ISBBC17

Valencia

Networks of Evolutionary Processors for Optimization

Mathematical Models & Biocomputing Algorithms Group

Outline

1. A brief reminder of Networks of Bioinspired Processors – NBP
2. Optimization problems: NEPO
3. Example: $0/1$ Knapsack Problem

A brief reminder of Networks of Bioinspired Processors - NBP

1. A brief reminder of NBP

- Highly parallel and distributed computing models. [Mitrana et al., 2001-2007-2014]
- Symbolic processing
  - Point mutations DNA sequences.
  - Evolutionary rules.
  - Data is represented by words (strings).
  - Selection process:
    - Context conditions.
    - Polarization.
- Can be used to solve NP-complete problems.

1. A brief reminder of NBP

Underlying architecture

- NBP family share the same architecture:
  - Simple operations, normally rewriting rules
  - Graph like structure:

    Processing within nodes, communicating through edges.
  - Extensible and scalable.
- Similar to Bulk Synchronous Parallel model (BSP).

  [Valiant, 1990]

  Parallel computing architecture implemented by Google Pregel, Giraph, Dato, etc.

NBP Family

Evolutionary

- Insertion
- Deletion
- Substitution

Context

Polarized

- Insertion
- Deletion
- Substitution

Polarization

Splicing

Splicing

Context

Genetic

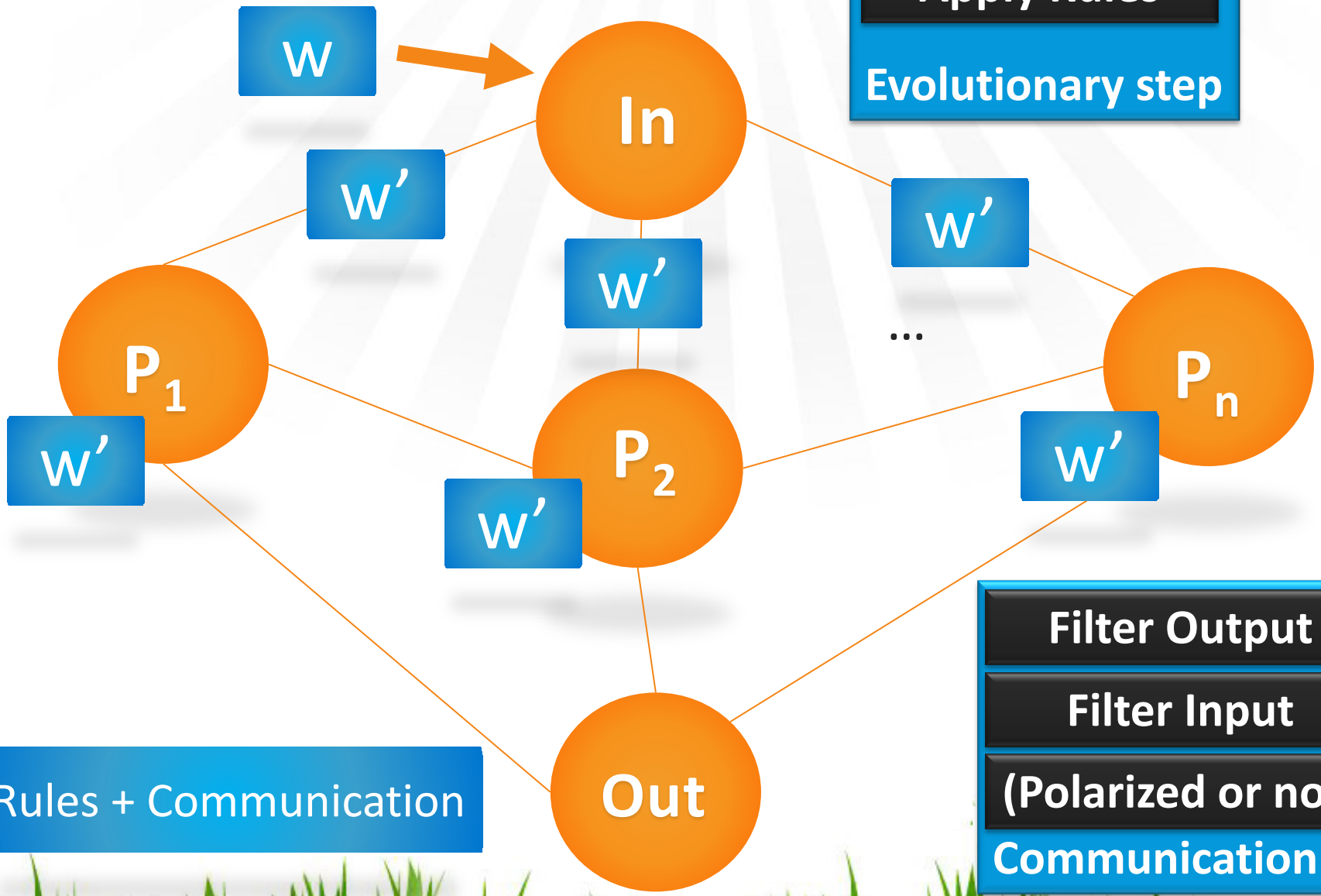
Splicing Crossover

Context

Numerical evaluation over words

- Used from a qualitative perspective
- Don't distinguish the overall charge: negative, positive o neutral only

NEP (evolutionary) dynamics



