Core problems in Knapsack Algorithms.[*]

David Pisinger
Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark.

June, 1994

Abstract

Since Balas and Zemel a dozen years ago introduced the so-called core problem as an efficient way of solving the Knapsack Problem, all the most successful algorithms have been based on this idea. Balas and Zemel proved, that there is a high probability for finding an optimal solution in the core, thus avoiding to consider the remaining items. However this paper demonstrates, that even for randomly generated data instances, the core problem may degenerate, making it difficult to obtain a reasonable solution. This behavior has not been noticed before due to inadequate testing, since the capacity usually is chosen such that the core problem becomes as easy as possible.

A model for the expected hardness of a core problem as function of the capacity is presented, and it is demonstrated that the hitherto applied test instances are among the easiest possible. As a consequence we propose a series of new randomly generated test instances, and show how recent algorithms behave when applied to these problems.

Keywords: Packing; Knapsack Problem; Analysis of algorithms; Expected Hardness.

1 Introduction

We consider the following problem: Given n items with corresponding *profits* p_j and *weights* w_j. The *Knapsack Problem* is the task of packing some of these items in a knapsack of *capacity* c, such that the profit sum of the included items is maximized.

In order to solve the problem efficiently, it is suitable to order the items according to nonincreasing profit-to-weight ratios, since in this way tight bounds may be derived in a branch-and-bound algorithm. Balas and Zemel (1980) noted, that this sorting often takes

[*]Technical Report 94/26, DIKU, University of Copenhagen, Denmark.

up the majority of the computational time, but the sorting may be avoided by considering a sufficiently small subset of the items known as the *core*. The core problem is a usual knapsack problem defined on a small subset of the available items, where there is a high probability of finding an optimal solution. A solution to the core problem is then used as a lower bound in a reduction algorithm, which tries to fix decision variables at their optimal values by applying some bounding rules. The remaining problem is finally solved exactly through enumeration.

Balas and Zemel (1980) used an approximate algorithm for solving the core problem, showing that the probability for the heuristic to find an optimal solution grows with the size of the instance. The proof is based on the assumption that the weights w_j in a core are uniformly distributed, making it quite easy to obtain a filled knapsack by repeatedly removing an item and replacing it with some others.

Martello and Toth (1988) improved the algorithm by solving the core problem exactly through branch-and-bound. The benefits are obvious: a better solution is found, thus making it possible to reduce more variables in the reduction part. If the found solution reaches any upper bound for the Knapsack Problem, the algorithm may even halt after having solved the core problem. In a comprehensive test, Martello and Toth (1990) demonstrate that their code MT2 is the best of the codes by Nauss (1976), Fayard and Plateau (1982), and Martello and Toth (1977,1988).

However Pisinger (1994c) detected some situations where the variance of the weights in a core is very small, contradicting the assumption by Balas and Zemel. This means, that it may be very difficult to obtain a filled knapsack, and thus to obtain a good lower bound. For the approximate algorithm by Balas and Zemel it means that the heuristic solution is worse than expected, implying that less items may be fathomed in the reduction phase. For MT2 — which solves the core problem to optimality — it may result in extremely long computational times, since no good lower bound can be used to cut off branches of the search tree.

In this paper we prove that hard core problems are not the exception but rather the rule, even for randomly generated data instances. This problem has not been noted before due to inappropriate testing: In all experiments reported, the capacity c was chosen as half of the total weight sum, which results in very easy core problems. Therefore we will here focus on how the hardness of a core problem depends on the capacity c.

We will consider the most common randomly generated data instances from the literature: *Uncorrelated data instances* (UC): p_j and w_j are randomly distributed in $[1, R]$. *Weakly correlated data instances* (WC): w_j randomly distributed in $[1, R]$ and p_j randomly distributed in $[w_j - R/10, w_j + R/10]$ such that $p_j \geq 1$. *Strongly correlated data instances* (SC): w_j randomly distributed in $[1, R]$ and $p_j = w_j + 10$. *Subset-sum data instances* (SS): w_j randomly distributed in $[1, R]$ and $p_j = w_j$. The *data range* R will be tested with values $R = 100, 1000$ and 10000.

This paper is organized as follows: First we will define the core problem in Section 2, and describe which problems this may cause. Then a model for the expected hardness of a core problem is proposed in Section 3, and it is demonstrated that the hitherto applied method for testing algorithms is based on the easiest possible data instances. In Section

4 we use the model developed for predicting the hardness of different large scale data instances. Finally in Section 5, we will present a better way of generating test instances, and use this method for testing different algorithms, which all solves some kind of core problem.

2 Definitions

The *Binary Knapsack Problem* is defined as the following optimization problem:

$$\begin{aligned} \max \quad & z = \sum_{j=1}^{n} p_j x_j, \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \le c, \\ & x_j \in \{0,1\}, \quad j = 1,\dots,n, \end{aligned} \tag{1}$$

where p_j, w_j and c are positive integers. Without loss of generality we may assume that all items fit into the knapsack, i.e. that

$$w_j \le c \;\text{ for }\; j = 1,\dots,n, \tag{2}$$

and to avoid trivial problems we assume that $\sum_{j=1}^{n} w_j > c$.

The *Linear Knapsack Problem*, which is defined by relaxing the integrality constraint on x_j in (1) to $0 \le x_j \le 1$, may be solved by using the *greedy principle*: Order the items according to their *efficiencies* $e_j = p_j/w_j$ such that

$$e_i \ge e_j \;\text{ when }\; i < j, \tag{3}$$

and then include items $1, 2, 3, \dots$ as long as the next item fits into the knapsack. The first item which does not fit into the knapsack is denoted the *break item* b, and we obtain the LP-optimal solution by setting $x_j = 1$, for $j = 1,\dots,b-1$, and $x_j = 0$, for $j = b+1,\dots,n$, while x_b is set to

$$x_b = \frac{c - \sum_{i=1}^{b-1} w_i}{w_b}. \tag{4}$$

Having found the break item, we may derive upper and lower bounds on the Knapsack Problem, which again may be used for reducing the problem size (Ingargiola and Korsh 1973, Dembo and Hammer 1980, Martello and Toth 1988). The reduced problem is then solved exactly using enumerative techniques.

