WMC 2006

Leiden – Lorentz Center – 21 July 2006

Dipartimento di Informatica, Sistemistica e Comunicazione

Università degli Studi di Milano – Bicocca

A Membrane Algorithm for the Min Storage problem

Alberto Leporati – leporati@disco.unimib.it

Dario Pagani – dario.pagani@gmail.com

Talk outline

- The ConsComp and Min Storage problems
- Computational properties of Min Storage (NP-hardness, 2-approximability)
- Nishida's membrane algorithm for TSP
- A membrane algorithm for Min Storage:
  - first try: crossover and mutation
  - second try: simple local optimization
- Some traditional heuristics for Min Storage
- Computer experiments:
  - generation of random instances
  - comparison with traditional heuristics

Conservative computations

Let:

- G be an n-input/m-output gate (function)
- $S_{in} = \langle \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_k \rangle$ be a sequence of input values
- $S_{out} = \langle G(\underline{x}_1), G(\underline{x}_2), \ldots, G(\underline{x}_k) \rangle$ be the corresponding sequence of output values

For $i = 1,2,\ldots,k$, we define:

$$e_i = E_n(\underline{x}_i) - E_m(G(\underline{x}_i))$$

energy function

If $\sum_{i=1}^{k} e_i = 0$, then we say the computation is conservative

Gates with storage unit

- In a conservative computation it may be $e_i > 0$ or $e_i < 0$ for some $i \in \{1,2,...,k\}$
- We assume to use gates equipped with a storage unit, able to store a limited amount C of energy
- We call C the capacity of the gate
- Remark: without loss of generality, we can assume that all e_i's are integer values

Stored energy during a computation

- If $G(\underline{x}_1), G(\underline{x}_2), \ldots, G(\underline{x}_k)$ are computed in this order, then the energy stored into the gate at each step is:

$$st_0 = 0$$
$$st_1 = e_1 = E_n(\underline{x}_1) - E_m(G(\underline{x}_1))$$
$$st_2 = e_1 + e_2 = e_1 + E_n(\underline{x}_2) - E_m(G(\underline{x}_2))$$
$$\vdots$$
$$st_k = e_1 + e_2 + \ldots + e_k = 0$$

- Question: is there a permutation $\pi \in S_k$ such that the computation of $G(\underline{x}_{\pi(1)}), G(\underline{x}_{\pi(2)}), \ldots, G(\underline{x}_{\pi(k)})$ is C-feasible?

The ConsComp decision problem

- Instance:
  - a set $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ of integer numbers such that $e_1+e_2+\ldots+e_k = 0$
  - an integer number $C > 0$
- Question: is there a permutation $\pi \in S_k$ such that

$$0 \le \sum_{j=1}^{i} e_{\pi(j)} \le C$$

i-th prefix sum

for each $i \in \{1,2,\ldots,k\}$?

- Theorem: ConsComp is NP-complete

  Proof: reduction from Partition (ask for details!). ■

Complexity of ConsComp

- Theorem: ConsComp is NP-complete in the strong sense.

  Proof: reduction from (3-)Partition. ■
- We conjecture that the problem remains NP-complete even with constant elements in the instance
- Notice that Bin Packing and 3-Partition with a constant number of element types are polynomial. Of course, the proofs for these problems do not work for ConsComp

The Min Storage optimization problem

Optimization problem whose natural decision version is ConsComp

- name: Min Storage
- instance: a set $\{e_1, e_2, \ldots, e_k\}$ of integer numbers such that $e_1+e_2+\ldots+e_k = 0$
- solution: a permutation $\pi \in S_k$ such that

$$\sum_{j=1}^{i} e_{\pi(j)} \geq 0$$

for each $i \in \{1,2,\ldots,k\}$

- measure: $\max_{1 \leq i \leq k} \sum_{j=1}^{i} e_{\pi(j)}$

Complexity of Min Storage

- Min Storage $\in$ NPO
- Since ConsComp is strongly NP-complete, Min Storage is NP-hard in the strong sense
- Trivial bounds for the optimal solution:
  - upper bound: $\frac{1}{2} \cdot \sum_{i=1}^{k} |e_i|$
  - lower bound: $\max_{1 \le i \le k} |e_i|$
- It is easily shown that the problem is 2-approximable
  - best known upper bound: $2 \cdot \max_{1 \le i \le k} |e_i| - 1$
- Open problem: is it 3/2-approximable?