Apply Rules

Evolutionary step

Rules + Communication

Filter Output

Filter Input

(Polarized or not)

Communication step

1. A brief reminder of NBP

Networks of evolutionary processors - NEP

- Simple NEP [1]
  - Apply evolutionary rules.
  - Use context conditions to communicate (filtering).
- Polarized NEP [2]
  - Apply evolutionary rules.
  - Use valuation of words to communicate (the sign only).

NEP have qualitative behavior: processors do not preserve any information over data while the network is working.

[1] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, Acta Inf. 39 (6–7) (2003) 517–529.
[2] P. Alarcón, F. Arroyo, V. Mitrana, Networks of polarized evolutionary processors, Inf. Sci. 265 (2014) 189–197.

1. A brief reminder of NBP

Drawbacks for NEP models

NEP models …

- Do not provide any information about data structure.

  Any permutation of a given word can pass the same filters.

- Lack of details about the interaction between the symbols which are present in the current word.

  For instance, suppose that a word represents a substance. It is not possible to determine if some chemical reaction will be take place by considering the symbols in the word only.

- Can’t track / store relevant data while computing.

Optimization problems: NEPO

2. Optimization problems: NEPO

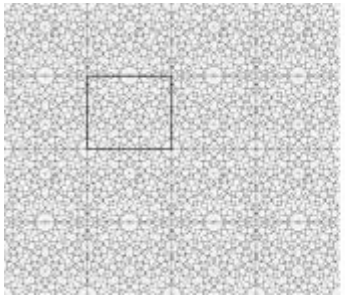
Neccesity of addressing optimization problems

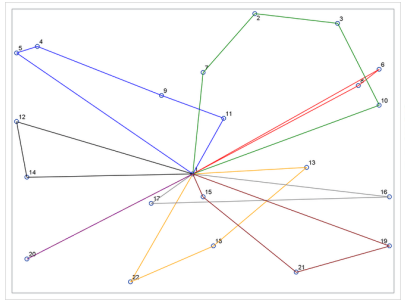
NP-complete problems

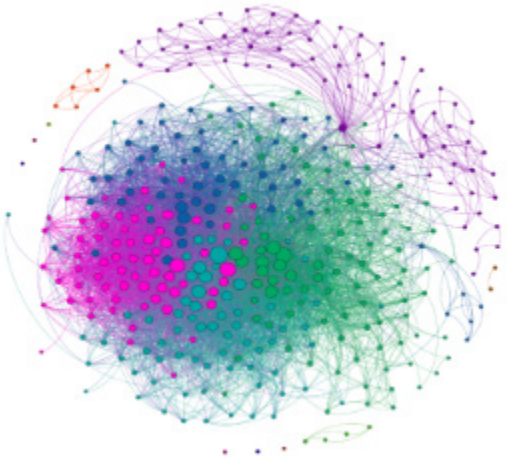
- Knapsack Problems.