Balas and Zemel (1980) noticed, that the sorting in (3) often takes up a majority of the computational time, introducing the *core problem* as an efficient way of solving the Knapsack Problem: Assume that the optimal solution was given in advance. Then we could choose the *core* C as an interval $[\alpha, \beta]$ of the sorted items, were α is the first item where $x_\alpha = 0$ and β is the last item for which $x_\beta = 1$. The core problem — an ordinary Knapsack Problem defined on the core C — could then be solved in a common way, while we set $x_j = 1$ for $j = 1,\dots,\alpha-1$ and $x_j = 0$ for $j = \beta+1,\dots,n$.

j	p_j	w_j	e_j
1	92	91	1.01099
2	93	92	1.01087
3	93	92	1.01087
4	93	92	1.01087
5	93	92	1.01087
6	93	92	1.01087
7	93	92	1.01087
8	94	93	1.01075
9	95	94	1.01064
10	95	94	1.01064
11	95	94	1.01064
12	95	94	1.01064
13	95	94	1.01064
14	95	94	1.01064
15	96	95	1.01053
16	96	95	1.01053
17	97	96	1.01042
18	97	96	1.01042
19	98	97	1.01031
20	98	97	1.01031
21	98	97	1.01031
22	98	97	1.01031
23	98	97	1.01031

Table I: Section of a difficult core. Uncorrelated instance, $n = 30000$, $R = 100$. The break item is $b = 14$ with efficiency $e_b = 1.01064$.

Obviously the core is not known in advance, but an *approximate core* may be found in $O(n)$ time (Balas and Zemel 1980) by partitioning the items in three sets H, C, L according to their efficiencies, such that the break item is in the core C:

$$e_i \geq e_j \geq e_k, \quad i \in H, j \in C, k \in L, \tag{5}$$

$$\sum_{j \in H} w_j \leq c < \sum_{j \in H \cup C} w_j. \tag{6}$$

Since all items in the core will have similar efficiencies, the core problem basically consists of finding an optimally *filled* knapsack. Assuming that the weights w_j in the core are randomly distributed, Balas and Zemel proved that a simple exchange-algorithm called H (described in appendix A) will find a *filled* knapsack solution, with a probability that grows with larger core size, and smaller data range. Moreover the probability that a filled solution is an optimal solution, is growing as n is increased.

The size of the core should be chosen sufficiently large to find an optimal solution, but also small enough to avoid unnecessary enumeration. Balas and Zemel (1980) proposed the size $|C| = 50$, while Martello and Toth (1988) chose

$$|C| = \begin{cases} n & \text{if} \quad n \leq 100 \\ \sqrt{n} & \text{if} \quad n > 100. \end{cases} \tag{7}$$

Although solving the core problem means that a smaller problem is enumerated, it does not necessarily mean that the problem is easier to solve (Pisinger 1994c). Table I shows a section of a difficult core: Although the profits and weights are randomly and independently distributed in $[1, 100]$, the only items with efficiencies around $e_b = 1.01064$

are pairs $p_j = w_j + 1$, where w_j is close to 94. Evidently it is difficult to fill the knapsack if the capacity is not close to a multiple of 94. Since the core is of fixed size, no lighter items from outsides the core may be included, although they fit exactly into the residual capacity.

According to Balas and Zemel (1980), this core problem should be easy to solve, since the profits and weights of the original problem are randomly distributed independently on each other. Moreover the data-range R is relatively small, and the size n is large. But what we see here is a degeneration of the core, which Balas and Zemel did not take into account.

3 A model for the hardness of a core problem

Essentially the core problem consists of finding a filled knapsack, since the uniformity of the efficiencies implies that a filled solution generally also is an optimal solution. But the ability to obtain a given weight sum c clearly depends on two properties of the affected items: How well the weights in C are spread out across the range R, as well as the average weight of the items in C. Light items may easily fill the empty gap in a knapsack, and if the weights are equally scattered, almost any capacity c may be obtained. Thus the expected hardness $\mathcal{H}$ of the core problem may be defined as

$$\mathcal{H} = \overline{w}\chi, \tag{8}$$

where $\overline{w}$ is the average weight of the core items, and χ is a measure of the clustering. The average weight of the items in C is

$$\overline{w} = \frac{1}{|C|} \sum_{j \in C} w_j, \tag{9}$$

while the clustering of the items is found by dividing the data range R in 10 equally sized intervals I_i, with N_i being the number of items $j \in C$ where $w_j \in I_i$. The χ^2 (chi-square) value is a common test for the scatteredness of some values. It is found as the squared distance of N_i from the expected frequency $\mu = |C|/10$, scaled by μ. Thus as a measure for the clustering of the weights we may use χ given by

$$\chi = \sqrt{\frac{\sum_{i=1}^{10}(N_i - \mu)^2}{\mu}}. \tag{10}$$

This model conforms with Proposition 1 by Balas and Zemel (see appendix A), since if the weights are homogeneously scattered in a small interval $[1, R]$ (meaning that both χ and $\overline{w} = R/2$ are small), the heuristic H easily will find a filled solution.

As an example we will consider a weakly correlated data instance with 3000 items, and data range $R = 100$. The core size is chosen as $|C| = 50$, and we monitor the average weight and clustering as a function of the knapsack capacity c. To smooth out the stochastic variation of each instance, we give the average values for 500 different data

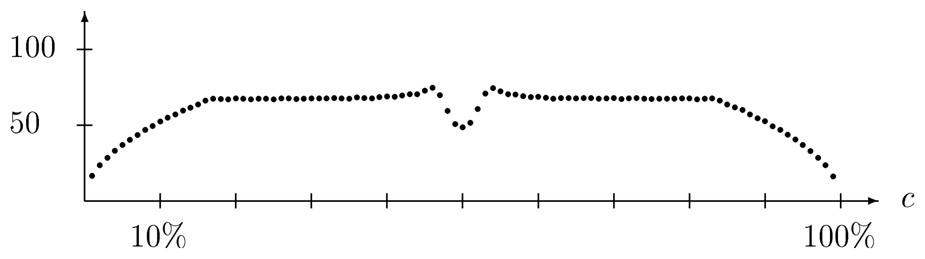

Figure 1: Average weight $\overline{w}$ of core as function of the capacity c. Average of 500 weakly correlated instances, $n = 3000, R = 100$.

