" $b_i = a_i + u_i$
      where the $u_i$ are Uniform$(0, \theta)$ deviates and $\theta$ is a tuning parameter

2.    Sort the list from largest to smallest according to the $b_i$ values.
      (Note that the list still contains $a_i$ values. The $b_i$ are used only in the sorting of the list).

3.    Apply steps 1 and 2 of algorithm A1 to the resulting list, and observe the objective
      function value.

Repeat steps 1 through 3 many times, and report the best objective function value found.
```

Algorithm A2 does not explicitly generate the partition (i.e. assign each number to one of the two sets). To accomplish this, one would need to remember the random number seed used to generate the b_i values which lead to the best solution found. One can then reapply algorithm A2 while at the same time building the tree, then apply the two coloring step.

A slight modification to algorithm A2 can improve the run time while maintaining the same level of performance. In algorithm A2, the sortation order is scrambled slightly in order to "trick" the differencing algorithm into generating alternative solutions. We have found that it is not necessary to scramble the entire list to accomplish the desired effect. Instead we scramble only the 20 largest elements of the sorted list. In cases where N<20, we scramble all elements. This modification eliminates much of the sortation effort (subsequent to the first iteration when the original numbers must be sorted). This modification does not seem to diminish algorithm performance and leads to:

(Algorithm A3)

```
0.      Sort the numbers $a_i$ from largest to smallest.

For iterations = 1 to (say)1000

    1.  Create "perturbed numbers" for the first 20 numbers on the list:
        $b_i = a_i + u_i$ for i=1 to 20
        where the $u_i$ are Uniform(0 , $\theta$) deviates and $\theta$ is a tuning parameter

    2.  Sort the first 20 numbers from largest to smallest according to the $b_i$ values.
          (Note that the list still contains $a_i$ values. The $b_i$ are used only in the sorting of the first
          20 elements of the list).

    3.  Apply steps 1 and 2 of algorithm A1 to the resulting list, and observe the objective
        function value.

    4.  If the solution is the best yet, save it, and the seed used to generate the $b_i$ values

Next iteration

5.      Generate the best partition given the seed saved in step 4.
```

In the description above we have specified 1000 iterations. This seems to be a reasonable number that gives good results. Clearly better results are possible with more iterations, but the law of diminishing returns seems to take effect around 1000 iterations.

2.2 Preliminary Computational Testing

In this section we present a brief empirical summary of the performance of algorithm A3. In the first experiment we investigate the effects of the tuning parameter θ while in the second experiment we demonstrate the power of the algorithm.

In the first experiment, we generated uniformly distributed integers in the range 1 to 999,999,999. We generated 1000 test problems each for problems of size N=10, 20, 30, 40, 60, 80, and 100. We then ran

Algorithm A3 with varying values of the tuning parameter θ. In this experiment, θ is defined as a proportion of the range of the data. Thus θ = 0.04 indicates that the b_i are distributed *IntegerUniform*[0 , 40000000]. The results of this experiment are shown in Figure 1.

(Figure 1. goes here)

The results show that there is an optimum value of θ which varies as a function of problem size. However, the results also show that algorithm performance is quite insensitive to θ as long as θ is large enough. If θ is too small, algorithm A3 will generate only a small number of solutions repeatedly thus degrading performance. As long as θ is large enough to scramble the first 20 numbers sufficiently, algorithm performance will be consistent over a wide range. As a result, and to keep things simple, θ = 0.04 * Range is used in subsequent experiments.

Figure 2 shows the performance of the differencing algorithm and of algorithm A3 as a function of N. In figure 2 we plot the partition value divided by the range of the numbers in the problem. Each plotted point represents the average over 1000 randomly generated problems with 9 digits (i.e. IntegerUniform[1,999999999]).

(Figure 2 goes here)

Figure 2 shows that algorithm A3 provides roughly 3 orders of magnitude improvement over the differencing algorithm for problems with 100 numbers, and that *extremely* small partition values, relative to the range of the numbers, are easily achieved.

The second experiment was designed to indicate the performance achievable from the differencing algorithm and from the randomized extension. Test problems were generated with N=10, 20, 30, 40, 60, 80, and 100 numbers, and D=2, 3, 4,...., 9 digits. Thus for D=4, the numbers are IntegerUniform[1 , 9999] and θ = 0.04*10000. In addition, two methods were used to generate the numbers. Method 1 simply generates the numbers randomly. For problems generated randomly, it is difficult to know the optimal solution due to problem complexity. However it can be said that if a solution is found with a partition value of 0 or 1, then that solution must be optimal. The second problem generation method (Method 2) generates the numbers in such a way that the optimal solution is known to be 0. The first N-1 numbers are generated randomly while the last number is generated so as to guarantee that a partition exists with value 0. Two sums are initialized to zero. As each of the N-1 numbers is generated, it is added to the smaller of the two sums. Then the last (Nth) number is set equal to the absolute difference of the two sums. For each problem generation method, and for each DxN combination, 1000 problems were generated. Tables II and III show the performance of algorithm A3 and the differencing algorithm (respectively) on randomly generated problems. Tables IV and V show the performance of algorithm A3 and the differencing algorithm (respectively) on problems with known optimal solution. Values in the tables show the percentage of the 1000 problems for which the optimal solution was found.

(Tables II to V go here)

For problems of size N=10, the relatively low percentages in Tables II and III do not mean that the optimal was rarely found, but rather that the optimal is rarely 0 or 1. For problems of size 100 or larger we see that the optimal solution value of randomly generated problems is invariably equal to 0 or 1.

The basic conclusion one draws from the tests is that the number partitioning problem is, in some sense, easy. Randomlu generated problems with 5 or fewer significant digits can be solved optimally with high probability. Problems with 9 significant digits can be solved optimally when the problem is small (N<20) or large (N>250). Intermediate problems with many digits cannot be solved optimally. However, the objective function values are extremely small relative to the magnitudes of the numbers to be partitioned. It is also worth pointing out that both the differencing algorithm and algorithm A3 run extremely quickly so that computational effort is rarely an issue.

Most practical problems rarely have more than 5 significant digits of accuracy in the problem data. Certainly 9 digits is enough to capture what is known about a real problem. Thus we conclude that for practical problems, the number partitioning problem is *essentially easy*. For more complex problems which have "embedded number partitioning" our approach is to embed the above algorithms within more complex heuristics. In the rest of the paper we develop algorithms for some well-known problems using this approach.

3.0 Number Partitioning Extensions

In this section we briefly illustrate how one can develop heuristics for 3 extensions of the basic number partitioning problem; 1) fixed cardinality number partitioning, 2) the subset sum problem, and 3) stochastic number partitioning.

3.1 Fixed Cardinality Number Partitioning

In number partitioning there is no restriction on the size of the two sets of numbers after partitioning. Suppose one desired an equal number of numbers in each of the two sets (assuming N is even). This can be achieved by:

1) sorting the numbers,
2) taking the difference of each pair of consecutive numbers, and
3) solving the resultant number partition problem (which has N/2 numbers).

This is equivalent to joining each pair of consecutive (in the sorted list) numbers with an arc, thereby placing them on opposite sides of the partition. This guarantees an equal number in each set in the final solution. Interestingly, the objective function values degrade very little when equal cardinality is enforced (as we have verified empirically).

Similar tricks can be used to enforce other cardinality requirements (e.g. 40 numbers on one side, 60 on the other). For example to obtain a 60-40 split of N=100 numbers, replace 20 randomly selected numbers with their sum, then apply the equal cardinality number partitioning algorithm.

3.2 Subset Sum Problems

One may use number partitioning algorithms to solve the subset sum problem which seeks a subset of the numbers who's sum is as close a possible to a target value (perhaps without exceeding the target value). Suppose a given set of numbers sums to T=400 and our desired subset sum target is B=80. We create a new number, $a_{N+1} = 240$, and add it to the list. The total sum is now 400+240=640. After applying a number partitioning algorithm, the two partitions will sum to approximately 640/2 = 320 on either side. One of the sides will contain the added number $a_{N+1} = 240$. The remaining numbers on this side will sum to approximately 320-240 = 80 and is thus the desired subset.

In general if the numbers sum to T and the subset sum target is B, then the number we add to the problem is $a_{N+1} = |T-2B|$. When $0 < B < T/2$ the desired subset will be on the same side of the partition as a_{N+1}. When $T/2 < B < T$, the desired subset will be on the opposite side from a_{N+1}. We envision running a randomized algorithm such as A3. In this case roughly half of the solutions generated should yield a subset sum less than the target (for problems requiring the sum to be less than or equal to the target).

To illustrate how equal cardinality and subset sum algorithms might be used in practice, we refer to the recent article by Chu and Antonio (1999) on cutting stock problems in the steel rod industry. In this problem, rods exist in inventory that must be cut up to fulfil ordered rods of various sizes. An important sub-problem in their algorithm is to select exactly b rods from the set of orders to be cut from an inventory rod of length L. Thus they seek a subset of specified cardinality (b) and specified sum (L).

3.3 Stochastic Number Partitioning

In this section we demonstrate that a stochastic version of the number partitioning problem can be solved by solving a deterministic version of the problem. We require an assumption that the Central Limit Theorem applies to the sums over each set in the partition. The stochastic problem that we solve is described as follows:

Let a_i , $i=1,\ldots,N$ be independent random variables with $E(a_i) = \mu_i$ and $V(a_i) = \sigma_i^2$.

We seek to partition the a_i into two sets denoted by A and B so as to minimize the expected value of the maximum of the set sums.