Machine learning problems

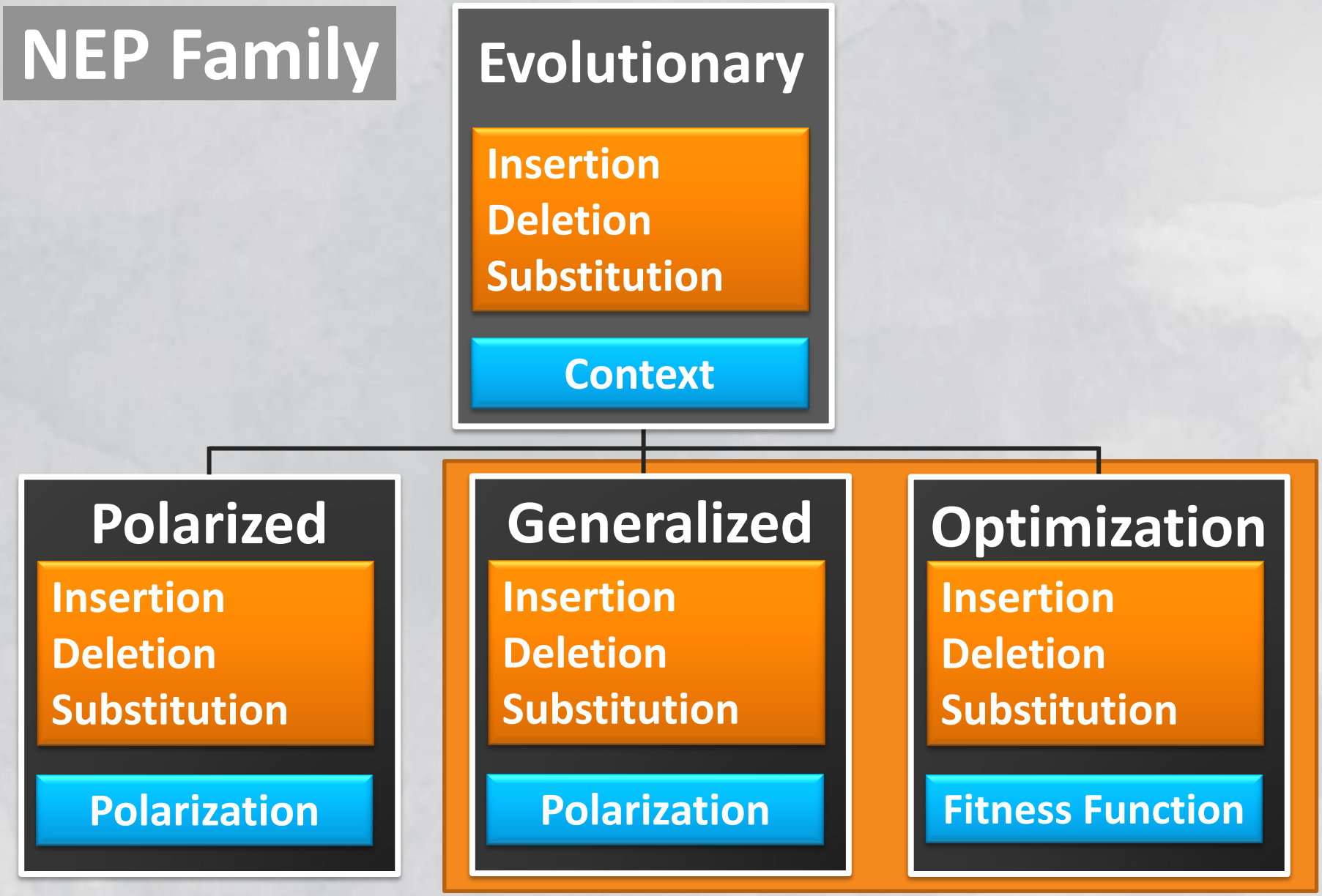
- Supervised and unsupervised learning.
- Pattern recognition.
- Generalized assignment.
- Others.







NEP Family


Evolutionary
Insertion
Deletion
Substitution
Context
Polarized
Insertion
Deletion
Substitution
Polarization
Generalized
Insertion
Deletion
Substitution
Polarization
Optimization
Insertion
Deletion
Substitution
Fitness Function

2. Optimization problems: NEPO

What's new?

- The design of a solution to an optimization problem can be built up as an architecture of different NEP, working each one as a solver of one or several parts of the problem in question.
- An agent is a list of parameters which indicate how to select, by using some criterion, the best elements of a given dataset. Precisely, an agent can be viewed as a generic vector $A_i = \left(a_i^1, a_i^2, \ldots, a_i^k\right)$, where each a_i^s is updated when computing the fitness (objective) function.
- A set of N agents is represented by a matrix $(A_i)_{i=1}^N$.

2. Optimization problems: NEPO

What's new?

- NEPO architecture is similar to that of NEP, but now every processor interacts with its own matrix of agents.
- The matrix of agents $A = (A_i)_{i=1}^{N}$ is updated by using a vectorial function $F = (F_1, F_2, \dots, F_N)$. Here F_i acts over the words currently placed at a given node, updating the single agent A_i if necessary.
- Communication steps in the network are now governed by the matrices of agents (one for each processor).