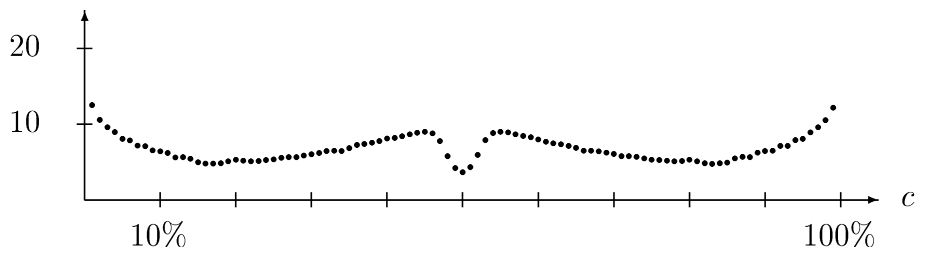

Figure 2: Clustering χ of core as function of the capacity c. Average of 500 weakly correlated instances, $n = 3000, R = 100$.

instances. In Figure 1 the average weight $\overline{w}$ of the core is shown when varying the knapsack capacity from 1% to 99% of the total weight sum. Figure 2 gives the corresponding χ-value of each core.

To test the theory that $\mathcal{H} = \overline{w}\chi$, we show the expected core hardness in Figure 3 and compare it to the quality of solutions found by heuristic H, measured as the residual capacity of the best filling (Figure 4). If heuristic H is able to find a filled solution and thus a good lower bound, obviously the core problem will be easy to solve (and vice versa). Both figures show the average values of the same 500 data instances for each percentage of the capacity. It is seen, that the two figures have the same characteristics, supporting the correctness of our model.

Finally we compare the expected core hardness with the actual running times of the MT2 algorithm in Figure 5. MT2 solves the core problem to optimality, using a considerable amount of the solution time for this step. The same instances are considered as before. Due to the exponential growth of computing times, we use a logarithmic scale. It is seen, that actual running times conform with the expected core hardness.

However the model (8) should be taken with some caution. Data instances may easily be constructed such that there is a small average weight and clustering, although the instances are very hard to solve. As an example choose the items as

$$p_j = w_j = 2j, \quad j = 1, \ldots, 50, \tag{11}$$

and assume that c is odd. The χ-value of this core is zero meaning that the expected core hardness is $\mathcal{H} = 0$. Still any branch-and-bound algorithm will have to enumerate all 2^{50} variations of the solution vector, since no bounding stops the process. However for randomly generated data instances, the model is very suitable, giving a good estimate

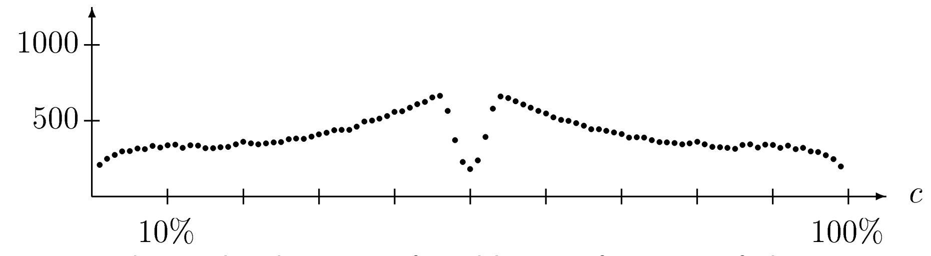

Figure 3: Expected core hardness $\mathcal{H}$ of problem as function of the capacity c. Weakly correlated instances, $n = 3000, R = 100$.

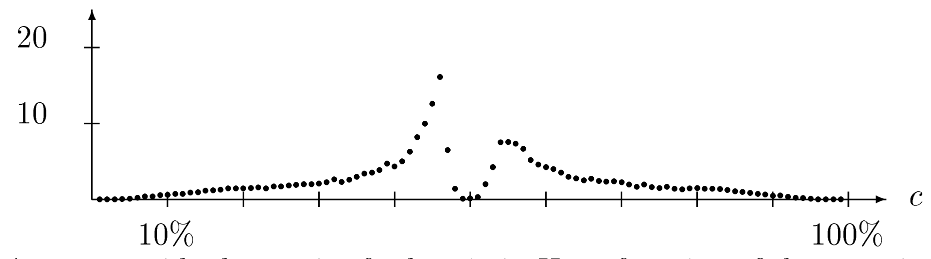

Figure 4: Average residual capacity for heuristic H, as function of the capacity c. Weakly correlated instances, $n = 3000, R = 100$.

of the expected core hardness. But the expected core hardness only reflects the actual running times for a *large* series of instances. Single runs may be solved very fast if the items exactly fit into the capacity.

4 Expected core hardness

We will now apply the model developed for predicting computational times of very large data instances as a function of the capacity. These instances are so hard, that we cannot make a direct comparison like between Figure 3 and 5, since solution times of several days occur for many instances, when solved by MT2.

Figures 6 to 13 give the expected core hardness for uncorrelated, weakly correlated, strongly correlated, and subset-sum data instances as a function of the capacity c. In

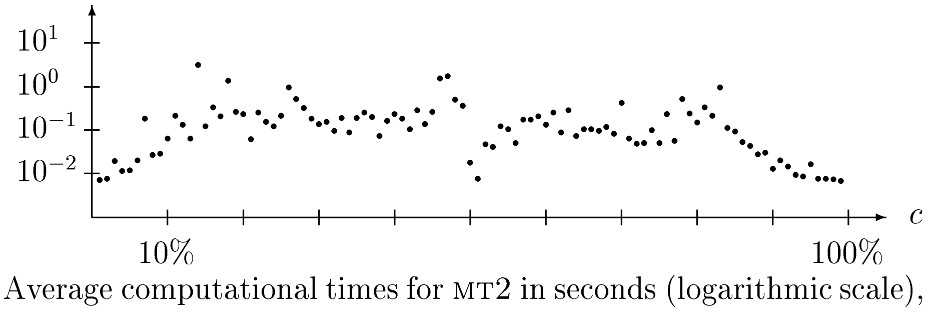

Figure 5: Average computational times for MT2 in seconds (logarithmic scale), as function of the capacity c. Weakly correlated instances, $n = 3000, R = 100$.