$$\text{Let } S_A = \sum_{i\in A} a_i \quad \text{and} \quad S_B = \sum_{i\in B} a_i$$

$$\text{Let } T = \sum_{i=1}^{N} a_i\ , \quad \text{let } E(T) = \mu_T = \sum_{i=1}^{N} \mu_i \quad \text{and let } V(T) = \sigma_T^2 = \sum_{i=1}^{N} \sigma_i^2$$

We seek to Minimize $E[\,\mathrm{Max}(S_A , S_B)]$. Naturally we assume that numbers must be assigned to partitions before the outcomes are observed.

Let $D = S_A - S_B$.

Lemma 1:

The partition which minimizes $E[|S_A - S_B|]$ is equivalent to the partition which minimizes $E[\text{Max}(S_A, S_B)]$

Proof:

$$2\text{Max}(S_A, S_B) = S_A + S_B + |S_A - S_B|$$

$$= 2S_A \text{ when } S_A \geq S_B$$

$$= 2S_B \text{ when } S_A < S_B$$

Thus $E[\text{Max}(S_A, S_B)] = \frac{1}{2}E[S_A + S_B + |S_A - S_B|]$

$$= \tfrac{1}{2}\mu_T + \tfrac{1}{2}E[|S_A - S_B|]$$

Theorem: When the random variables a_i are such that the Central Limit Theorem can be reasonably assumed to apply to S_A and S_B, then the stochastic number partitioning problem can be solved by solving the deterministic problem on the expected values $E(a_i)$.

Proof:

Let $\mu_A = E[\sum_{i \in A} a_i]$ and let $\mu_B = E[\sum_{i \in B} a_i]$

Then $D \sim N(\mu_A - \mu_B, \sigma_T^2)$

The random variable $|D|$ is distributed according to the "Folded Normal". As discussed in Leone, Nelson, and Nottingham (1961), the expectation of $|D|$ is given by:

$$\mu_{|D|} = \sqrt{2/\pi}\,\sigma_T\, e^{-(\mu_A - \mu_B)^2/2\sigma_T^2} + (\mu_A - \mu_B)[1 - 2F(-(\mu_A - \mu_B)/\sigma_T)]$$

where $F(a) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-t^2/2}\, dt$ is the standard Normal cumulative distributi on function

$\mu_{|D|}$ is an increasing function in $|\mu_A - \mu_B|$ as can be seen by taking the derivative which can be shown to be:

$$\frac{d\mu_{|D|}}{d\mu} = 1 - 2F\left(\frac{-(\mu_A - \mu_B)}{\sigma}\right)$$

We see that the derivative is positive for positive $(\mu_A - \mu_B)$, and negative for negative $(\mu_A - \mu_B)$. That is, $\mu_{|D|}$ increases in $|\mu_A - \mu_B|$. This in turn implies that the partition minimizing $|\mu_A - \mu_B|$ also minimizes $E[Max(S_A, S_B)]$. Note that minimizing $|\mu_A - \mu_B|$ is just the deterministic number partitioning problem solved using $E(a_i)$ as data.

The Central Limit Theorem can be reasonably assumed when the a_i are independent, the variance of the sums S_A and S_B are not dominated by one, or a few of the random variables a_i, and the cardinality of the sums is large enough. For example if the a_i are iid and if the cardinality of each sum is more than 12, the Central Limit Theorem assumption is usually considered reasonable. Conversely when the Central Limit Theorem does not apply, solutions to the deterministic and stochastic problem may differ. This may be proven with the following counter example:

Let a_1 and a_2 both have the distribution: $P(X=0) = 2/3$, $P(X=3) = 1/3$ with $E(X)=1$

Let $a_3 = 0.99$ and $a_4 = 1.01$ be constants. Assume all a_i independent.

Case 1: The Partition $A = \{a_1, a_2\}$, $B = \{a_3, a_4\}$ leads to $|\mu_A - \mu_B| = 0$

Table VI. Calculations for number partitioning solution to example problem.

a_1	a_2	S_A	a_3	a_4	S_B	$Max(S_A,S_B)$	Prob
0	0	0	0.99	1.01	2	2	4/9
0	3	3	0.99	1.01	2	3	2/9
3	0	3	0.99	1.01	2	3	2/9
3	3	6	0.99	1.01	2	6	1/9

Let $Y = Max(S_A, S_B)$. Then $P(Y=2) = 4/9$, $P(Y=3) = 4/9$, $P(Y=6)=1/9$, and $E(Y) = 26/9$

Case 2: The Partition $A = \{a_1, a_3\}$, $B = \{a_2, a_4\}$ leads to $|\mu_A - \mu_B| = 0.02$, and

Table VII. Calculations for optimal solution to example problem.

a_1	a_3	S_A	a_2	a_4	S_B	$Max(S_A,S_B)$	Prob
0	0.99	0.99	0	1.01	1.01	1.01	4/9
0	0.99	0.99	3	1.01	4.01	4.01	2/9

3	0.99	3.99	0	1.01	1.01	3.99	2/9
3	0.99	3.99	3	1.01	4.01	4.01	1/9

P(Y=1.01)=4/9 P(Y=4.01)=3/9 P(Y=3.99)=2/9, and E(Y) = (24.05)/9

Thus the deterministic solution does not yield the optimum solution to the stochastic problem in this example.

While this example shows that solving the deterministic problem will not work in all cases, we believe that in many, if not most, real problems the Central Limit Theorem is a reasonable assumption. In closing we also note that this approach can be extended to multiple partitions (stochastic parallel machine scheduling) using the method of the next section.

4.0 Parallel Machine (Multi-Processor) Scheduling

Number partitioning algorithms may be used within a more general scheme to partition numbers into M groups rather than just two. This problem is generally known as "parallel machine scheduling", or "multi-processor scheduling". We have M identical machines, and N jobs indexed by j. Each job has a processing time p_j, and must be assigned to one of the machines. The goal is to assign jobs to machines so as to minimize the total completion time of all jobs (i.e. the makespan). When there are M=2 machines, this problem is precisely number partitioning.

Our number partitioning based algorithm begins with an initial solution generated by assigning jobs to the M machines using the well-known "Longest Processing Time First (LPT) algorithm" (Graham 1969). In the LPT algorithm, jobs are assigned to machines one at a time starting with the job with largest p_j and proceeding in sorted order. At each iteration, the next job is assigned to the machine with the least amount of work (smallest sum of p_j , ties being broken arbitrarily).

From this initial solution we find the machine with the largest sum and the machine with the smallest sum. Numbers (p_j) from these two machines are merged together to form a single set. In our implementation, the differencing algorithm is employed to "repartition" the numbers into two groups. This repartitioning step is applied repeatedly until an iteration occurs which does not improve the makespan.

The basic idea behind the algorithm is to reduce the largest sum at each iteration thus reducing the makespan.

This number partition based algorithm was tested against the LPT algorithm and the Multifit algorithm of Coffman, Garey and Johnson (1978). Multifit and LPT are probably the two best known algorithms for multiprocessor scheduling. Problems with N=50,75,100,125,150,175,200,225,250 jobs and M=5,10,15,20,25,30,35,40,45,50 machines were generated. All combinations of N and M were used except where N/M<2. For each N, M pair, 1000 problems were generated with random processing times distributed Uniform(0,1). Results of this experiment appear is figures 3 and 4. Performance is measured as percent above lower bound where the lower bound is given by the sum of the p_j divided by M. All three algorithms were coded by the author and run on a PC with a Pentium processor running at 266 MHz.

(Figures 3 and 4 go here)

The results show that the number partitioning based algorithm can achieve several orders of magnitude improvement in the percentage from lower bound for problems with relatively large N/M ratio. While this improvement comes with increased computation time, the number partition based algorithm is quite fast, never taking more than 0.02 seconds of elapsed time.

Finally, we note that the number partitioning based algorithm could be modified in several ways. First, if one desires less computation time, the algorithm can be run for fewer iterations. Instead of running until an iteration with no improvement is found, one could instead run k iterations where k is chosen to balance run time and performance. Figures 5 and 6 show results from running the number partition based algorithm for only 4 iterations on the same data sets used to generate figures 3 and 4. We see that the run times are reduced significantly, but that performance degrades as well. Never-the-less, the number partition based algorithm appears to be a viable competitor of multifit for problems with larger N/M ratios.

(Figures 5 and 6 go here)

A second alternative is to go in the other direction and achieve better performance with increased computation time. For example one could use a randomized version of the differencing algorithm such as algorithm A3 in place of the differencing algorithm. Further, if no improvement is found when applying a number partitioning algorithm to the pooled jobs from the "longest" and "shortest" machines, one could try pooling jobs from the longest and second shortest machines in an attempt to achieve further improvement.

We have implemented these ideas and found performance similar to that of the number partitioning algorithm in that extremely small objective function values can be obtained (of course at the expense of much greater computation). Details of these experiments may be found in the master's thesis of Snyder (1995).

5.0 Turbine Rotor Blade Balancing

The turbine rotor blade balancing problem has been studied by several authors including Amiouny, Bartholdi and Vande Vate (1997), Mason and Ronnqvist (1997), Laporte and Mecure (1988), Fathi and Ginjupalli (1993), and Mosevich (1986). Many authors use quadratic assignment problem models, and/or employ various versions of local search to find solutions, (the most recent such approach being Mason and Ronnqvist 1997). The most recent paper we found on turbine rotor blade balancing is Amiouny, Bartholdi and Vande Vate (1997) (forthcoming in *Operations Research*) who develop constructive heuristics for the problem. In this section we will develop number partitioning based heuristics and compare the results to those of Amiouny, Bartholdi and Vande Vate (1997).