2. Optimization problems: NEPO

General idea to solve optimization problems

Join two networks sequentially [3]

1. Consider a first NEPO which is responsible of approximating, even generating, all the possible candidates that solve some instance of a given problem.

2. Construct a second NEPO which is able to find an optimal solution between the candidates.

The data met in the output node of the first network will be transferred to the input node of the second one.

[3] J.R. Sánchez Couso, S. Gómez Canaval, D. Batard Lorenzo, How to search optimal solutions in big spaces with networks of bio-inspired processors, in:Advances in Computational Intelligence, LNCS,vol. 9094, Springer International Publishing, 2015, pp. 29–39.

2. Optimization problems: NEPO

Formal definitions - EPO

Let $\mathcal{A}$ be a given fixed class of matrices. An **evolutionary processor for optimization** (EPO) over an alphabet V is a triple (M, α, A), where

- M is a set of either **substitution**, or **insertion**, or **deletion** rules over V (i.e. $a \to b$, $\varepsilon \to a$, $a \to \varepsilon$, with $a, b \in V$).
- $\alpha \in \{l, *, r\}$ is the **action mode** of the rules: left side, any place, right side.
- $A \in \mathcal{A}$ is a matrix representing a set of agents.

2. Optimization problems: NEPO

Formal definitions - NEPO

A **network of evolutionary processors for optimization** (NEPO) over $\mathcal{A}$ is a 7-tuple $\Gamma_{\mathcal{A}} = (V, U, G, R, F, \underline{In}, \underline{Out})$, where

- V and U are the input and the network alphabets, $V \subseteq U$.
- $G = (X_G, E_G)$ is the underlying graph of the network.
- $R: X_G \longrightarrow EPO_U$ associates each node $x \in X_G$ with the EPO $R(x) = (M_x, \alpha_x, A^x)$ over U, with $A^x \in \mathcal{A}$.
- $F: 2^{U^*} \longrightarrow \mathcal{A}$ is the fitness function for optimization.
- $\underline{In}, \underline{Out} \in X_G$ are the input and output nodes of $\Gamma_{\mathcal{A}}$.

2. Optimization problems: NEPO

NEPO dynamics

A **configuration** of a NEPO Γ is a mapping

$$C: X_G \longrightarrow 2^{U^*}$$

Initial configuration of Γ,

$$C_0(x) = \{w_{i_1}^x, \ldots, w_{i_s}^x\}, \quad \forall x \in X_G - \{\underline{Out}\}$$

- Eventually it could be $C_0(x) = \emptyset$.
- The input node $\underline{In}$ of Γ might lack of relevance.

2. Optimization problems: NEPO

NEPO dynamics

Configuration changes

- **Evolutionary step**: Each component $C(x)$ is changed in accordance with the set of evolutionary rules M_x in $R(x)$, and the way of applying these rules α_x.
- **Communication step**: Each node processor $x \in X_G$ sends a copy of its **eligible words** to all nodes connected to x. The **eligible words** are determined from the matrix A^x which is dynamically updated, if necessary, in accordance with the fitness function F. The processor x receives all the words sent by any node processor connected with x, provided these words are eligible in their respective nodes.

2. Optimization problems: NEPO

NEPO dynamics

Evolutionary step

$C \Rightarrow C'$ if and only if $C'(x) = M_x\big(C(x)\big) \quad \forall x \in X_G$

Communication step

$C \vdash C'$ if and only if

$$C'(x) = \big(C(x) - \pi(A^x)\big) \cup_{\{x,y\} \in E_G} \{z \in C(y) : z \in \pi(A^y)\}$$

- $\pi(A^y)$, set of eligible words of node y.
- $A^y = F(C(y))$, best matrix of agents coming from words at node y.

2. Optimization problems: NEPO

NEPO dynamics

The **computation** of a NEPO Γ on a node x is

$$C_0(x), C_1(x), C_2(x), \ldots$$

by alternating evolutionary & communication steps,

$$C_{2i}(x) \Rightarrow C_{2i+1}(x) \;\&\; C_{2i+1}(x) \vdash C_{2i+2}(x) \quad \text{for } i \geq 0.$$

The computation **halts** if no further step is possible.