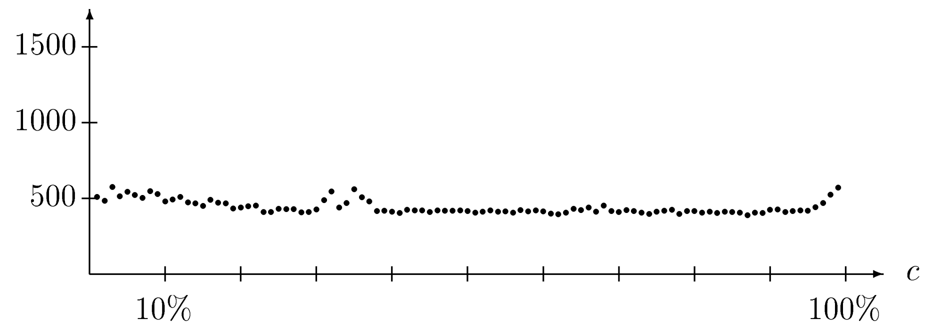

Figure 6: Expected core hardness, uncorrelated instances, $n = 10000, R = 100$.

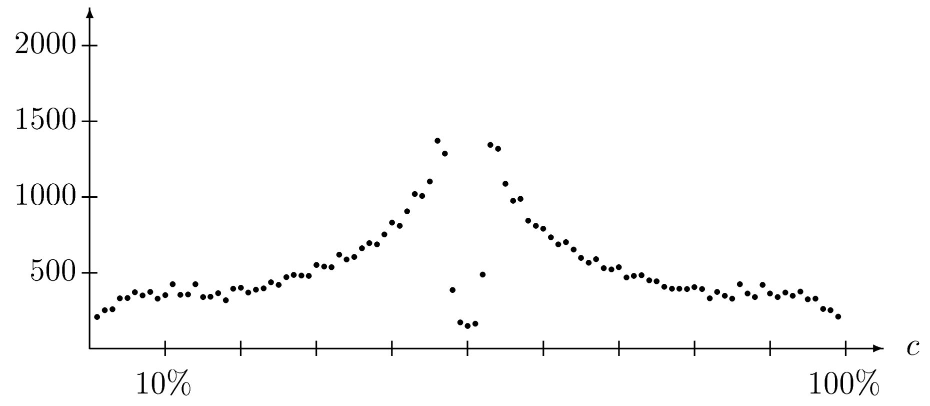

Figure 7: Expected core hardness, weakly correlated instances, $n = 10000, R = 100$.

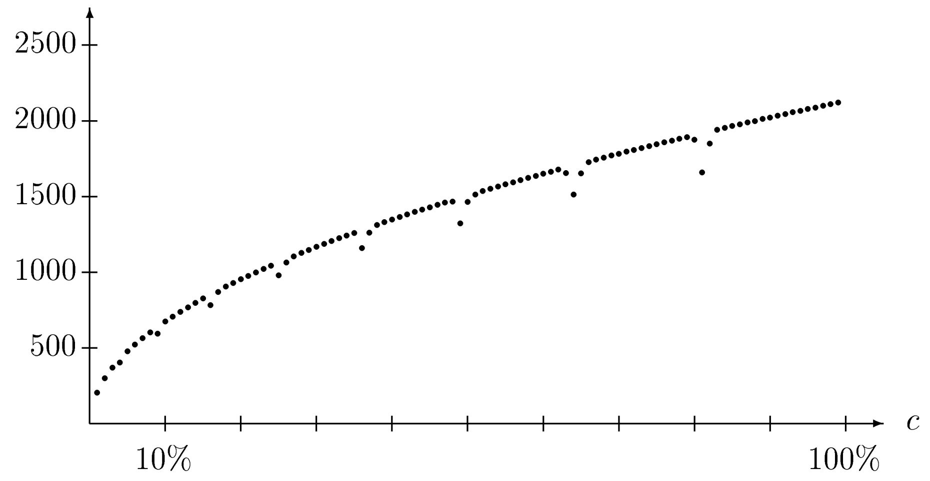

Figure 8: Expected core hardness, strongly correlated instances, $n = 10000, R = 100$.

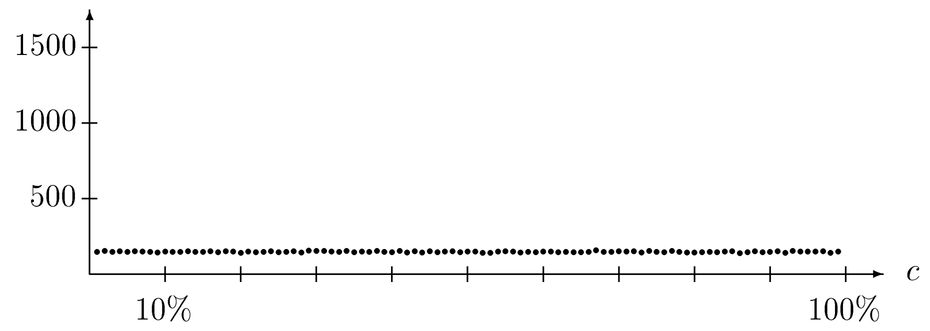

Figure 9: Expected core hardness, subset-sum instances, $n = 10000, R = 100$.

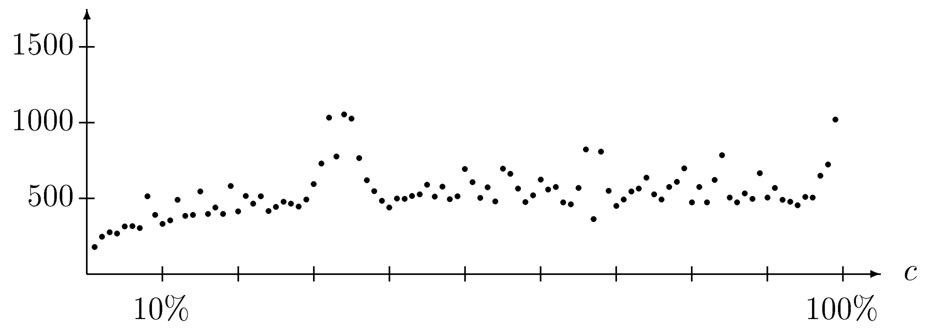

Figure 10: Expected core hardness, uncorrelated instances, $n = 100000, R = 100$.