The turbine rotor blade balancing problem simplifies to one of placing the fan blades (each with known mass w_i) at equally spaced intervals around a circle so as to make the center of mass as close to the center of the circle as possible. Amiouny, Bartholdi and Vande Vate (1997) propose several heuristics of which two seem to dominate, *ordinal pairing* and *greedy pairing*. Ordinal pairing requires very little computation time and produces good results while greedy pairing requires more computation, but gives the best performance of the heuristics developed. Both heuristics begin by sorting the blades from heaviest to lightest. Next consecutive pairs in the sorted list are placed across from each other on the circle. The two algorithms differ as to how the locations of each pair of blades are determined. Ordinal pairing places the blades in a fixed pattern. Letting 1 represent the heaviest blade, 2 the second heaviest, etc., the placement pattern for ordinal pairing is shown in figure 7a. In greedy pairing, the placement location is determined by a greedy algorithm. For each pair, all possible open positions on the circle are examined, and the position which yields the center of mass (of currently place blades) closest to the center of the circle is chosen.

We propose an alternative heuristic that uses an embedded number partition algorithm. We begin by placing the blades in random locations on the circle. Then we choose two perpendicular axes of symmetry as shown in Figure 7b (as has been the custom in all previous work on this problem, we assume an even number of fan blades). Next we balance the center of mass around "the X-axis" then around "the Y-axis".

To balance the center of mass around an axis, we consider pairs of weights which are symmetric with respect to the axis as shown in figure 7c. First we arrange each symmetric pair so that the heaviest weight in each pair is on the same side of the axis (actually we do this to make the algorithm description simpler). Next we create a single number d_i for each pair of weights ($i = 1$ to $N/2$) as shown in Figure 7d. This number d_i is the center of mass of the pair with respect to the axis of symmetry. We next apply a number partitioning algorithm to the set $\{d_1, d_2, \ldots d_{N/2}\}$ of pairwise centers of mass. The result of partitioning is two sets of pairs. The final step is to arbitrarily select one of the two sets of pairs, and interchange the weights of each pair in that set. The result will be that the center of mass with respect to the axis of symmetry will be nearly balanced and equal to the objective function found by the number partitioning algorithm.

The final step of the algorithm is to balance the center of mass with respect to the second perpendicular (Y) axis of symmetry. We apply the same algorithm as in the last paragraph. A key observation is that applying the algorithm to produce balance around the Y-axis does not effect the balance around the X-axis obtained in the first step.

Following Amiouny, Bartholdi and Vande Vate (1997), we generated blade weights from a Normal distribution with a mean of 100 and standard deviation of 5/3. We generated problems over a range of sizes from 20 blades to 200 blades. Again following Amiouny, Bartholdi and Vande Vate (1997), we assume the circle radius 100 and that the objective function is the Euclidean distance between the center of mass and the center of the circle. For each problem size, 1000 instances were generated. Three algorithms were compared, ordinal pairing, greedy pairing, and the number partition based method. The number partitioning problems were solved using the differencing algorithm. All three algorithms were coded by the author and run on the same PC. Results appear in Figures 8 through 11. Figure 8 shows the objective function value averaged over 1000 problem instances for each problem size. Figure 10 shows the elapsed runtime required to solve all 1000 problems of a given size. The results are striking. The number partition based algorithm improves on greedy pairing by up to three orders of magnitude yet runs in roughly the same time as ordinal pairing.

(Figures 8 through 11 around here)

Amiouny, Bartholdi and Vande Vate (1997) commented that their algorithms seemed to have a reasonably poor worst case performance as indicated by figure 9. In this figure the worst objective function value over the 1000 problem instances of each size is plotted. We note the relative poor performance of ordinal and greedy pairing in the same figure. Similar behavior may be observed in the standard deviation over the 1000 instances as seen in figure 11. This behavior lead Amiouny, Bartholdi and Vande Vate to conjecture that high variance was an inherent property of the blade balancing problem. The performance of the number partition based algorithm does not show the same high variability, and seems to disprove the conjecture of Amiouny, Bartholdi and Vande Vate.

Finally we note that had we solved the embedded number partitioning problems using algorithm A3 rather than the differencing algorithm, performance would have improved significantly (albeit with increased computation times).

6.0 Conclusions

In this paper we have attempted to demonstrate the usefulness of the differencing algorithm and its extensions for solving problems with embedded number partitioning problems. We have shown that number partitioning is easy in that solutions very close to optimal, or indeed optimal can typically be found with very little computational effort. Further we have demonstrated our approach by developing new algorithms for two well-known problems, multi-processor scheduling and turbine rotor blade balancing. The results of these algorithms were impressive, especially in the case of blade balancing where the algorithms proved better than the best known and as fast as the fastest reasonable alternative. Given that many problems have been proven NP-complete by reducing them to number partitioning, we suspect that many other basic combinatorial optimization problems are amenable to approaches similar to those presented in this paper.

This is the first in a series of two papers on extensions of and uses for the differencing algorithm for number partitioning. In the second paper we address the problem of "multi-criteria number partitioning" where each element in the set to be partitioned is associated with more than one number. This problem also has several applications including data splitting in statistical model building, and allocating subjects to trial groups in designing clinical trials.

Acknowledgements

This work was supported in part by National Science Foundation grant DMI 9809479, and NATO Collaborative Research Grant CRG-971489 for which we are grateful. We also thank Professor Wonjoon Choi for introducing us to the rotor blade balancing problem, and Dr. Ray Correll for providing insights on fixed cardinality and subset sum problems.

References

Amiouny, S.V., Bartholdi, III, J.J., and Vande Vate, J.H. 1997. Heuristics for Balancing Turbine Fans. Technical report, Department of Industrial and Systems Engineering, The Georgia Institute of Technology, Atlanta. (Forthcoming in Operations Research).

Arguello, M.F., Feo, T.A., and Goldschmidt, O. !996. Randomized methods for the number partitioning problem.Computers Ops. Res. 23, 103-111.

Chu, C. and Antonio, J. 1999. Approximation algorithms to solve real life multicriteria cutting stock problems, Ops. Res. 47, 495-508.

Coffman Jr., E.G., Garey, M.R., and Johnson, D.S. 1978. An application of bin-packing to multiprocessor scheduling. Siam J. Computing. 7, 1-17.

Fathi, Y. and Ginjupalli, K.K. 1993. A mathematical model and a heuristic for the turbine balancing problem. European Journal of Operations Research. 63, 336-342.

Garey,M.R., and Johnson, D.S. 1979.Computers and intractability a guide to the theory of NP-completeness. W.H. Freeman and Company, New York.

Graham, R.L. 1969. Bounds on multiprocesing time anomalies. Siam J. Applied Mat. 17, 263-269.

Johnson, D.A., Aragon, C.R., McGeoch, L.A., and Shevon, C. 1991. Optimization by simulated annealing: an experimental evaluation, part II, graph coloring and number partioning. Opns. Res. 39, 378-???

Karmarkar, N.R.M., and Karp, R.M. 1982. The differencing method for set partitioning. Report No. UCB/CSD 82/113, Computer Science Division, University of California, Berkeley.

LaParte, G. and Mercure, H. 1988. Balancing hydraulic turbine runners: a quadratic assignment problem. European Journal of Operations Research. 353, 378-381.

Mason, A. and Ronnqvist, M. 1997. Solution methods for the balancing of jet turbines. Computers & Operations Research. 24(2), 153-167.

Mosevich, J. 1986. Balancing hydraulic turbine runners: a discrete combinatorial optimization problem. European Journal of Operations Research. 26, 202-204.

Snyder, K.T. 1995. Problem space local search approaches for parallel machine scheduling. Master's thesis #04-95, Department of Industrial and Manufacturing Systems Engineering, Lehigh University, Bethlehem PA.

Storer, R.H., Flanders, S.W., and Wu, S.D., 1996. Problem space search for number partitioning. Annals of Operations Research. 10, 465-487.

Subject Classifications

Mathematics, combinatorics: extensions of number partitioning heuristics.

Production/scheduling, multiple machine: multiprocessor heuristics.

Manufacturing: rotor blade balancing.

Statement of Contribution

AUTHORS: Robert H. Storer

TITLE: Extensions of and Uses for the Differencing Algorithm for Number Partitioning

STATEMENT: While the number partitioning problem is well known, and often used to prove other problems NP-complete, few if any uses for the problem itself have been put forth. We conjecture that this is because both standard integer programming approaches, and more recently popular local search heuristics are both essentially useless for solving number partitioning problems. It seems that when a number partitioning problem is found embedded in some other problem, the larger problem is declared NP-complete, and alternative approaches for solution are sought. What seems to be not widely known is that there are extremely fast and powerful algorithms for solving number partitioning. In this paper we demonstrate the effectiveness of these surprisingly simple approaches and conclude that number partitioning is in *some sense* easy.

The ability to solve number partitioning problems quickly and effectively makes possible algorithms that directly address number partitioning problems embedded in more complex problems. We conjecture that such an approach may prove useful in many applications. After discussing extensions to basic number partitioning (fixed cardinality number partitioning, subset sum problems, and stochastic number partitioning), we develop algorithms for two well known combinatorial optimization problems.