2. Optimization problems: NEPO

NEPO algorithm for single solution

1: Generate the set of words $W = \{w_i\}_{i \in I}$ encoding an instance of a problem
2: A NEPO Γ with graph X_G and fitness function F
3: $L \leftarrow$ List of nodes of $X_G - \{\underline{Out}\}$
4: **for all** $i \in I$ **do**
5: Select randomly $x_j \in L$
6: $x_j \leftarrow w_i$
7: **end for**
8: Generate matrix $(A_i^{x_j})_{i=1}^N$ of agents
Require: In parallel way: for each $x \in X_G$
9: **while** Γ does not halt **do**
10: Generate an evolutionary step
11: Update matrix $(A_i^x)_{i=1}^N$ if necessary
12: Generate a communication step
13: **end while**
14: **return** Set of words met in node $\underline{Out}$

Example: 0/1 Knapsack problem

3. Example: 0/1 Knapsack problem

Problem statement

"Given a knapsack of capacity K and a set $\mathcal{E}$ of n elements, where each element has a weight and a profit, choose a subset of m elements of $\mathcal{E}$ such that its corresponding profit sum is maximized without having the weight sum to exceed the capacity K"

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

Let $K > 0$ and consider the parameters

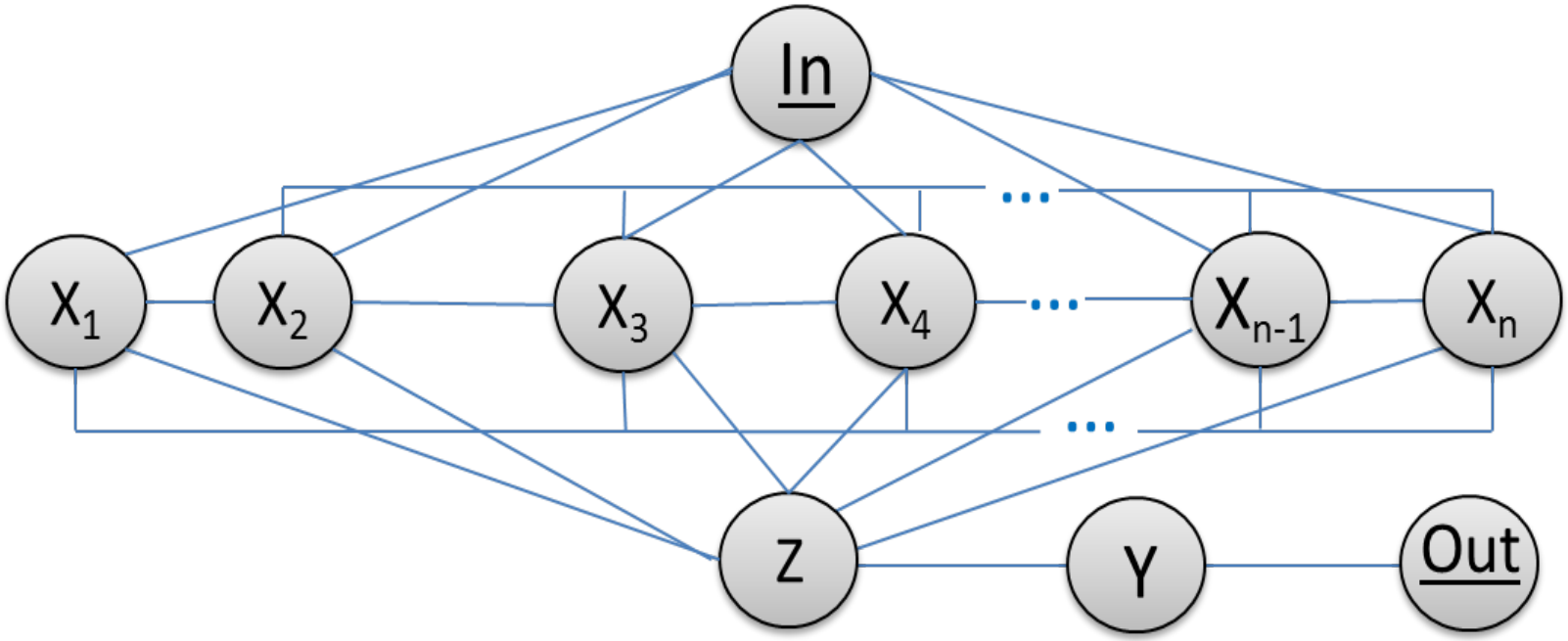
$\mathcal{E} = \{e_1, e_2, \dots, e_n\}$ with $\text{weight}(e_i) = w_i$ and $\text{profit}(e_i) = p_i$.