Complexity of Min Storage

- A 2-approximation algorithm for Min Storage:

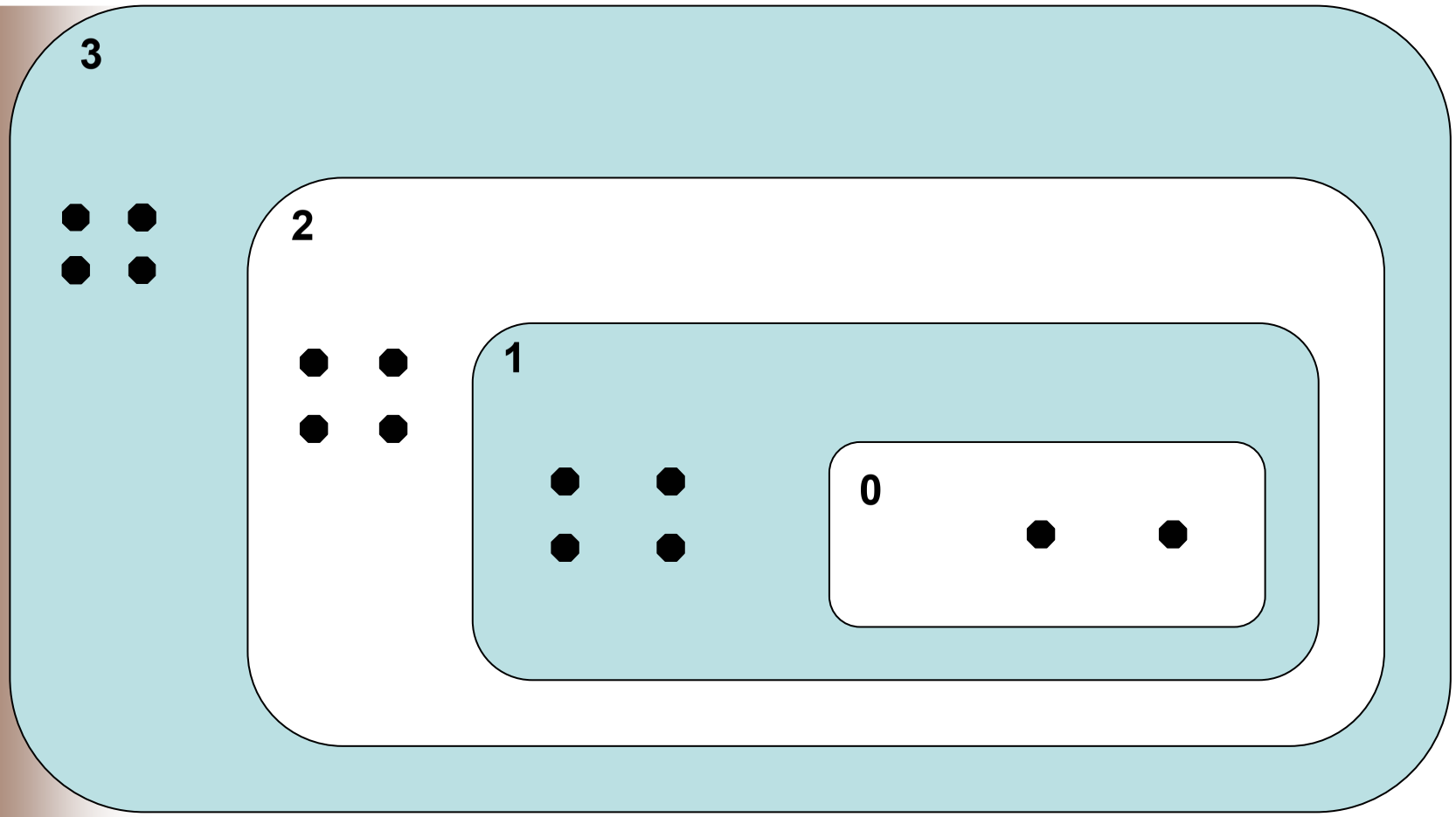
```
APPROX MIN STORAGE(E)
M ← max_{1≤i≤k} |e_i|
E_p = E_n = ∅
for i ← 1 to k
    do if e_i ≥ 0
          then E_p = E_p ∪ {e_i}
          else E_n = E_n ∪ {e_i}
max ← st ← 0
while E_p ≠ ∅
    do if st < M
          then x ← an element of E_p
               st ← st + x
               if st > max then max ← st
               E_p = E_p \ {x}
          else x ← an element of E_n
               st ← st + x
               E_n = E_n \ {x}
return max
```