The first application is to parallel machine (a.k.a multi-processor) scheduling, perhaps one of the oldest and most widely studied combinatorial optimization problems. A number partitioning based algorithm is shown to be an attractive alternative to known algorithms when the number of jobs is relatively large in comparison to the number of machines.

The second application is to the fairly well studied turbine rotor blade balancing problem. A number partitioning based heuristic is shown to be far superior to the most recently developed approaches. We conclude that the number partition based approach is superior to the best known (and time consuming) alternative by several orders of magnitude. Further, the run time of the algorithm is as fast as the fastest reasonable heuristic.

Table I. Progress of Cplex 6.5 on a problem with 100 numbers each with nine digits

Node Number	Best Integer Solution	Run Time (secs.)
0	9.04E+08	unkn.
10	1.69E+08	unkn.
30	5.74E+07	unkn.
57	4691137	unkn.
187	2934660	unkn.
222	242690	unkn.
604	203090	unkn.
877	94031	unkn.
2437	7233	unkn.
8778	6141	unkn.
39124	196	unkn.
10000000	196	606.74
20000000	30	1215.38
60000000	8	3662.38
21000000	5	12867.31
36000000	4	22044.52
56000000	2	34273.94
64534941	0	39161.61

Table II. Percent Optimal for Algorithm A3. 1000 Iterations, theta=0.04.
Problems generated at randomly

		N									
		10	20	30	40	60	80	100	150	200	250
	2	90.9	100	100	100	100	100	100	100	100	100
	3	30.5	100	100	100	100	100	100	100	100	100
	4	3.6	100	100	100	100	100	100	100	100	100
Digits	5	0.4	76.7	100	100	100	100	100	100	100	100
	6	0	13.7	98.2	100	100	100	100	100	100	100
	7	0	1.9	41.6	92.5	100	100	100	100	100	100
	8	0	0.1	5.8	24.2	91.3	100	100	100	100	100
	9	0	0	1	3.6	25	77.3	99	100	100	100

Table III. Percent Optimal for Differencing Algorithm
Problems generated at randomly

		N									
		10	20	30	40	60	80	100	150	200	250
	2	58	98.2	100	100	100	100	100	100	100	100
	3	12.6	72.7	96.9	99.6	100	100	100	100	100	100
	4	1.9	14.1	59.3	90.6	99.5	100	100	100	100	100
Digits	5	0.3	1.3	11.6	35.5	90.2	98.4	100	100	100	100
	6	0	0.1	1.3	5.6	34.6	75.7	95.6	99.9	100	100
	7	0	0	0.1	0.5	5.4	16.9	54	96.3	99.6	100
	8	0	0	0	0	0.7	1.7	7.9	59	95.8	99.8
	9	0	0	0	0	0.3	0.2	1	10.4	53.2	87.4

Table IV. Percent Optimal for algorithm A3. 1000 Iterations, theta=0.04.
Problems have known optimal solution of zero

		N									
		10	20	30	40	60	80	100	150	200	250
	2	99.8	100	100	100	100	100	100	100	100	100
	3	99.1	100	100	100	100	100	100	100	100	100
	4	98.7	100	100	100	100	100	100	100	100	100
Digits	5	98.7	86.1	100	100	100	100	100	100	100	100
	6	99	67.6	97.2	100	100	100	100	100	100	100
	7	98.6	66.7	33.8	84.2	100	100	100	100	100	100
	8	99	66.4	6.7	17.5	85	100	100	100	100	100
	9	98.5	65.9	3.5	1.9	16.9	65.3	98.5	100	100	100

Table V. Percent Optimal for Differencing Algorithm. 1000 Iterations.
Problems have known optimal solution of zero

		N									
		10	20	30	40	60	80	100	150	200	250
	2	63.8	97.4	99.9	100	100	100	100	100	100	100
	3	34.7	62.3	93.9	99.8	99.9	100	100	100	100	100
	4	31.3	11	52.5	85.7	99.2	100	100	100	100	100
Digits	5	33.4	2	7.8	25.3	84.6	98.9	99.8	100	100	100
	6	32.3	0.4	0.7	2.8	27.2	67.2	93.3	99.8	100	100
	7	30.4	0.7	0	0	3.3	13.9	42.5	95.3	99.7	99.9
	8	30.3	0.7	0	0	0.3	1.6	5.9	52.7	92.6	99.2
	9	31.5	0.3	0	0	0.1	0	0.8	7.6	42.6	84.7

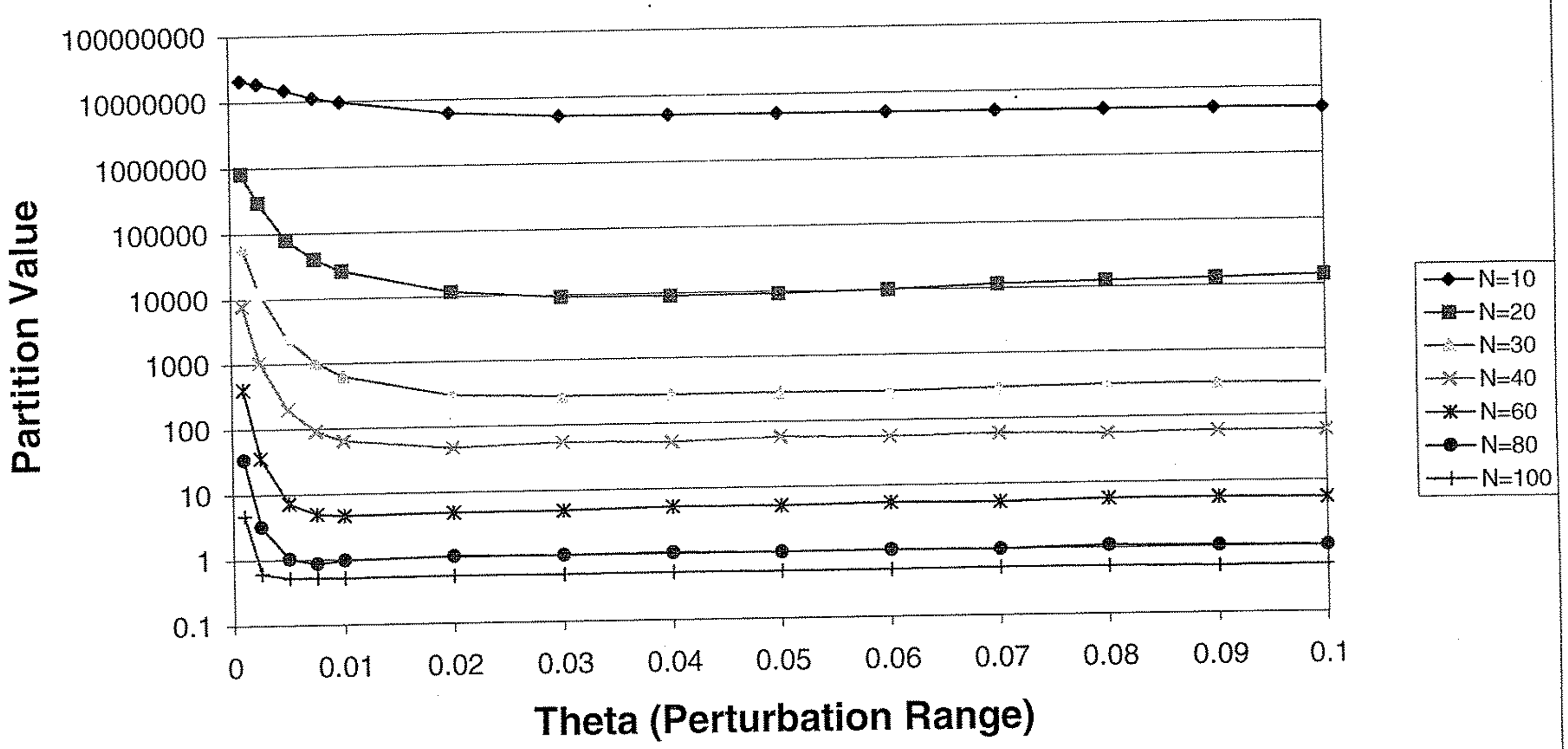

Figure 1. Results of the tuning experiment showing the effect of varying theta value on performance for various problem sizes N.