Construct a NEPO $\Gamma = (V, U, G, R, F, \underline{In}, \underline{Out})$ as follows:

- $V = \{e_1, e_2, \dots, e_n\}$
- $U = V \cup \{e_1', e_2', \dots, e_n', e_1'', e_2'', \dots, e_n''\}$
- $X_G = \{\underline{In}, Z, Y, \underline{Out}, X_1, \dots, X_n\}$

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver



Underlying graph X_G for the $0/1$ Knapsack problem

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

Valuation function (base for the fitness function)

$$\phi : U^* \longrightarrow \mathbb{Z},$$

$$\phi(e_i) = 0\,, \quad \phi(e_i') = w_i\,, \quad \phi(e_i'') = p_i\,, \quad 1 \le i \le n,$$

$$\phi(s) = \sum_j \phi(s_j)\,, \qquad s = s_1 s_2 \ldots s_k\,, \quad s_j \in U.$$

ϕ is invariant under anagrams

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

Matrix class $\mathcal{A}$ degenerates into a 3 component vector,

$$A^x = (a^{x,1}, a^{x,2}, a^{x,3}), \quad \forall x \in X_G.$$

– $a^{x,1} \in \{0, 1\}$ (in the multiset sense).

– $a^{x,2}, a^{x,3} \in \mathbb{Z}$.

Initially, $A^x = (0, 0, K), \quad \forall x \in X_G.$

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

Fitness function

$$F: 2^{U^*} \longrightarrow \mathcal{A}$$

For clarity, F is defined as a piecewise function, depending on what node $x \in X_G$ it acts:

$$F(s) = \begin{cases} F_1(s) & \text{if } x \in X_G - \{Z, Y, \underline{Out}\}, \\ F_2(s) & \text{otherwise.} \end{cases}$$

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

1. For $s \in C(x)$, $x \notin \{Z, Y, \underline{Out}\}$,

$$F_1(s) = \begin{cases} 1 & \text{if } \phi(s) \leq K, \\ 0 & \text{otherwise,} \end{cases}$$

$$A^x = (F_1(s), 0, K).$$

2. For $s \in C(Z)$,

$$F_2(s) = \phi(s),$$

$$A^Z = (1, \max(F_2(s), T), K),$$

$T \geq 0$ is the last value found in $a^{x,2}$.

3. For $s \in C(Y)$,

$$F_2(s) = \begin{cases} 1 & \text{if } \phi(s) = T, \\ 0 & \text{otherwise,} \end{cases}$$

$$A^Y = (F_2(s), T, K).$$

Note that A^x is updated **after** applying an evolutionary rule to the word s.

3. Example: 0/1 Knapsack problem

NEPO as optimization problem solver

The mapping $R(x)$ for the NEPO Γ

Node x	Rules in M_x	Action mode α_x	A^x
$\underline{In}$	$\emptyset$	$\emptyset$	$(1, 0, K)$
X_i	$e_i \to e_i'$	any place	$(F_1(\cdot), 0, K)$
Z	$e_i' \to e_i''$, $\forall i, 1 \le i \le n$	any place	$(1, \max(F_2(\cdot), T), K)$
Y	$e_i \to \varepsilon$, $\forall i, 1 \le i \le n$	left	$(F_2(\cdot), T, K)$
$\underline{Out}$	$\emptyset$	$\emptyset$	$(-, -, -)$

3. Example: 0/1 Knapsack problem

Considerations about NEPO Γ

- Notice that there is an abuse of notation on $F_i(\cdot)$ since it is defined on words instead of a set of words, but this is for not complicating the writing only.
- Γ hasn't any type of input filters on the nodes, therefore any copy of a word sent out by a processor is never lost.
- Once a processor has transmitted all of the copies of a given word, this word is removed from the processor in question.
- As a consequence, the management of the subset $\pi(A^x)$ of eligible words at node x is rather simple in this network.

3. Example: 0/1 Knapsack problem

Technical Results

Lemma 1. *The NEPO Γ defined above computes the optimal solution for the 0/1 Knapsack problem.*

Proof. At the beginning of the computation, Γ receives within the node $\underline{In}$ a word $s_0 = e_1, e_2, \ldots, e_n$ encoding the elements to be evaluated. Thus, the initial configuration of Γ on s_0 is defined by $C_0(\underline{In}) = \{s_0\}$ and $C_0(x) = \emptyset$ for all $x \in X_G - \{\underline{In}\}$. At this stage, we have $A^x = (0, 0, K)$ for all $x \in X_G$, where $K > 0$ represents the capacity of the knapsack. Now, a copy of s_0 is sent out to every node X_i since $F_1(s_0) = 1$. Note that each node X_i is specialized to apply the single rule $e_i \rightarrow e'_i$ only. This rule, if applicable, produces a new word $s = ue'_iv$ which is communicated by X_i, provided s satisfies the restriction of capacity, namely $\phi(s) \leq K$. By construction of the graph G, after replacing all of the possible occurrences of e_i, the node Z has necessarily a set of words encoding each and every admissible content of the knapsack. The transition of Z to Y is immediate: after at most n evolutionary steps, all of the symbols e'_i in Z are replaced by the respective e''_i and then, each of these words is communicated to Y after updating repeatedly the vector A^Z. Observe that $a^{Z,1} = 1$ identically for any word and, ultimately, $T = a^{Z,2} = \max\{\phi(s) : s \in C(Z)\}$. Finally, again by construction, the node Y sends out to $\underline{Out}$ those words of the form $e''_{i_1}, e''_{i_2}, \ldots, e''_{i_k}$ whose profit equals the maximum T. □

