Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering
and Networks Laboratory

Bio-inspired Computation and Optimization

Project 1: Knapsack Problem – Part I
Discussion of Task1

Task 1 a)

a) Describe how you would represent a solution x for the knapsack problem and define the search space X. *(5 Points)*

Task 1 a), Solution

1.1 Solution representation

Given are n items and a capacity c, where each item i has a positive weight w_i and profit p_i. From each item, only one instance is available (*0/1 bounded problem*).

We define a solution as a vector $x \in X$, with

$$x = (\alpha_1, \alpha_2, \ldots, \alpha_n)$$

where $\alpha_i \in \{0, 1\}$ denotes the presence (1) or absence (0) of item i in the knapsack. Therefore $x \in \{0, 1\}^n$ and the search space is

$$X = \{0, 1\}^n$$

- Give x
- Give meaning of elements of x
- Give X

Task 1 a) (alternative solution)

S : set of items I
Cardinality of S : $Card(S) = n$
Each item I is a pair : $I_i = (w_i, p_i), i \in [\![1, n]\!]$

$$\begin{cases} w_i \in \mathbb{N} \text{ (weight)} \\ p_i \in \mathbb{N} \text{ (profit)} \end{cases}$$

One solution x is one subsets of S
$x \in \mathcal{P}(S)$
$x = \{I_i, I_j, I_k, ...\}; i, j, k, ... \in [\![1, n]\!]$
The search space is the powerset of S : $X = \mathcal{P}(S)$
The size of the search space is $Card(\mathcal{P}(S)) = 2^n$)

- Harder to implement (variable length vectors)

Task 1 a) common pitfalls

- not giving meaning of x:

We use the following solution vectors : $\vec{x} = (x_1, x_2, \ldots, x_n), x_i \in \{0,1\}$

- excluding infeasible solutions:

$x = (i_1, i_2, .., i_N)$ with $N < n$ items such that $\sum_n w_i \leq c$

- only considering optimal solutions:

Representation of a Solution: We are looking for the largest accumulated profit

Task 1 a) common pitfalls, cont.

- only using words:

  - A solution x can be represented as a vector of size n with each position representing an item switched on or off
  - The Search space is the sum of all possible solutions, there are $\{0,1\}^n = 2^n$ possibilities.

- talking about implementation:

```
Our search space X is a matrix with binary numbers; each column
represents a candidate to be evaluated.
```

- Allowing copies of the same item:

Task 1 b)

b) Give an expression for the objective function f. Solutions that exceed the given capacity bound shall receive an objective function value of zero. *(5 Points)*

Task 1 b), (alternative solutions)

$$f(x) = \begin{cases} \{\sum p_i \mid x_i = 1\} & \text{if } \{\sum w_i \mid x_i = 1\} \leq c \\ 0 & \text{otherwise} \end{cases}$$

$$f\,(x) = \begin{cases} xp^T & \text{if } xw^T \leq W \\ 0 & \text{if } xw^T > W \end{cases}$$

$$f(x) = \left(\sum_{i=1}^{n} p_i\right) \cdot min\left\{max\left\{c + 1 - \sum_{i=1}^{n} w_i, 0\right\}, 1\right\}$$

Task 1 b), common pitfalls

- +- 1 errors on the index variable, i.e., ranging from 0 to n:

$$f(x) = \begin{cases} \sum_{i=0}^{n} x_i \; p_i & if \sum_{i=0}^{n} x_i \; w_i \leq c \\ 0 & otherwise \end{cases}$$

- < rather than <= when checking the constraint:

$$f(x) = \begin{cases} \sum_{i=1}^{n} p_i x_i & if \sum_{i=1}^{n} w_i x_i < c \qquad x_i \in \{0,1\} \\ 0 & else \end{cases}$$

Task 1 b), common pitfalls, cont.