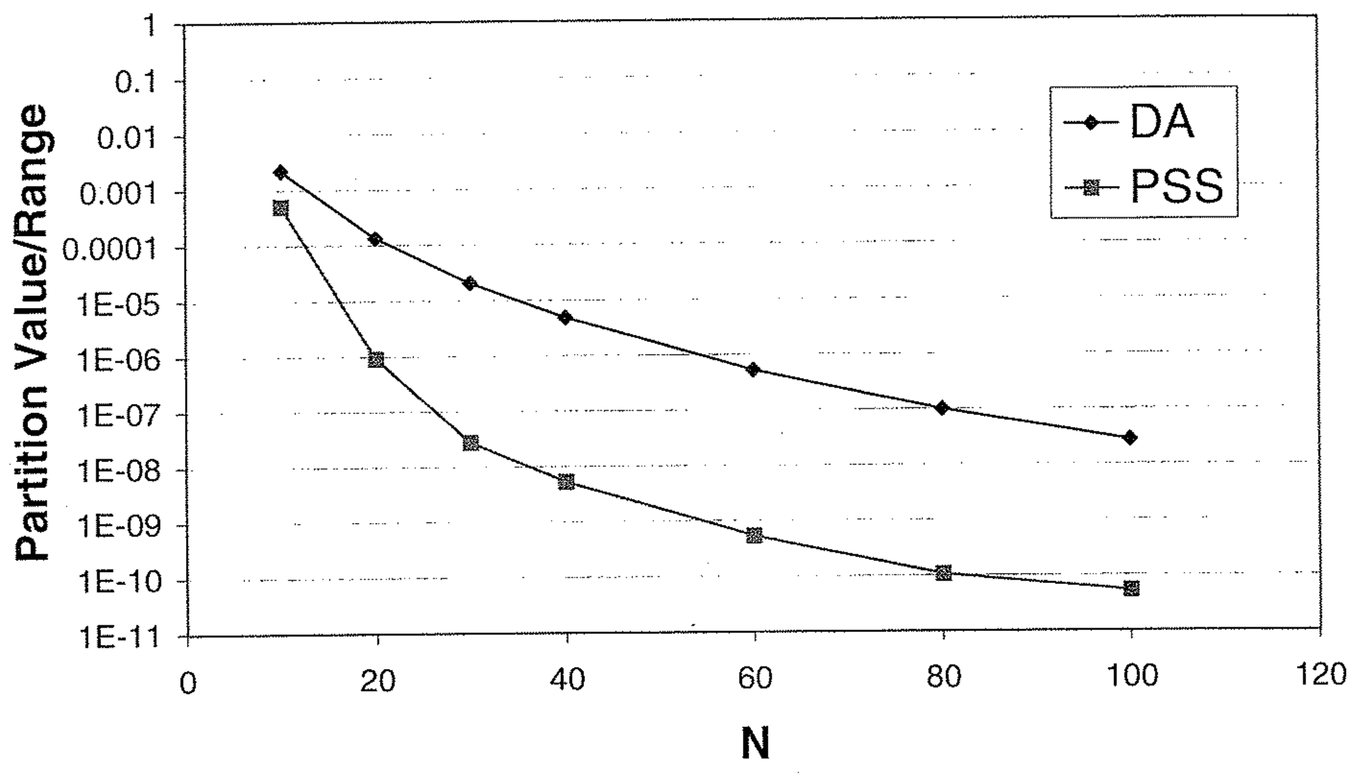

Figure 2. Performance of the differencing algorithm and algorithm A3 as a function of N. Theta = 0.04*Range

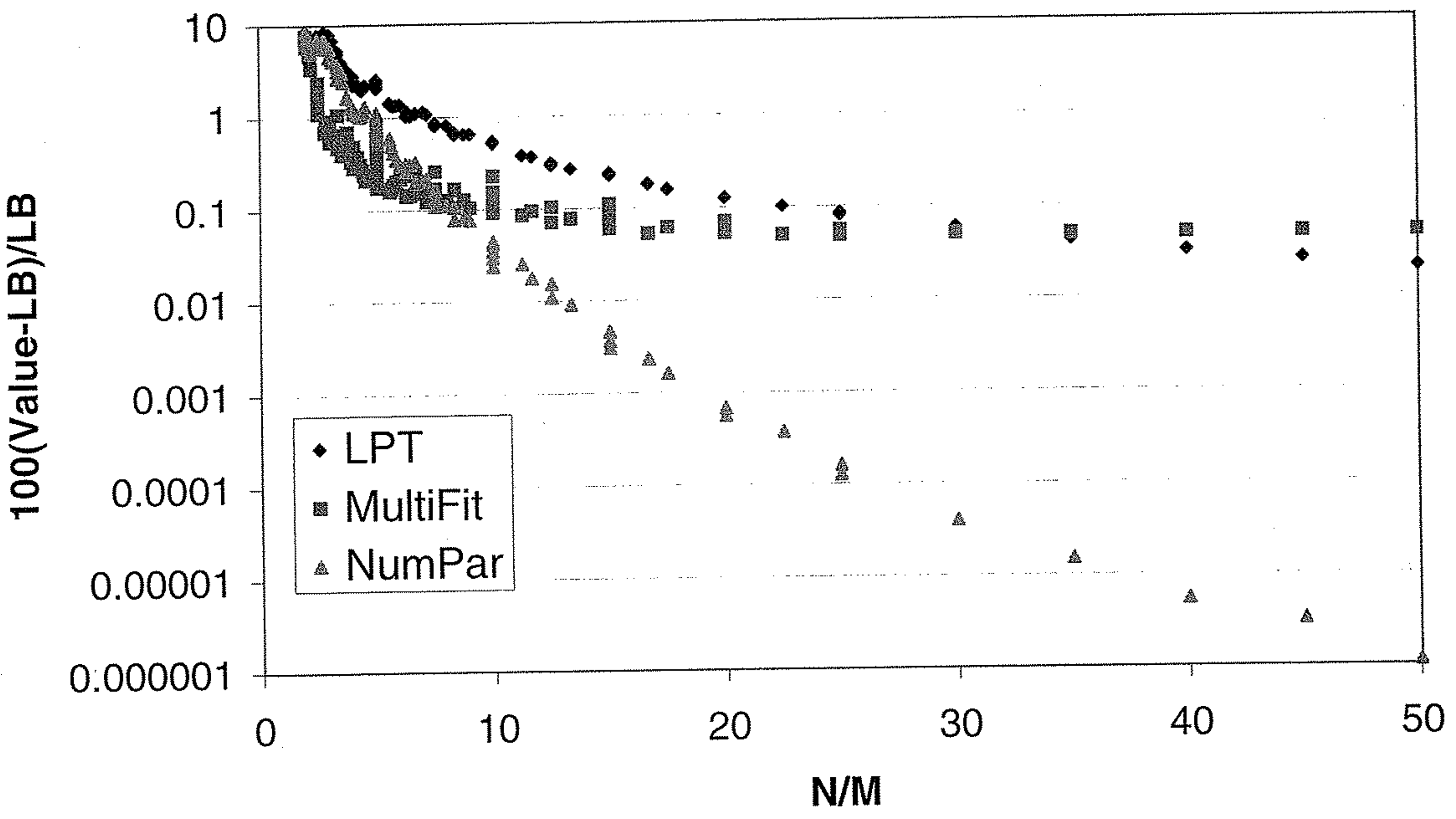

Figure 3. Performance of LPT, MULTIFIT, and NP based method on multi-processor scheduling problems of various sizes.

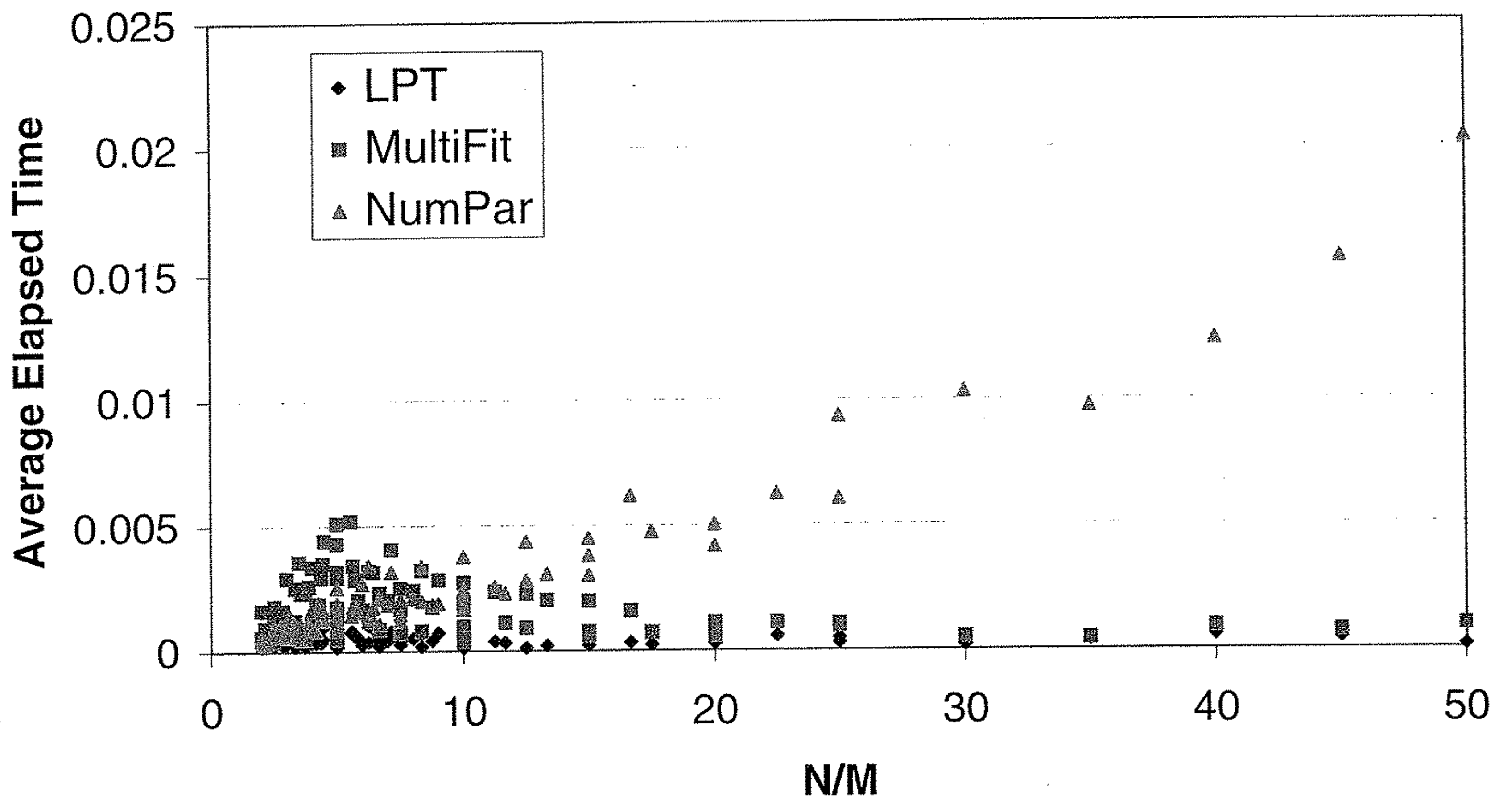

Figure 4. Run time comparison of LPT, MULTIFIT, and the NP based method