- There is no FPTAS for Min Storage
- Open problem: is there a PTAS?

Membrane algorithms

- Introduced by Nishida, and applied to TSP
- Inspired by Membrane Computing
- Composed by three components:
  - a linear collection of regions separated by nested membranes
  - local optimization sub-algorithms that works upon a small number of candidate solutions
  - a trasport mechanism to move candidate solutions to the immediately inner or outer region
- Notation:
  - M = number of nested membranes
  - D = number of iterations before halting

Membrane algorithms



Nishida's algorithm for TSP

- Local optimization subalgorithm for region 0 is tabu search
  - exchanges two nodes in candidate solution (throws away previously considered solutions)
- In the other regions:
  - edge exchange crossover: combines two solutions to produce two new solutions
  - mutation, performed in region i with probability i/M
- Halting condition: prefixed number of iterations

Nishida's algorithm for TSP

- 20 tests on *eil51*, with $D = 40000$. Optimal solution is 426

ALGORITMO	MA2	MA10	MA30	MA50	MA70	SA
Best	440	437	433	429	429	430
Average	544	450	442	435	434	438
Worst	786	457	750	444	443	445

- 20 tests on *kroA100*, with $D = 100000$. Optimal solution is 21282

ALGORITMO	MA2	MA10	MA30	MA50	SA
Best	24524	22319	217770	21651	21369
Average	32973	23422	23200	22590	21763
Worst	49667	24862	23940	24531	22564

Membrane alg. for Min Storage

First try: MA4MS

- Fitness measure:

$$F(\pi) = \begin{cases} \max_{1 \le i \le k} \sum_{j=1}^{i} e_{\pi(j)} & \text{if } \sum_{j=1}^{i} e_{\pi(j)} \ge 0 \quad \forall i \in \{1,2,\ldots,k\} \\ \sum_{i=1}^{k} |e_i| - \text{NumVPS}_{\pi} & \text{otherwise} \end{cases}$$

- Same structure Nishida's algorithm for TSP
  - membrane structure, number of solutions
- Local optimization subalgorithms:
  - region 0: LocalSearch4MS
  - other regions: Partially Matched Crossover (PMX), followed by Mutation