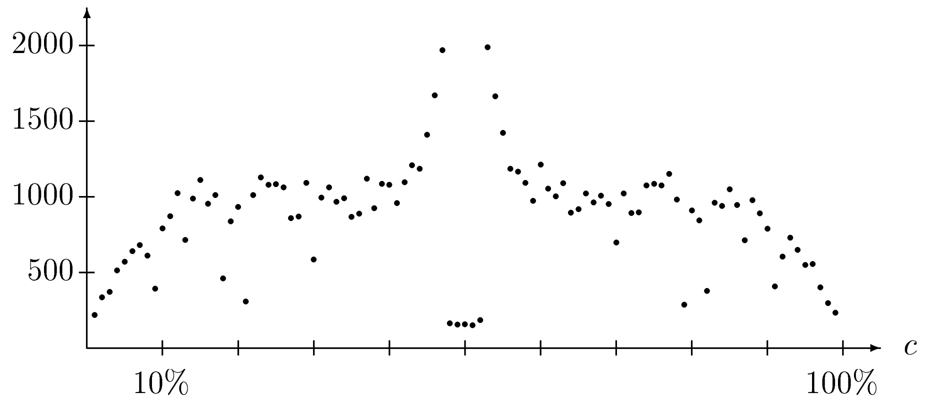

Figure 11: Expected core hardness, weakly correlated instances, $n = 100000, R = 100$.

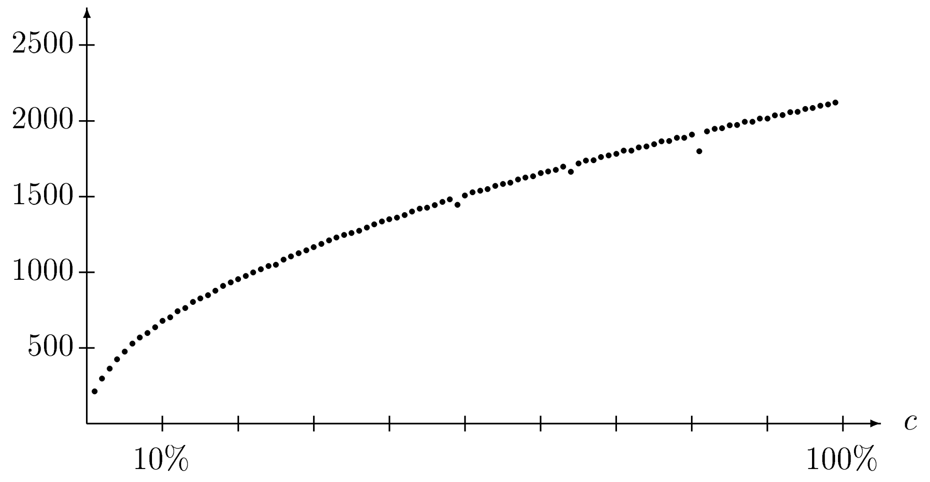

Figure 12: Expected core hardness, strongly correlated instances, $n = 100000, R = 100$.

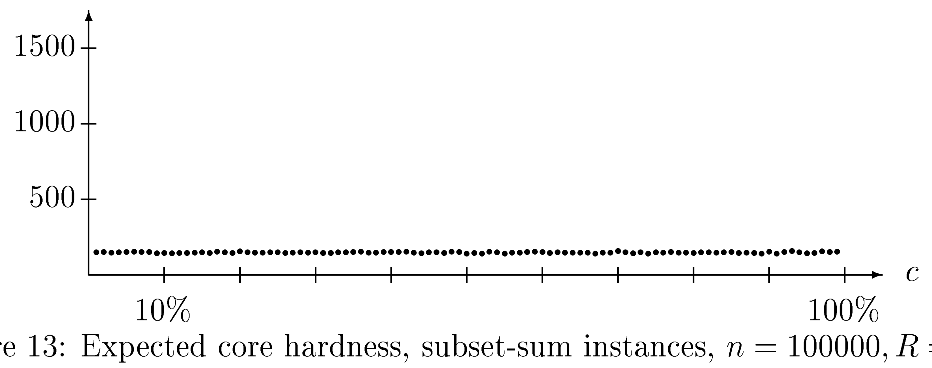

Figure 13: Expected core hardness, subset-sum instances, $n = 100000, R = 100$.

figures 6 to 9 we consider instances of size $n = 10000$ while figures 10 to 13 consider instances with $n = 100000$. All figures are based on the data range $R = 100$.

To start with the weakly correlated data instances (Figure 7, 11), it is seen that the easiest instances occur for c chosen as half of the weight sum, while almost any other instances are very hard. The minimum at $c = 50\%$ is due to a core of items with efficiencies $e_j = 1$, meaning that there is no clustering of the involved weights.

The uncorrelated data instances (Figure 6,10) are considerably easier than the weakly correlated instances. A peak at $c = 35\%$ is noticed, but it is of much lower extent than in the weakly correlated instances. Again this peak is around efficiencies $e_j = 1$. As the size of the data instance grow to $n = 100000$ the graph gets more turbulent, meaning that it is very difficult to predict the running times.

Strongly correlated instances (Figure 8,12) are very hard, as noticed by Balas and Zemel (1980) and Martello and Toth (1988). The core hardness grows with the capacity, as the average weight of the core increases. The peaks at the graph are noise due to the division of R in 10 intervals I_i when deriving χ. The core hardness grows smoothly with the capacity.

Finally the subset-sum data instances (Figure 9,13) are very easy for any capacity chosen. This could be expected, since there is no clustering at all: The core hardness only depends of the average weight of an instance.

We may draw several conclusions from these observations:

- For algorithms solving a core problem, the difficulty of a data instance is highly dependent on the capacity c. Traditionally only the correlation and data size was varied in computational experiments, while the capacity was fixed at half of the weight sum (Balas and Zemel 1980, Martello and Toth 1988, Pisinger 1994a, 1994b). Fayard and Plateau (1982) are using capacities $c = 20\%, 50\%, 80\%$ for their testings, but none of these values are representative for the hardness of a problem.