3. Example: 0/1 Knapsack problem

Technical Results

Theorem 1. *The 0/1 Knapsack problem for a set $\mathcal{E}$ of cardinality $n \in \mathbb{N}$ can be solved by a NEPO Γ in time $O(n)$.*

Proof. In the same conditions as before, the evolutionary steps in Γ are:

- There are n evolutionary steps in the subgraph with nodes $\{X_i\}_{i=1}^{n}$.
- There are at most n evolutionary steps in node Z.
- There are at most n evolutionary steps in node Y.

Hence the total number of evolutionary steps does not exceed $3n$. Note that, by construction, Γ finds not only one, but all of the optimal solutions to a given instance of the problem. □

3. Example: 0/1 Knapsack problem

Final remarks

- NEPO models require massively scalable platforms to implement SW that works reasonably.
- For this motive, it is used the **diamond subgraph** with nodes X_i in the underlying graph. This distribution seems to be more adequate to reallocate units of parellel computing in real machines.
- However, NEPO models may have a significant drawback that is the **intrinsic complexity** for communication steps. Normally, these steps may be more laborious to establish than other types of NEP. Clearly, this is not the case for evolutionary steps.

3. Example: 0/1 Knapsack problem

Preliminary experimental results

NPEPE [4] : High scalable engine designed to run polarized NEP models in distributed and parallel computational platforms.

NPEPE engine follows the BSP (Bulk Synchronous Parallel) model implemented by Giraph (open source version of Google Pregel).

[4] S. Gómez-Canaval, B. Ordozgoiti, A. Mozo, NPEPE: massive natural computing engine for optimally solving NP-complete problems in big data scenarios, in:New Trends in Databases and Information Systems, Communications in Computer and Information Science, vol. 539, Springer, 2015, pp. 207–217.

3. Example: 0/1 Knapsack problem

Preliminary experimental results

NPEPE : How it works

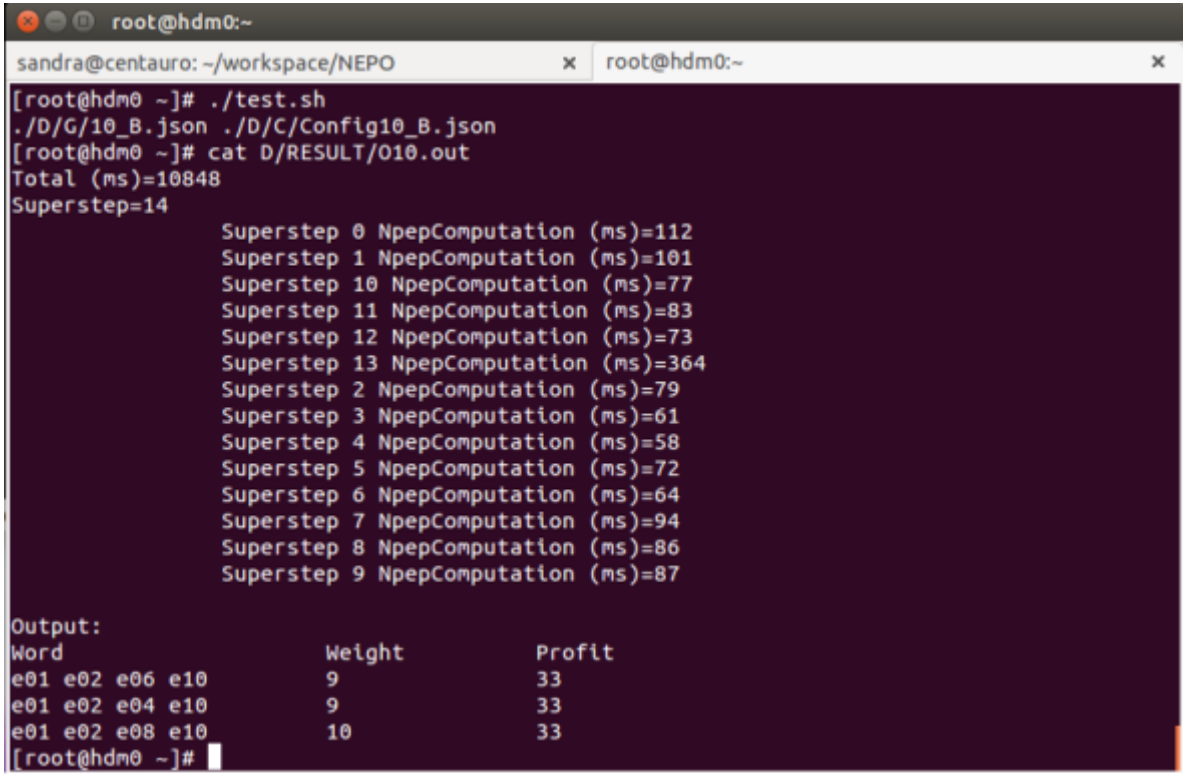
- Every computational component (similar to a processor) is associated with a node of some graph.
- A NPEPE computation is a sequence of iterations called **supersteps**.
- Each node has a user-defined computing function that establishes its behavior. All these functions are invoked simultaneously during a superstep.
- Any node can send messages and exchange values with the others.
- A global procedure checks if all the computing functions have ended.