LocalSearch4MS

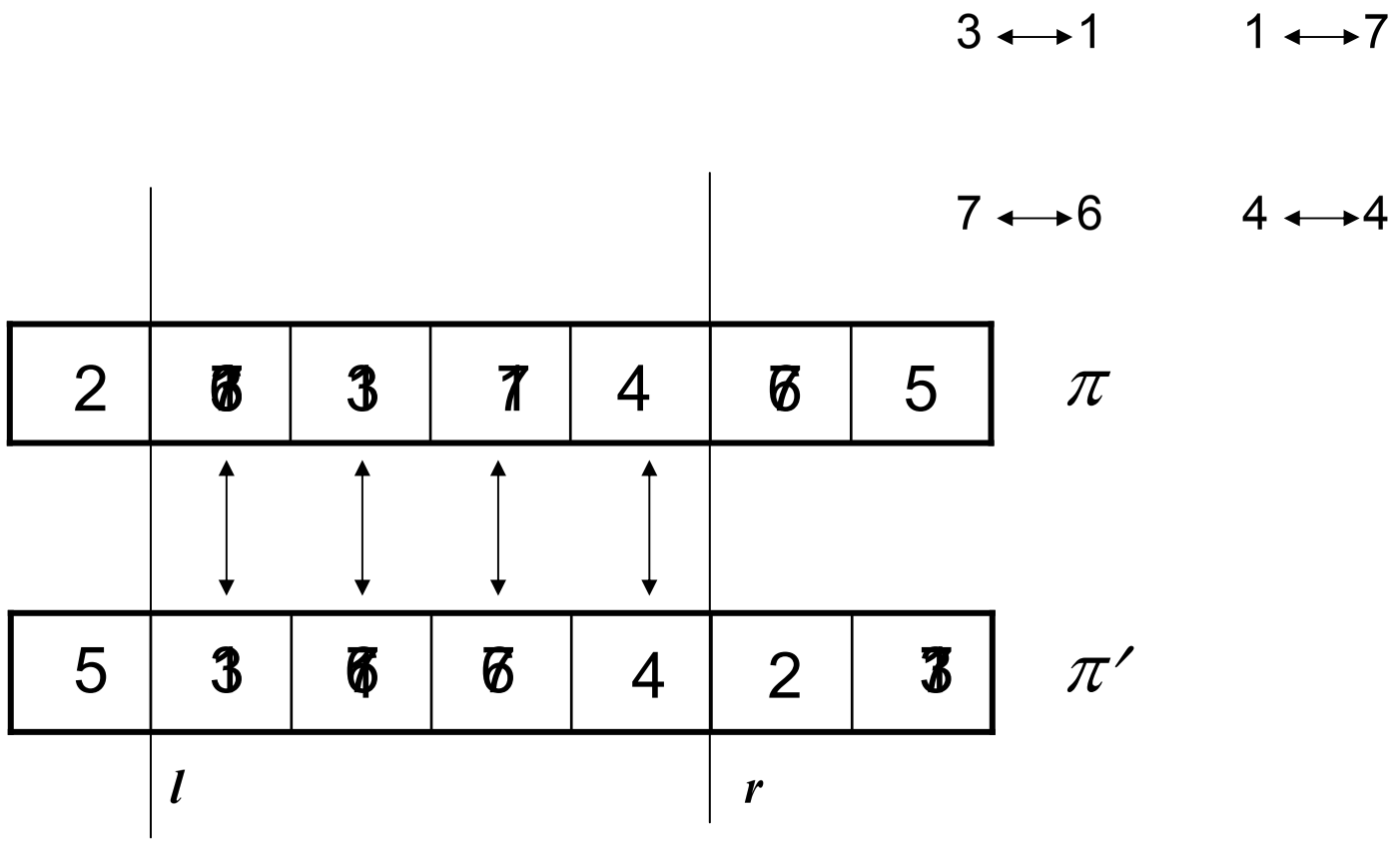
- Neighborhood of candidate solution π, w.r.t. position α: the set of $k-1$ solutions

$$Neigh(\pi,\alpha) = \bigcup_{i \neq \alpha} \{\pi_{i,\alpha}\}$$

where $\pi_{i,\alpha}$ is obtained from π by exchanging elements in positions i and α

- LocalSearch4MS looks for the best (i.e., lowest fitness) solution in $Neigh(\pi,\alpha)$
  - α is chosen as the first position for which the prefix sum is negative (if any)
  - if all prefix sums are non negative, α is chosen at random

Partially Matched Crossover



Mutation

- After applying PMX, to each of the new solutions we apply Mutation
- in region i, the probability to apply Mutation is i/M
- Mutation simply chooses two positions at random (with uniform distribution) and exchanges the two elements

π	2	6	3	1	4	7	5

Generation of instances

- Problem: generate k-tuples $(X_1, X_2, \ldots, X_k)$ such that

$$\sum_{i=1}^{k} X_i = 0$$

in a uniform way, where $X_1, X_2, \ldots, X_k$ are discrete, independent random variables uniformly distributed over the set of integers $\{-M, \ldots, M\}$