- If the capacity is chosen as half of the weight sum, we end up with the easiest possible data instances for all weakly correlated instances. Thus the comprehensive comparative tests in Martello and Toth (1990) are of little use. We cannot draw any conclusions from these tests.

- Balas and Zemel (1980) proves that the heuristic H for solving the core problem finds an optimal solution with a probability that grows with increasing n and decreasing R (Appendix A). Thus for uncorrelated data instances of size $n = 100000$ and range $R = 100$ we should have a probability larger than 0.9999 for finding an optimal solution. However the results presented here indicate that large problems with small data range will have an extremely high clustering of the weights, contradicting the assumption by Balas and Zemel saying that the weights in the core are uniformly distributed. Thus we should not expect to find good solutions by using a heuristic algorithm for solving the core problem. Balas and Zemel do not recognize this problem, since their tests are based on problems with $c = 50\%$.

- We may explain the adequate behavior of MT2 for weakly correlated problems, when $c = 50\%$, as reported by Martello and Toth (1990) and Pisinger (1994a,1994b): The partitioning algorithm for finding the approximate core finds a core which is not necessarily symmetrical around b. Very asymmetrical cores end up in difficult instances, since the peak around $c = 50\%$ is so narrow.
- The presented observations are also valid for other types of Knapsack Problems like the Bounded Knapsack Problem, or Multiple Knapsack Problem. Thus testing with capacity $c = 50\%$ should be avoided for these problems in order to evade trivial problems.

All the presented graphs have been given for the data range $R = 100$. Similar results hold for $R = 1000$ or $R = 10000$, although the instances must be 10 (resp. 100) times lager to obtain the same χ-value. The expected core hardnesses $\mathcal{H}$ for different data ranges are not directly comparable: They should merely be taken as an indication of which capacities may cause problems, than as absolute running times.

5 New method for testing algorithm

In order to test algorithms for the Knapsack Problem more thoroughly, we propose a new way of constructing test instances. First we notice that due to the stochastic nature of the Knapsack Problem, the applied series of test instances should be large – much larger than the previously used 10 or 20 instances (Balas and Zemel 1980, Martello and Toth 1990, Pisinger 1994a,1994b). Thus we propose series of $S = 1000$ instances for each type of problem.

Moreover the capacity should be uniformly scattered among the possible weight sums, so that the capacity-dependent behavior is annihilated. For test instance i we choose the capacity as

$$c = \frac{i}{S+1} \sum_{j=1}^{n} w_j, \tag{12}$$

meaning that a variety of different capacities are tested as i runs from 1 to S. Finally the test instances should be reproducible by other authors. The applied algorithm for generating test instances is thus given in appendix B, with some checksums on the optimal solutions.

We consider four different algorithms for solving the Knapsack Problem. Although they are all based on solving a core problem exactly, they differ essentially in several respects. The FPK79 algorithm (Fayard and Plateau 1982) is based on a heuristic solution of the core problem, while the MT2 algorithm by Martello and Toth (1990) solves a core problem of fixed size to optimality through branch-and-bound. The EXPKNAP algorithm (Pisinger 1994a) is using a depth-first branch-and-bound algorithm for solving the core problem like MT2, but the core is expanded by need, as the branching propagates. Finally the MINKNAP algorithm (Pisinger 1994b) is using dynamic programming for solving an expanding core. Since the breadth-first search ensures that all variations of the solution

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	0.00	0.00	0.00	0.00	0.00	0.00	28.78	28.13	56.02	0.00	0.00	0.01
300	0.00	0.00	0.00	0.00	0.00	0.00	—	—	—	0.01	0.01	0.02
1000	0.00	0.00	0.00	0.08	0.00	0.01	—	—	—	0.06	0.08	0.11
3000	0.01	0.01	0.01	—	0.01	0.01	—	—	—	0.49	0.62	0.86
10000	0.50	0.03	0.04	—	0.16	0.05	—	—	—	5.26	5.68	8.08
30000	—	0.10	0.11	—	—	0.14	—	—	—	—	—	—
100000	—	0.47	0.42	—	—	3.01	—	—	—	—	—	—

Table II: Total computing time in seconds, FPK79. Average of 1000 instances.

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	0.00	0.00	0.00	0.00	0.00	0.00	8.07	8.31	16.24	0.00	0.00	0.01
300	0.00	0.00	0.00	0.00	0.00	0.01	—	—	—	0.00	0.00	0.02
1000	0.00	0.00	0.01	0.00	0.01	0.02	—	—	—	0.00	0.00	0.02
3000	0.00	0.01	0.02	0.07	0.01	0.06	—	—	—	0.00	0.00	0.02
10000	0.02	0.03	0.06	—	0.02	0.14	—	—	—	0.01	0.01	0.03
30000	—	0.06	0.17	—	0.19	0.23	—	—	—	0.03	0.03	0.05
100000	—	0.27	0.52	—	—	0.44	—	—	—	0.12	0.13	0.14

Table III: Total computing time in seconds, MT2. Average of 1000 instances.

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	0.00	0.00	0.00	0.00	0.00	0.00	42.21	45.14	126.66	0.00	0.00	0.02
300	0.00	0.00	0.00	0.00	0.01	0.01	—	—	—	0.00	0.00	0.02
1000	0.00	0.00	0.01	0.00	0.02	0.06	—	—	—	0.00	0.00	0.02
3000	0.00	0.01	0.02	0.00	0.01	0.25	—	—	—	0.00	0.00	0.03
10000	0.01	0.03	0.11	0.01	0.02	0.65	—	—	—	0.01	0.01	0.05
30000	0.04	0.06	0.39	0.04	0.05	0.63	—	—	—	0.04	0.04	0.05
100000	0.17	0.19	0.91	0.17	0.18	0.39	—	—	—	0.15	0.15	0.15

Table IV: Total computing time in seconds, EXPKNAP. Average of 1000 instances.