- forgetting x_i:

$$f(x) = \begin{cases} \sum_{i=1}^{n} p_i & \text{if } \sum_{i=1}^{n} w_i \leq c \\ 0 & \text{else} \end{cases}$$

Task 1 c)

c) Describe an algorithm to calculate the objective function value of a given solution x in pseudo code. *(5 Points)*

- **Algorithm** calculateObjectiveValue(x, p, w, n, cap)

```
    Input: The set of chosen items x, the number of items n,
           the vectors p (profit) and w (weight) of length n and
           the capacity of the knapsack cap.
    Output: The objective value

    sum_profit = 0
    sum_weight = 0

    for i = 1 to n do
          sum_profit = sum_profit + x[i]*p[i]
          sum_weight = sum_weight + x[i]*w[i]
          if sum_weight > cap then
                return 0
          end if
    end for
    return sum_profit
```

Task 1 c), Avoid very brief notations

```
weight = scalarproduct(x,w)
```

$$\text{calculate } f(\vec{x}) = \sum_{i=1}^{n} p_i x_i$$

TOTAL_WEIGHT equals sum of weights of selected items of x

Task 1 c), common pitfalls

- Forgetting x_i:

```
for i = 1 to n
  weight = weight + w_i
  profit = profit + p_i
```

- Calculating the fitness for all solutions:

```
For each x do
   if sum(x.*weight) > capacity
       then tot_profit = 0;
   else
       tot_profit = sum(x.*profit)
```

- Confusing profit and weight:

```
for (i = 1; i <= n; i++){
        sum += p_i * x_i;
}
if (sum > c){
        sum = 0;
}
```

Task 1 c), common pitfalls, cont.

- Wrong beginning or ending number of iteration:

$$\mathrm{FOR}\ \ \mathrm{i}\ =\ \ 0:\mathrm{n}\ \ \mathrm{DO}$$

- Randomly generating solutions:

```
05: randomly choose a solution xt∈X
06: for i in range(n)
07:     pt+=p[i]
08:     wt+=w[i]
```

Task 1 d)

d) Choose a programming language. Write a program that is able to read all problem parameters (number of items, profit and weight values, capacity) from text files[1]. Further, the program shall generate 10000 random solutions and calculate the objective function value of each of them. The program output consists of the best solution encountered and the corresponding objective function value. Run the program on the different problem instances given on the lecture website. *(10 Points)*

Task 1 d) Part 1

```
NumberIterations = 100000;

for  m = 1:4

    a = strcat('instance',num2str(m),'.txt');
    A = textread(a,'%u');

    numberItem = A(1);
    capacity = A(end);
    profit - A(2:1+numberItem);
    weight = A(2+numberItem:end-1);


    solution = 0;
    x = [];
```

Thanks to Raphael Suard, Michael Benz und Olle Sundström

Task 1 d) Part 2

```
    for i = 1:NumberIterations

        x1 = [];
        for j = 1:numberItem
            x1(j) = randn()>0;
        end

        solution1 = sum(x1.*profit');
        if sum(x1.*weight')>capacity
            solution1 = 0;
        end

        if solution1 > solution
            solution = solution1;
            x = x1;
        end

    end
    solution
    x

end
```

Thanks to Raphael Suard, Michael Benz und Olle Sundström

Task 1 d) (short solution)

```
data = load('project1_task1d_instance1.txt');

% data = load('project1_task2b.txt');
n=data(1);
p=data(2:1+n);
w=data(2+n:end-1);
c=data(end);

tic
%generate 10000 random solutions
solutions=randerr(10000,n,1:n);

obj    = solutions*p;
constr = solutions*w<=c;
obj    = obj.*constr;

[a bestsolution]=max(obj);

disp(['Best solution: ' num2str(solutions(bestsolution,:)   ) ] );
disp(['Zielwert der Best solution: ' num2str( a ) ] );
toc
```

Task 1 e)