- First idea: extract a k-tuple and discard it if $\sum_{i=1}^{k} X_i \neq 0$

If you approximate the distribution of $Y = \sum_{i=1}^{k} X_i$ with $N\left(0, k\frac{M(M+1)}{3}\right)$, we get: $\Pr[Y = 0] \approx 6.91 \cdot 10^{-8}$

for k=100 and M=10^6

Generation of instances

- Alternative idea: extract a $(k-1)$-tuple and discard it if

$$\sum_{i=1}^{k-1} X_i \notin \{-M, \ldots, M\}$$

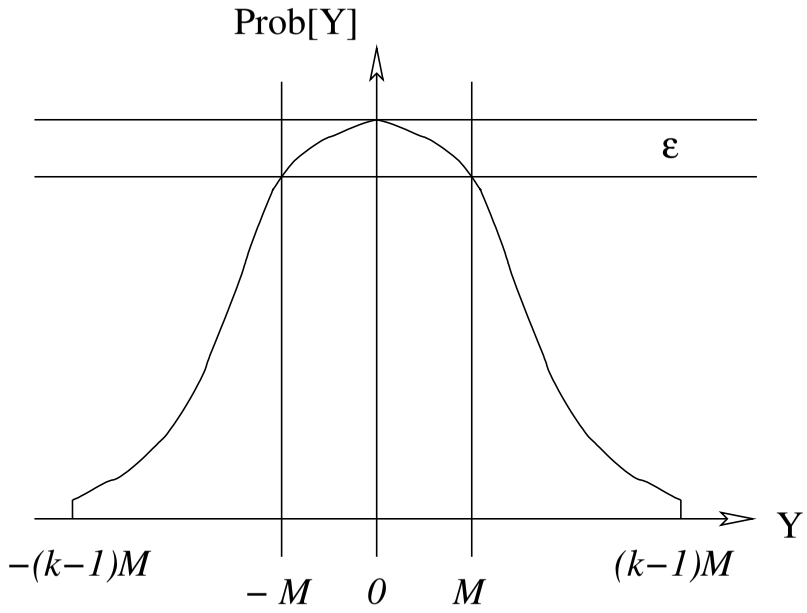
Approximating the distribution of $Y = \sum_{i=1}^{k-1} X_i$ with $N\left(0, (k-1)\frac{M(M+1)}{3}\right)$, we get:

$$\Pr\left[-M \le \sum_{i=1}^{k-1} X_i \le M\right] \approx 0.138$$

for k=100 and M=10^6

Error of approximation

- The probability distribution of $Y = \sum_{i=1}^{k-1} X_i$ is almost uniform:



- Example: $\varepsilon = \Pr[Y = 0] - \Pr[Y = M] \approx 1.04 \cdot 10^{-9}$ for k=100 and M=10^6

Coefficient of approximation

Given:

- a heuristic A
- an instance I of Min Storage

let:

- $c_A(I)$ be the value returned by A
- $opt(I)$ be the optimal solution

The performance ratio (coefficient of approximation) of A over I is:

$$app_A(I) = \frac{|c_A(I)|}{opt(I)}$$

Note that $app_A(I) \geq 1$, and the closer it is to 1 the better $c_A(I)$ is

Coefficient of approximation

- For large instances, we are not able to compute $opt(I)$
  - we use $\max_{1 \le i \le k} |e_i|$ instead
  - we obtain upper bounds to the coefficient of approximation
- We really compute the average coefficient of approximation:

$$app_{MA4MS} = \frac{1}{N} \sum_{i=1}^{N} \frac{F_i(\pi)}{opt_i}$$

where N is the number of tests

Some experiments with MA4MS

- 10000 tests with M=10 and D=50

K	AVG.COEFF.APPROX	VARIANCE
10	1.2052383	0.0433753
20	1.7645406	0.1412564
50	3.0863457	0.6402221
100	4.6576763	1.8045398

- 10000 tests with M=30 and D=150

K	AVG.COEFF.APPROX	VARIANCE
10	1.0875901	0.0139737
20	1.5258556	0.0582978
50	2.6124590	0.2444580
100	3.9430665	0.5004649

MA4MS with local search only

Second try: MA4MS LS

- we use LocalSearch4MS as a local optimization subalgorithm in every region
- 10000 tests with M=30 and D=150

K	AVG.COEFF.APPROX	VARIANCE
10	1.0032719	0.0002333
20	1.0093292	0.0003654
50	1.0600094	0.0045419
100	1.1978124	0.0162296

- In the following, we will use only MA4MS LS !