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.25	0.00	0.00	0.05
300	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.14	1.44	0.00	0.00	0.05
1000	0.00	0.00	0.00	0.00	0.00	0.01	0.06	0.61	7.43	0.00	0.01	0.05
3000	0.00	0.01	0.01	0.00	0.01	0.03	0.22	2.33	29.10	0.00	0.01	0.06
10000	0.01	0.01	0.02	0.01	0.01	0.05	1.00	10.47	125.12	0.01	0.01	0.06
30000	0.03	0.03	0.06	0.04	0.04	0.07	3.22	38.04	464.26	0.03	0.03	0.08
100000	0.10	0.10	0.16	0.15	0.14	0.15	14.22	152.51	1756.78	0.11	0.12	0.16

Table V: Total computing time in seconds, MINKNAP. Average of 1000 instances.

vector corresponding to the core have been tested before a new item is included in the core, this algorithm enumerates the smallest possible core.

The codes used are obtained from the mentioned references. Algorithm FPK79 had to be corrected a few places, since the reduction part of the code was wrong. The code by Balas and Zemel (1980) was not available for this test.

Table II to V gives the total running times for the four algorithms on a HP-9000/730 computer. The instances marked with a "—" were stopped after 10 hours, when there did not seem to be any progress in the solution process.

It is seen, that FPK79 and MT2 have substantial problems for all low-ranged weakly correlated instances of medium and large size. Also some low-ranged uncorrelated instances cause problems. On the other hand both EXPKNAP and MINKNAP show a stable behavior, with all running times within one second (the strongly correlated instances excepted). Only MINKNAP is able to solve large strongly correlated instances.

It is interesting to note that although MT2 and EXPKNAP both are using depth-first branch-and-bound techniques, EXPKNAP is not sensitive to the difficulty of the core, since the core is expanded by need. The MINKNAP algorithm is using dynamic programming for solving the core, so it is not sensitive to clustered weights: equal states are simply fathomed through a dominance test.

The last table (Table VI) gives the percentage of tested items by the reduction algorithm of MINKNAP. Since this reduction is performed by need, the values of Table VI indicates how many items need to be considered in order to solve the problem. If the percentage is low, then the problem is well suited for solving a core problem, while high percentages means that you have to reduce almost all items anyway, implying that a heuristic solution to the core problem may be more suitable (Pisinger 1994c).

If Table VI is compared to the running times of MT2, it is seen that MT2 is performing worst, when most effort could be saved. The stable running times of MT2 are for instances where solving a core problem is almost useless. Thus we may conclude, that solving a core problem of fixed size generally makes the instance harder to solve than using a complete reduction followed by enumeration of the remaining instance.

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	74	89	92	64	97	99	84	89	94	24	32	38
300	54	85	90	25	94	98	83	85	89	10	13	16
1000	19	79	88	6	76	97	84	86	87	4	5	6
3000	3	66	86	3	33	95	85	85	86	2	2	2
10000	1	31	84	1	5	85	88	86	86	1	1	1
30000	0	7	72	1	1	48	87	86	88	0	0	0
100000	0	1	51	0	0	11	84	85	84	0	0	0

Table VI: Percentage of items, which were tested by the reduction algorithm of MINKNAP. Average of 1000 instances.

6 Conclusions

We have seen that the core hardness of a data instance depends strongly on the capacity c, and that the hitherto applied test capacities lead to the easiest possible problems for weakly correlated data instances. Solving a core problem exactly may lead to seriously worse running times than if a more naive technique was used, but these problems may be avoided by using an expanding core, or through dynamic programming. Finally a new and more thorough way of testing Knapsack Problems has been proposed. The technique has been used for four algorithms — all based on a core problem — showing that EXPKNAP and MINKNAP are the only algorithms which have a stable behavior.

Appendix A: Heuristic *H*

Heuristic H by Balas and Zemel (1980) is a simple exchange algorithm for the core problem, which successively removes an item i and replaces it by one or two other items j, b in order to obtain a *filled* knapsack. In spite of the simple structure, Balas and Zemel are able to prove some interesting properties on the quality of the solution found by heuristic H.

Assume that the core items are indexed by $1, \ldots, m$, and that the capacity of the core problem is $\tilde{c}$. The break item in the core is denoted b, and the residual capacity by filling the knapsack with items $j < b$ is $r = \tilde{c} - \sum_{j=1}^{b-1} w_j$. Define the extreme core weights as $\text{MIN} = \min_{j=1,\ldots,m} w_j$ and $\text{MAX} = \max_{j=1,\ldots,m} w_j$. Heuristic H may then be sketched as:

```
procedure H; {Find the packing, which fills the knapsack most}
for $i := 1$ to $b-1$ do $x_i := 1$; rof;
for $i := b$ to $m$ do $x_i := 0$; rof;
$d := r$; {$d$ is the hitherto smallest obtained residual capacity}
for $i := 1$ to $b-1$ do
  $x_i := 0$;  $\tilde{r} := r + w_i$; {Remove item $i$ and eventually insert item $b$}
  if $(\tilde{r} > \text{MAX})$ then $x_b := 1$;  $\tilde{r} := \tilde{r} - w_b$; fi;
  for $j := b+1$ to $m$ do
    $x_j := 1$;  $\overline{r} := \tilde{r} - w_j$; {Insert an item $j$}
    if $(0 \le \overline{r} < d)$ then $d := \overline{r}$; Save the solution. fi;
    $x_j := 0$;
  rof;
  $x_i := 1$;  $x_b := 0$;
rof
```

Balas and Zemel shows the following results for heuristic H: Let $V_0 = \text{MAX} - \text{MIN} + 1$ be the span in the weights, and let V_1 be the number of items $i = 1, \ldots, b-1$ satisfying that $\text{MIN} \le \tilde{r} \le \text{MAX}$. Further assume that the weights w_j are independent random variables uniformly distributed in $[\text{MIN}, \text{MAX}]$.

	$m-b$			
V_1	10	20	30	50
10	0.65	0.88	0.96	0.986
15	0.80	0.97	0.993	0.999
20	0.89	0.99	0.999	0.999

Table VII: Probability P for obtaining a filled solution by heuristic H, as function of V_1 and $m-b$. The probabilities are for $V_0 = 100$ corresponding to data range $R = 100$.