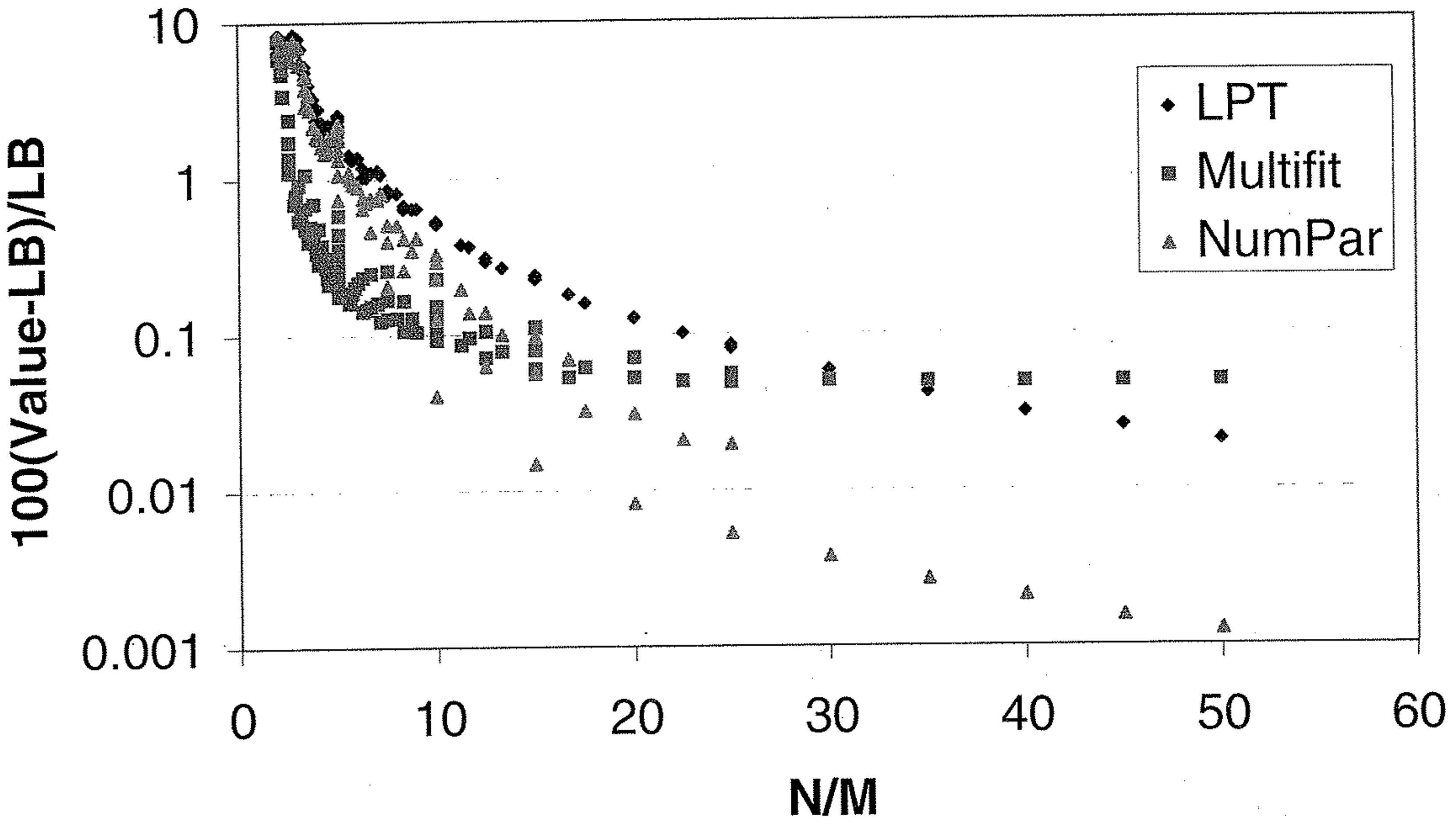

Figure 5. Performance of four iterations of the NP based algorithm compared to LPT and MULTIFIT

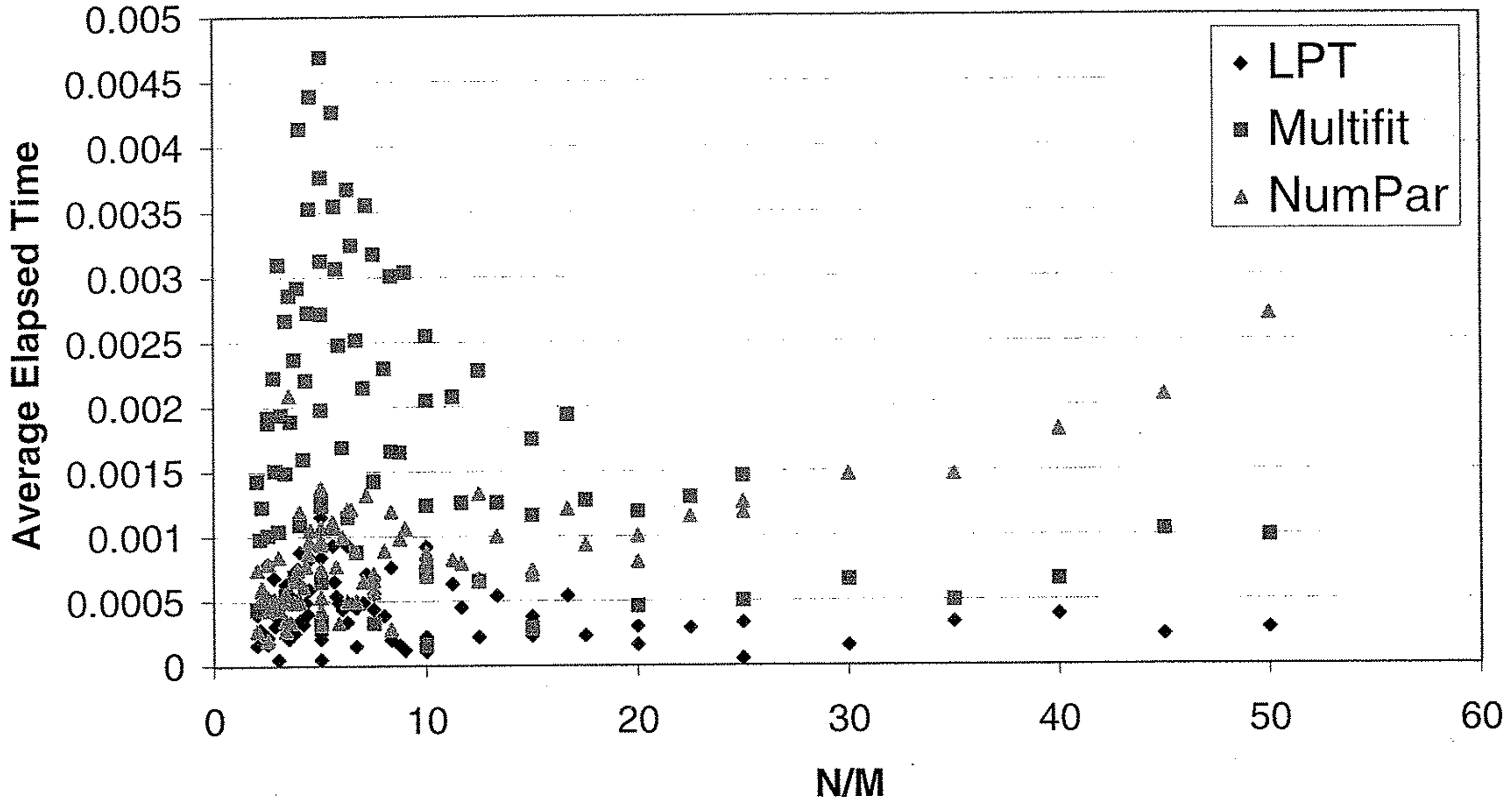

Figure 6. Run time comparison of four iterations of the NP based algorithm compared to LPT and MULTIFIT

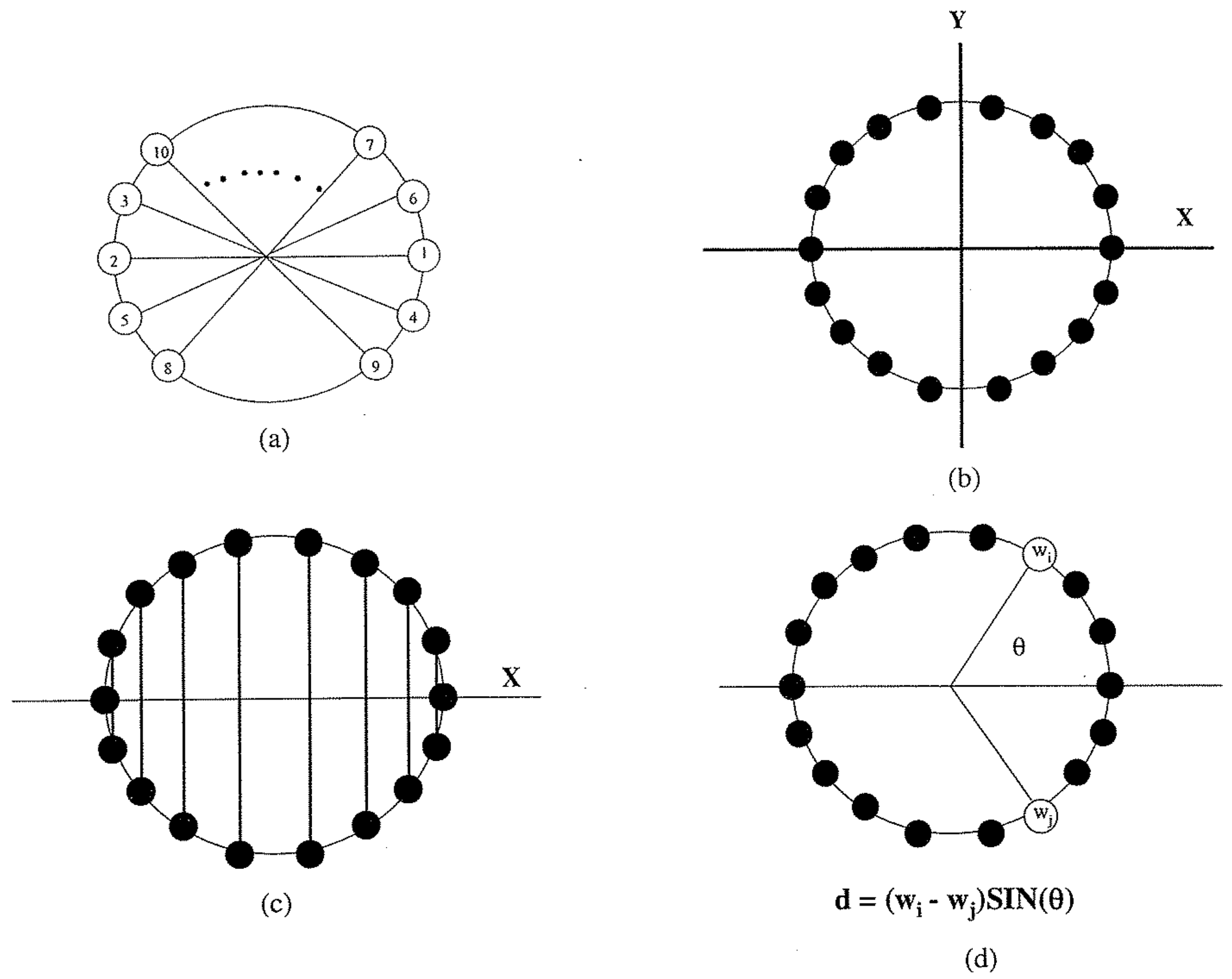

Figure 7. a) Weight location pattern for the Ordinal Pairing Algorithm. Weights labeled from heaviest to lightest starting with 1