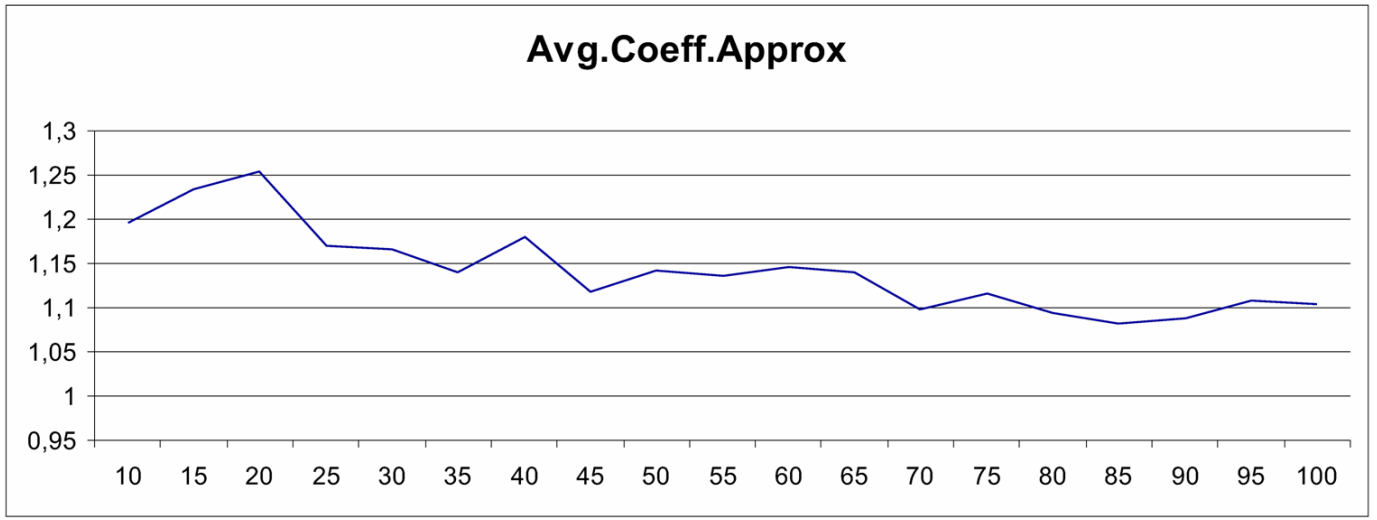
System's parameters

- we have tried to make M and D grow together…

M	D	Avg.Coeff.Approx
10	20	2.1003992
30	60	1.4055032
50	100	1.2228035
80	150	1.1003962
150	200	1.1066577
300	500	1.0214561