Proposition 1 The probability P that the heuristic H finds a filled solution (i.e. a solution with residual capacity $\overline{r} = 0$) is

$$P = 1 - \left(1 - \frac{V_1}{V_0}\right)^{m-b}. \tag{13}$$

See Balas and Zemel (1980) for a proof. The interesting consequence of Proposition 1 is that under reasonable assumptions, the probability for finding a filled knapsack grows exponentially with the core size m, and increases with smaller data-range R since $V_0 \leq R$. Table VII gives the expected probability for finding a filled solution by algorithm H. The Table is obtained from Balas and Zemel (1980).

Proposition 2 The probability for a filled solution found by heuristic H being an optimal solution to the Knapsack Problem is bounded below by a strictly increasing function $Q(n)$ satisfying that

$$\lim_{n\to\infty} Q(n) = 1. \tag{14}$$

Thus the probability for a filled solution obtained by heuristic H being an optimal solution grows with the size n of the instance.

However the results presented in this paper, indicate that Proposition 1 and 2 are of little use, even for uncorrelated data instances of large size. The problem is, that for large instances the core may degenerate, implying that V_1 becomes close to zero. Heuristic H is thus not able to find a filled solution, although it would be optimal for the global problem.

Appendix B: Generating test data

In order to let other authors generate the same test instances as applied in section 5, we here include an algorithm for generating the test data.

The standard LRAND48 generator of the C library is used for generating pseudo-random profits and weights. The LRAND48 generator is using the well-known linear congruential algorithm, which generates a series of numbers $X_1, X_2, X_3, \ldots$ as

$$X_{i+1} = (aX_i + d) \bmod m, \tag{15}$$

where $a = 25214903917$, $d = 11$ and $m = 2^{48}$. The first 31 bits of the number are returned for LRAND48. A seed s for the algorithm is given by procedure SRAND48 as:

$$X_0 = s \cdot 2^{16} + 13070. \tag{16}$$

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	283	67	410	505	591	257	348	202	681	391	111	897
300	717	402	272	333	188	717	481	45	443	952	924	381
1000	802	589	48	895	956	850	961	129	307	461	873	939
3000	932	320	780	193	942	146	415	225	718	545	265	342
10000	737	590	269	577	328	398	847	210	370	167	160	940
30000	689	846	820	794	153	117	507	361	320	457	801	490
100000	926	85	646	749	471	136	186	956	242	606	366	292

Table VIII: Checksums for optimal solutions given as $\sum_{i=1}^{S} z_i$ mod 1000.

For a given data range R, instance size n, and problem type t (uc, wc, sc, ss) we constructed $S = 1000$ different data instances as follows:

```
**procedure** testinstance($n, R, t, i$);  {*Generate test instance $i$, $1 \leq i \leq S$*}
$W := 0$; $R' := \lfloor R/10 \rfloor$; SRAND48($i$);  {*Use $i$ as seed for the sequence of random numbers*}
**for** $j := 1$ **to** $n$ **do**
    $w_j :=$ (LRAND48 **mod** $R) + 1$;   $W := W + w_j$;
    **case** $t$ **of**
        uc: $p_j :=$ (LRAND48 **mod** $R) + 1$;
        wc: $p_j := w_j - R' +$ (LRAND48 **mod** $(2R' + 1)$);
               **if** $(p_j \leq 0)$ **then** $p_j := 1$; **fi**;
        sc: $p_j := w_j + 10$;
        ss: $p_j := w_j$;
    **esac**;
**rof**;
$c := \lfloor (i * W)/(S + 1) \rfloor$;
**if** $(c \leq R)$ **then** $c := R + 1$; **fi**;
```

The **if** -statement in weakly correlated instances ensures that p_j is a positive integer, while the last line of the algorithm ensures that equation (2) is satisfied. Checksums of the optimal solutions and capacities are given in Table VIII and IX.

An interesting problem to investigate further is the instance $n = 100000$, $R = 10000$, $t =$ uc, $S = 500$ and $s = 157$. Here MT2 finds the solution 323792911 while EXPKNAP and MINKNAP both finds the solution 323792912.

	UC			WC			SC			SS		
$n \setminus R$	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
100	208	739	745	208	739	745	391	128	903	391	128	903
300	692	620	220	692	620	220	952	924	381	952	924	381
1000	653	696	125	653	696	125	461	873	939	461	873	939
3000	679	793	42	679	793	42	545	265	342	545	265	342
10000	32	850	127	32	850	127	167	160	940	167	160	940
30000	417	468	111	417	468	111	457	801	490	457	801	490
100000	933	384	858	933	384	858	606	366	292	606	366	292

Table IX: Checksums for applied capacities given as $\sum_{i=1}^{S} c_i$ mod 1000.

References

Balas, E. and E. Zemel, "An Algorithm for Large Zero-One Knapsack Problems," *Operations Research*, 28 (1980), 1130-1154.

Dembo, R.S. and P.L. Hammer, "A Reduction Algorithm for Knapsack Problems," *Methods of Operations Research*, 36 (1980) 49-60.

Fayard, D. and G. Plateau, "An Algorithm for the Solution of the 0-1 Knapsack Problem," *Computing*, 28 (1982), 269-287.

Ingargiola, G.P. and J.F. Korsh, "A Reduction Algorithm for Zero-One Single Knapsack Problems," *Management Science*, 20 (1973), 460-463.

Martello, S. and P. Toth, "An Upper Bound for the Zero-One Knapsack Problem and a Branch and Bound algorithm," *Europan Journal of Operational Research*, 1 (1977), 169-175.

Martello, S. and P. Toth, "A New Algorithm for the 0-1 Knapsack Problem," *Management Science*, 34 (1988), 633-644.

Martello, S. and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, England, 1990.

Nauss, R.M., "An Efficient Algorithm for the 0-1 Knapsack Problem," *Management Science*, 23 (1976), 27-31.

Pisinger, D., "An expanding-core algorithm for the exact 0-1 Knapsack Problem," To appear in *European Journal of Operational Research* (1994a).

Pisinger, D., "A minimal algorithm for the 0-1 Knapsack Problem," *DIKU, University of Copenhagen, Denmark,* Report 94/23 (1994b).

Pisinger, D., "Avoiding anomalies in the mt2 algorithm by Martello and Toth," To appear in *European Journal of Operational Research* (1994c).