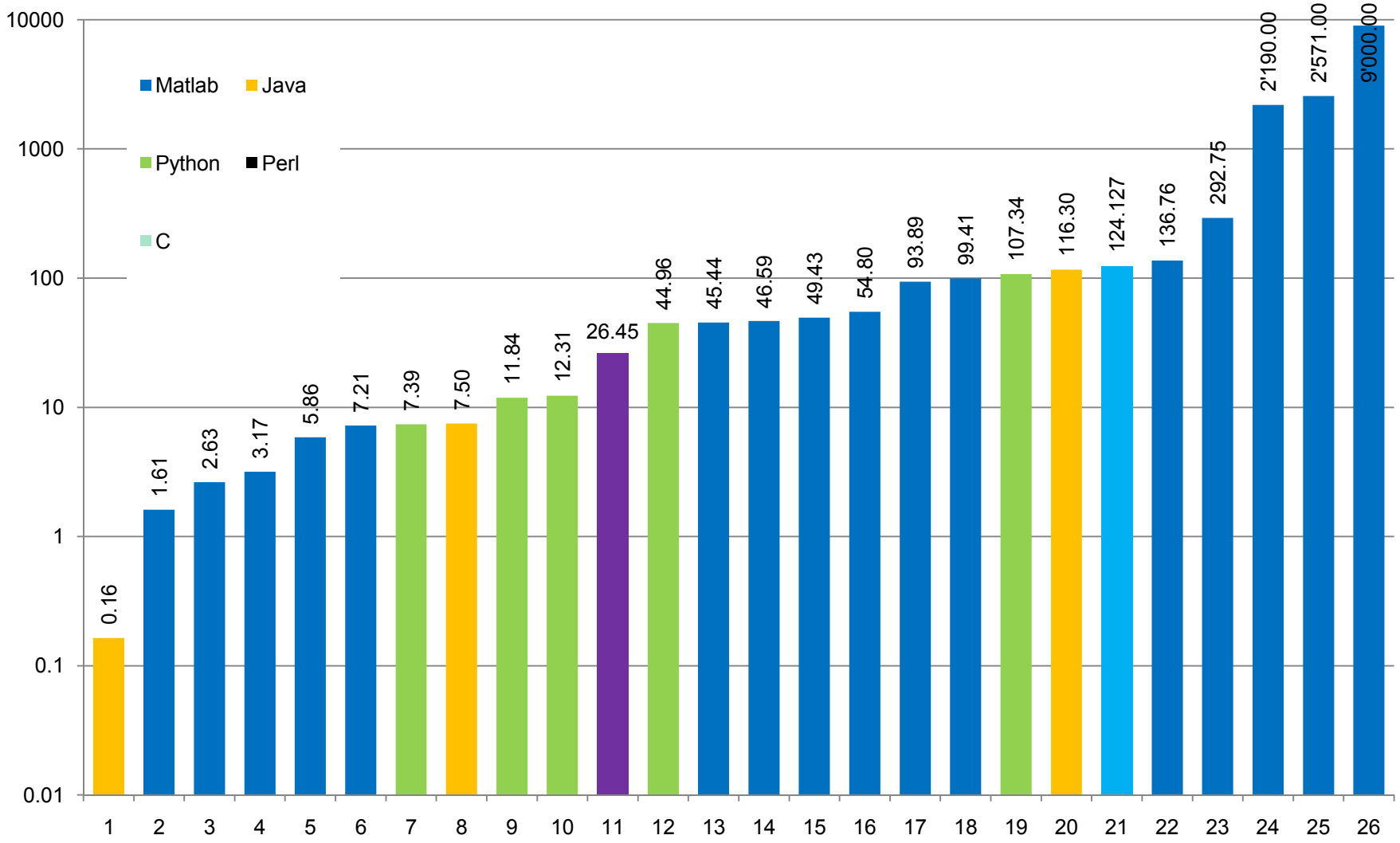
- Exhaustive search, check all 2^n solutions.
- To get the ith solution, you can use the binary representation of i, e.g, the solution 13 would be 00001101 (8 Items). (Make sure you cover the solution 00.....0.

```
for i = 1:2^numberItem

    x1 = [];
    bin = i;
    for j = 1:numberItem
        x1(j) = rem(bin,2);
        bin = floor(bin/2);
    end
```

Task 1 e), runtimes

- Runtimes in seconds on the instance with 20 items



Task 1 e), Fast way to generate all combinations

```
for ( int k=0; k<(1<<n); k++ ) {
      for ( int i=0; i<n; i++ ) {
            x[i] = (k>>i) & 1;
      }
      double p = f( x, weight, profit, capacity );
      if ( p > bestProfit ) {
            bestProfit = p;
            bestX = x.clone();
      }
}
```