b) Perpendicular axes of symmetry
c) Weight pairs symmetric with respect to the X axis
d) Calculation of d_i values

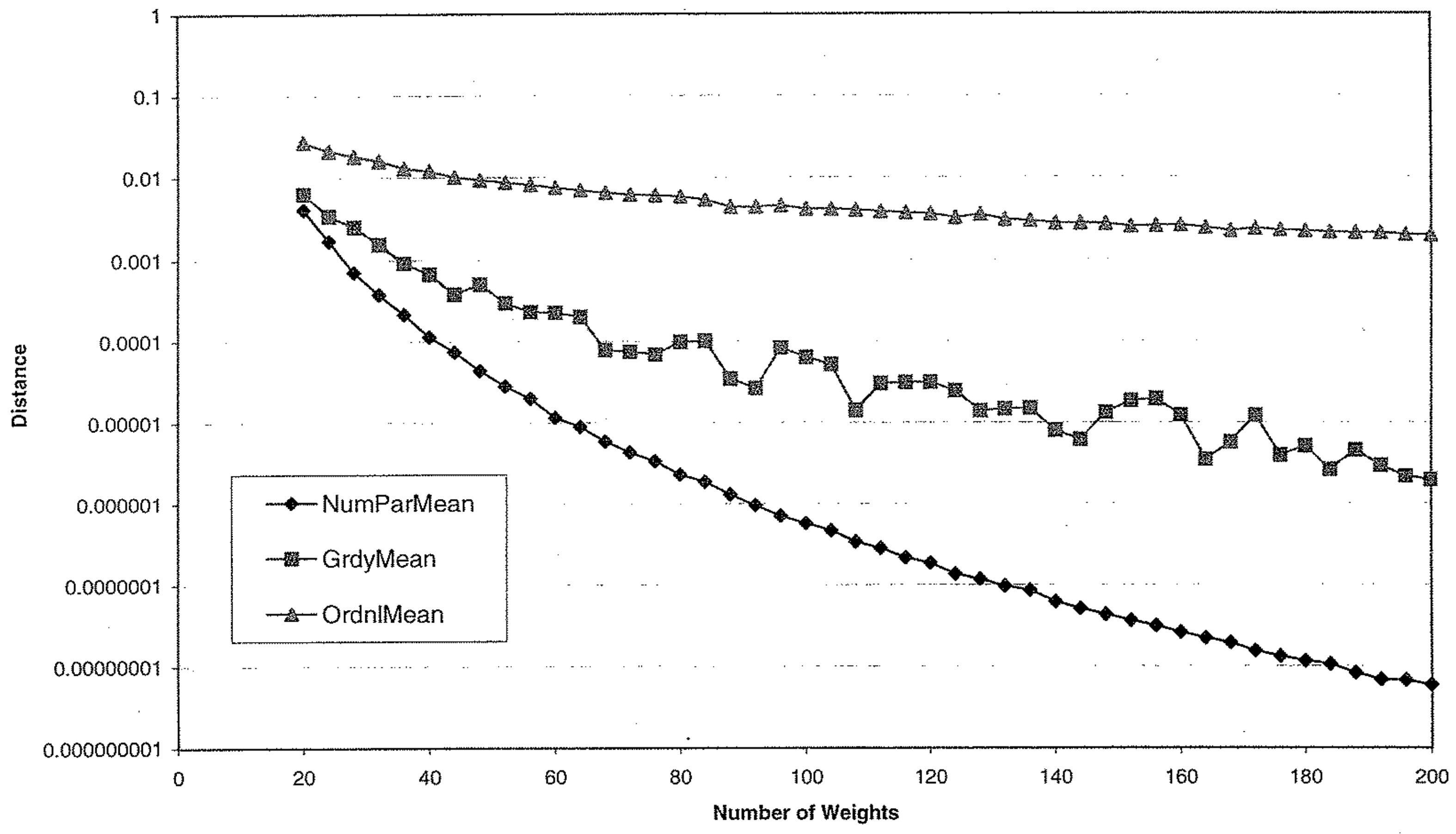

Figure 8. Comparison of the performance of ordinal pairing, greedy pairing, and the number partition based algorithm for various numbers of weights. Each point is the average performance over 1000 randomly generated problems.