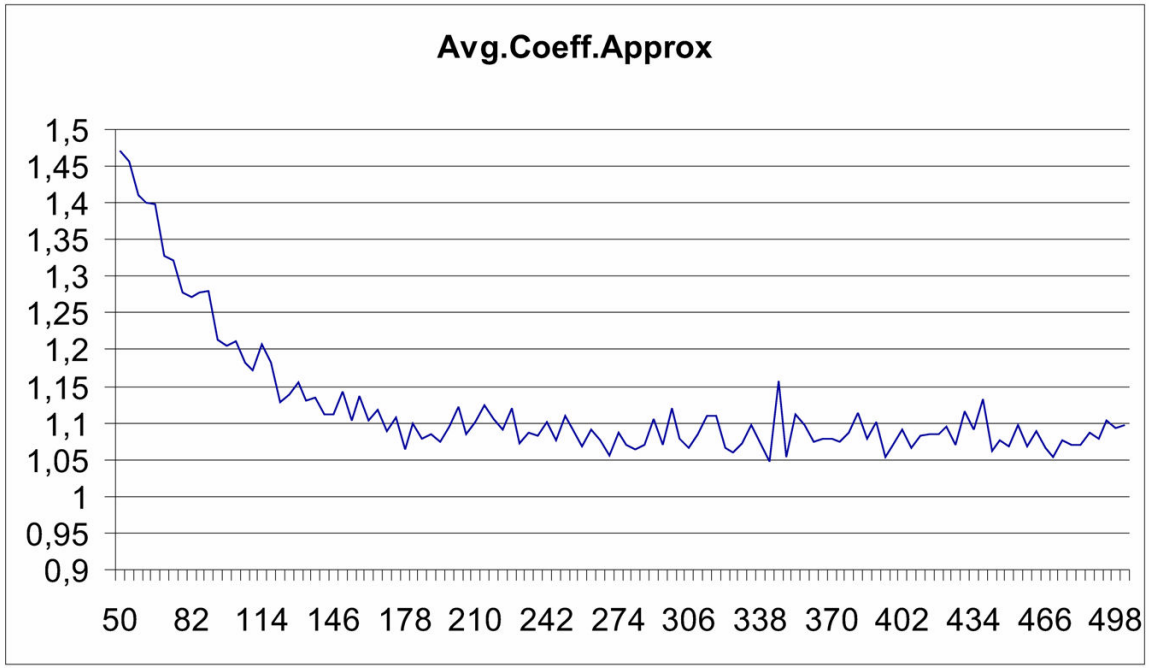
System's parameters

- ...as well as one at the time



- Average coefficient of approximation for M varying between 10 and 100, for D=150

System's parameters



- Average coefficient of approximation for D varying between 50 and 500, for M=80

Some heuristics for Min Storage

- The Greedy algorithm (time: $\Theta(k^2)$)
- Some $\Theta(k \log k)$ heuristics:
  - Min
  - Max
  - MaxMinMax
  - MinMaxMin
  - MinMaxMinMax
  - MaxMinMaxMin
- The Best Fit algorith

Some assumptions:

- all lists are implemented through arrays
- removal of a generic element from a list requires constant time
- sorting requires $\Theta(k \log k)$ steps

Some heuristics for Min Storage

GREEDY($\mathcal{E}$)

$L \leftarrow \text{Sort}(\mathcal{E})$
$st \leftarrow 0$
$stmax \leftarrow 0$
while Length$(L) > 0$ **do**
    $i \leftarrow 1$
    while $st + L[i] < 0$ **do**
        $i \leftarrow i + 1$
    endwhile
    $st \leftarrow st + L[i]$
    $stmax \leftarrow \max\{st, stmax\}$
    remove $L[i]$ from L
endwhile
return $stmax$

MIN($\mathcal{E}$)

$L_n \leftarrow$ negative values of $\mathcal{E}$
$L_p \leftarrow \mathcal{E} \setminus L_n$
sort L_p and L_n in increasing order
$st \leftarrow 0$
$stmax \leftarrow 0$
while Length$(L_p) > 0$ **do**
    $st \leftarrow st + \min(L_p)$
    $stmax \leftarrow \max\{st, stmax\}$
    remove $\min(L_p)$ from L_p
    while Length$(L_n) > 0$ **and**
            $st + \max(L_n) \geq 0$ **do**
        $st \leftarrow st + \max(L_n)$
        remove $\max(L_n)$ from L_n
    endwhile
endwhile
return $stmax$

Some heuristics for Min Storage

BEST FIT($\mathcal{E}$)

$L_n \leftarrow$ negative values of $\mathcal{E}$
$L_p \leftarrow \mathcal{E} \setminus L_n$
sort L_p and L_n in increasing order
$est \leftarrow \max_{1 \le i \le k} |e_i|$
$st \leftarrow 0$
while Length(L_p) > 0 **do**
  if $st + \min(L_p) > est$ **then**
    $est \leftarrow st + \min(L_p)$
    $st \leftarrow st + \min(L_p)$
    remove $\min(L_p)$ from L_p
  else
    for $i \leftarrow$ Length(L_p) **downto** 1 **do**
      if $st + L_p[i] \le est$ **then**
        $st \leftarrow st + L_p[i]$
        remove $L_p[i]$ from L_p
      endif
    endfor
  endif
  for $i \leftarrow 1$ **to** Length(L_n) **do**
    if $st + L_n[i] \ge 0$ **then**
      $st \leftarrow st + L_n[i]$
      remove $L_n[i]$ from L_n
    endif
  endfor
endwhile
return est