3. Example: 0/1 Knapsack problem

Preliminary experimental results

Testing Machine Setup

- Intel Core i7-3770 (3.4 GHz)
- 8 GB RAM, 1 TB HDD
- OS Ubuntu 14.04
- Apache Giraph 1.1.0
- Hadoop Framework 2.6
- Java 1.8



Small 0/1 Knapsack test

Objects $\mathcal{E} = 10$

Capacity $K = 10$

Weights $W = (1, 1, 5, 6, 7, 6, 4, 7, 8, 1)$, Profits $P = (10, 10, 2, 3, 2, 3, 2, 3, 10, 10)$

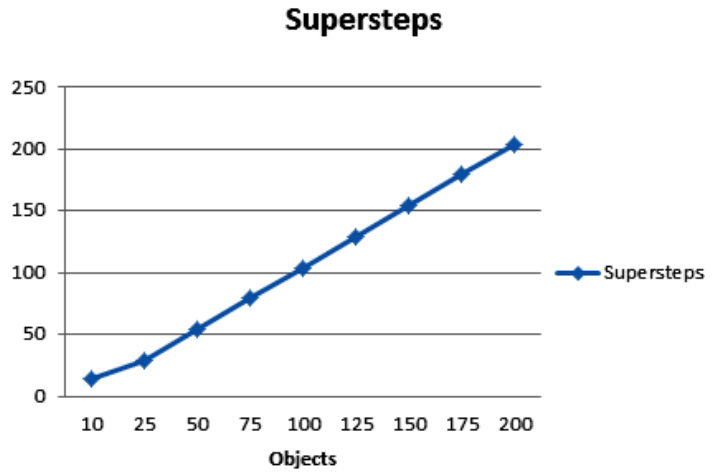
3. Example: 0/1 Knapsack problem

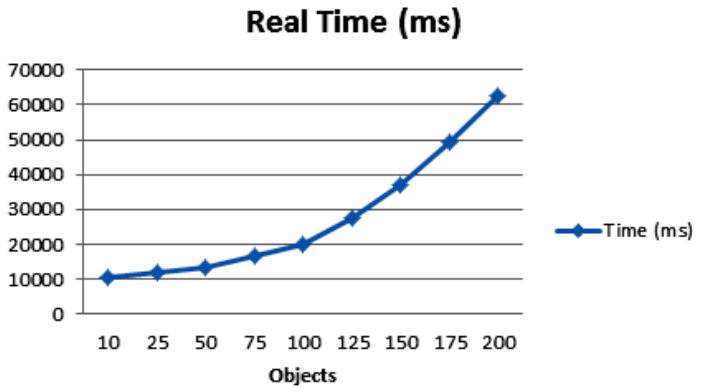
Preliminary experimental results

Some instances

- Objects: from 10 to 200 (+25)
- Capacity: always 50
- Random weights and profits
- Weights between 1 and 50

✓ Linear behavior in supersteps.
✓ Real time apparently polynomial.
✓ Simulated only, **NOT** run within a genuine parallel environment.





Thanks for your attention !