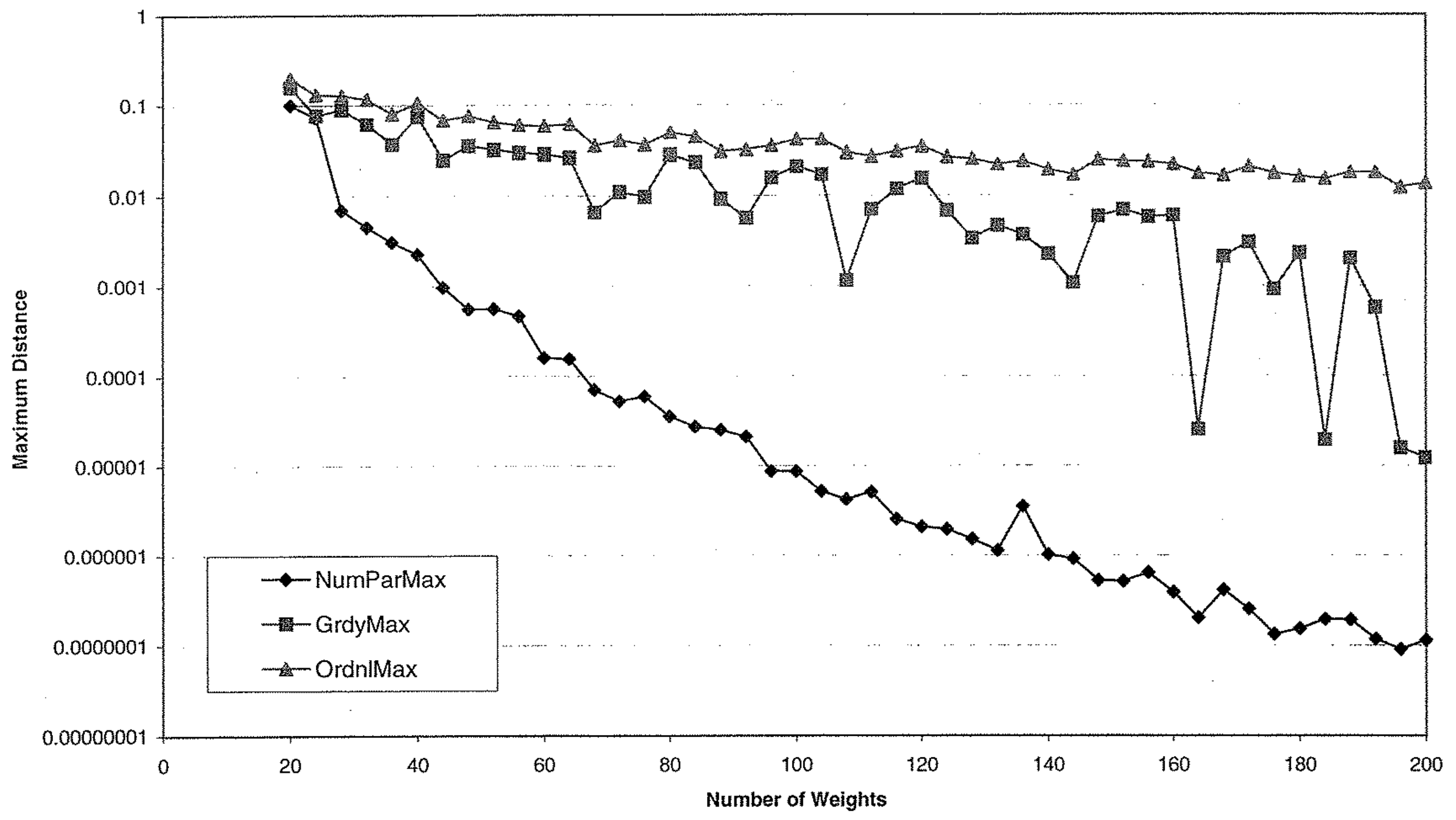

Figure 9. Comparison of worst case performance of ordinal pairing, greedy pairing, and the number partition based algorithm for various numbers of weights. Each point is the maximum of the performance measure over 1000 randomly generated problems

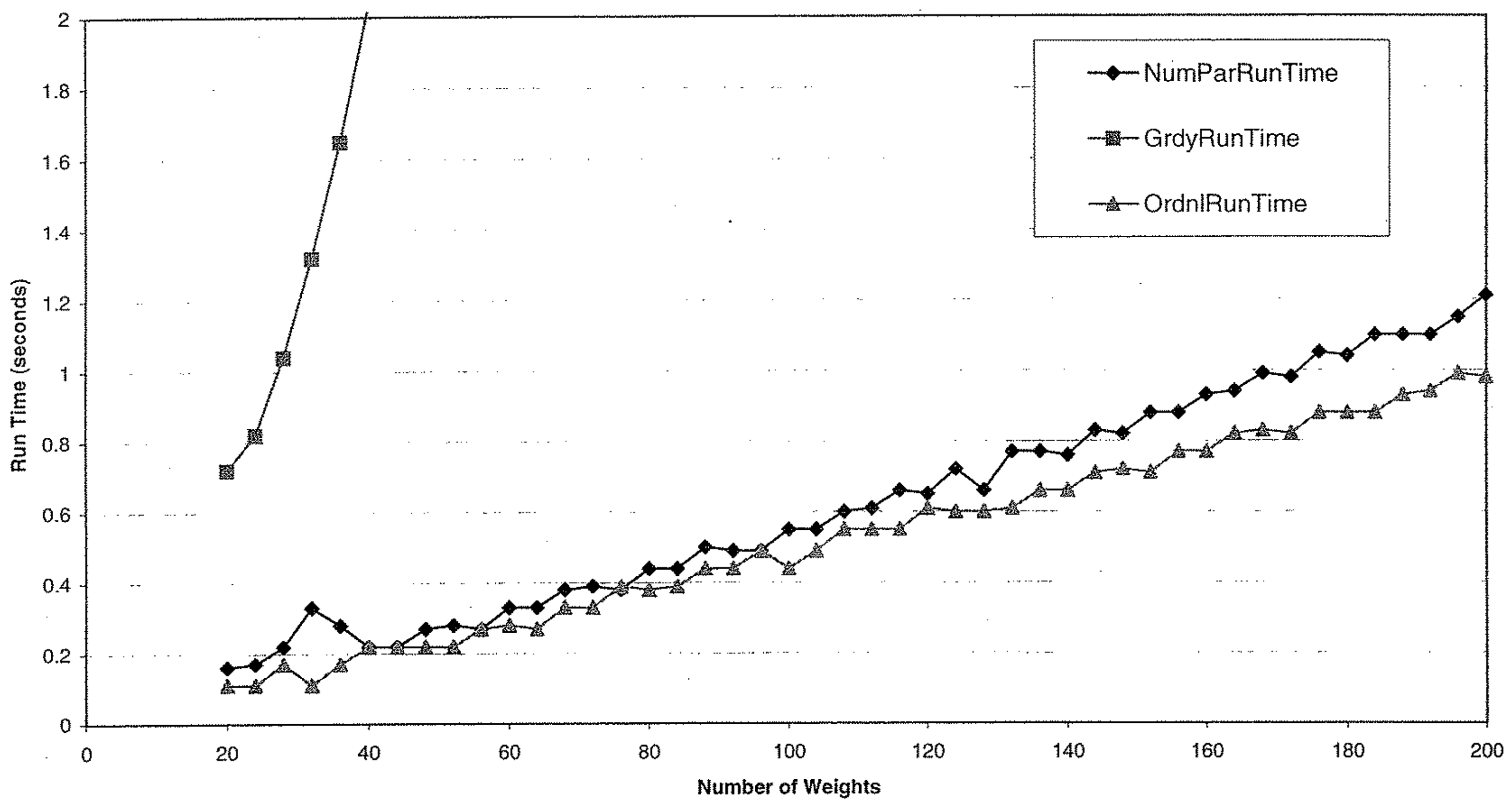

Figure 10. Comparison of run times of ordinal pairing, greedy pairing, and the number partition based algorithm for various numbers of weights. Each point is elapsed time to solve all 1000 randomly generated problems

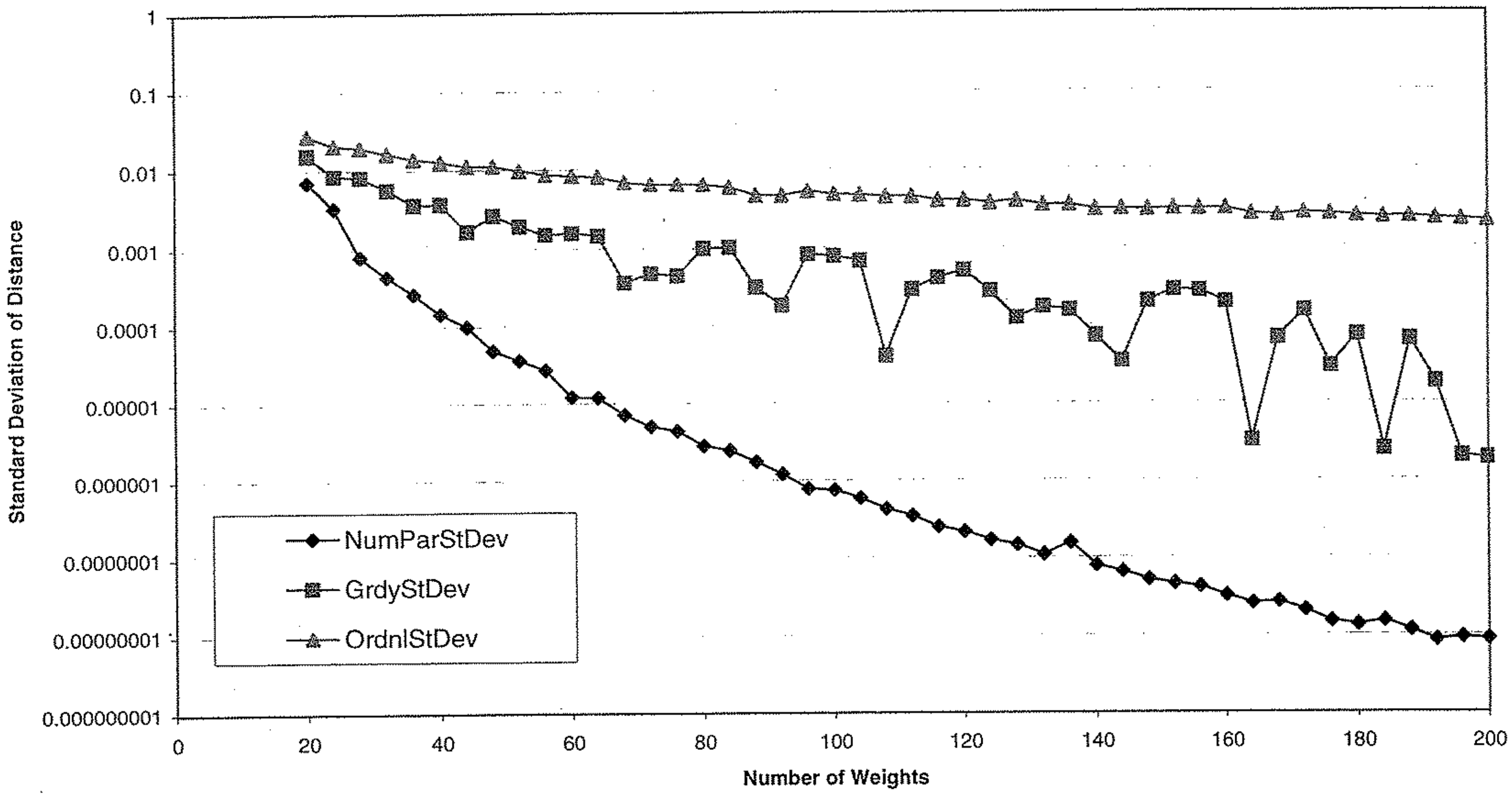

Figure 11. Comparison of performance variability of ordinal pairing, greedy pairing, and the number partition based algorithm for various numbers of weights. Each point is the standard deviation of the performance measure over 1000 randomly generated problems