First experiment

- 100 instances of 12 elements from the set $\{-10^6,\ldots,10^6\}$
- Goal: comparison between optimal solutions and coefficients of approximation

Algoritmo	Avg. appr. coeff.	Variance
Greedy	1.2052082	0.0615908
Min	1.3901304	0.0993216
Max	1.3804116	0.0979608
MaxMinMax	1.0863972	0.0143145
MinMaxMin	1.3901304	0.0993216
MaxMinMaxMin	1.1312751	0.0253605
MinMaxMinMax	1.1568342	0.0205104
Best Fit	1.0202729	0.0043468
MA4MS 2Loc	**1**	**0**

Second experiment

- 10^5 instances of 100 elements from the set $\{-10^6,\ldots,10^6\}$
- performance ratios have been computed using $\max_{1\le i\le k}|e_i|$ instead of *opt(I)*

ALGORITMO	AVG. APPR. COEFF.	VARIANCE
Greedy	1.3844832	0.0664088
Min	1.7585659	0.0720077
Max	1.7533215	0.0733613
MaxMinMax	1.1051660	0.0055823
MinMaxMin	1.7585659	0.0720077
MaxMinMaxMin	1.1356385	0.0057628
MinMaxMinMax	1.1368552	0.0058827
Best Fit	1.0072024	0.0017742
MA4MS 2Loc	**1.0202449**	**0.0006565**

Third experiment

- 100 instances of 100 elements from the set $\{-10^6,\ldots,10^6\}$
- Initial energy comprised between 0 and $\max_{1\le i\le k}|e_i|$ with steps of 100

ALGORITMO	AVG. APPR. COEFF.	VARIANCE
Greedy	1.3948121	0.0468795
Min	1.6441352	0.0416634
Max	1.6424797	0.0449882
MaxMinMax	1.4993568	0.0784421
MinMaxMin	1.6441352	0.0416634
MaxMinMaxMin	1.5032439	0.0757982
MinMaxMinMax	1.1470192	0.0042202
Best Fit	1.1619727	0.0219509
MA4MS 2Loc	**1.0185239**	**0.0005584**

Fourth experiment

- The same as the third experiment, but with initial energy comprised between 0 and $1.1 \cdot \max_{1 \le i \le k} |e_i|$ with steps of 100

ALGORITMO	AVG. APPR. COEFF.	VARIANCE
Greedy	1.3821281	0.0484121
Min	1.6259243	0.0468909
Max	1.6700879	0.0486042
MaxMinMax	1.5399753	0.0878942
MinMaxMin	1.6259243	0.0468909
MaxMinMaxMin	1.5435091	0.0852048
MinMaxMinMax	1.1446578	0.0042073
Best Fit	1.1489122	0.0216834
MA4MS 2Loc	**1.0179173**	**0.0005067**

Fifth experiment

- 10^5 instances of 100 elements from the set $\{-10^6,\ldots,10^6\}$
- Initial energy fixed to $\frac{1}{2}\cdot\max_{1\le i\le k}|e_i|$

ALGORITMO	AVG. APPR. COEFF.	VARIANCE
Greedy	1.3423574	0.0294341
Min	1.5627664	0.0075823
Max	1.5651455	0.0076282
MaxMinMax	1.4908638	0.0002487
MinMaxMin	1.5627664	0.0075823
MaxMinMaxMin	1.4908638	0.0002487
MinMaxMinMax	1.1511381	0.0041060
Best Fit	1.1833944	0.0227208
MA4MS 2Loc	**1.0206464**	**0.0006409**

Conclusions

- Min Storage seems to be easy to solve on (almost uniformly) randomly chosen instances
- MA4MS LS is among the best heuristics (the best for relaxed problem)
- Best Fit performs better when the initial energy is set to 0, but not when the initial energy is > 0
- MA4MS LS seems to be stable
- Comparison with analogous algorithms for further problems?

Last slide !