# Heuristic and Metaheuristic Optimization Techniques with Application to Power Systems

MIHAI GAVRILAS

"Gheorghe Asachi" Technical University of Iasi, Romania

## Main topics

- Optimization and (meta)heuristics
- Heuristic optimization
- Metaheuristics and metaheuristic methods
- Applications of (meta)heuristic methods in power systems
- Conclusions

## Optimization and (meta)heuristics

## Optimization

… is a branch of mathematics and computational science that studies methods and techniques specially designed for finding the "best" solution of a given "optimization" problem.

Such problems aim to minimize or maximize one or more objective functions based on one or more dependent variables, which can take integer or real values, and subject to a set of equality or inequality constraints.



## Traditional optimization methods

- Linear Programming
- Integer Programming
- Quadratic Programming
- Nonlinear Programming
- Stochastic Programming
- Dynamic Programming
- Combinatorial Optimization



## Difficulties faced by traditional optimization methods

- Passing over local optimal solutions
- The risk of divergence
- Handling constraints
- Numerical difficulties related to computing first or second order derivatives

## (Meta)Heuristic methods

- Heuristic and metaheuristic techniques were proposed in the early 70's.
- Unlike exact methods, (meta)heuristic methods have a simple and compact theoretical support, being often based on criteria of empirical nature.
- These issues are responsible for the absence of any guarantee for successfully identifying the optimal solution.

## Heuristic optimization



## What is a heuristic ?

- A heuristic is an alternative optimization methods able to determine not a perfectly accurate solution, but a set of good quality approximations to exact solution.
- Heuristics, were initially based essentially on experts' knowledge and experience and aimed to explore the search space in a particularly convenient way.

## Main characteristics

- A heuristic is designed to provide better computational performance as compared to conventional optimization techniques, at the expense of lower accuracy.
- The 'rules of thumb" underlying a heuristic are often very specific to the problem under consideration.
- Heuristics use domain-specific representations.



## Types of heuristics

- Uninformed or blind search strategies are applied with no information about the search space, other than the ability to distinguish between an intermediate-state and a goal-state.
- Informed search strategies use problem-specific knowledge, such as an evaluation function that assesses either the quality of each state in the search space, or the cost of moving from the current state to a goal-state.



## Uninformed search strategies

(basically non-heuristic)

- Depth First Search
- Breadth First Search
- Uniform Cost Search

## Informed search strategies

### Best First Search

Among all possible states at one level, the algorithm chooses to expand the most "promising" one in terms of a specified rule.



## Informed search strategies

### Beam Search

BeS is defined based on BrFS, which is used to build the search tree. At each level, all new states are generated and the heuristic function is computed for each state that is inserted in a list ordered by heuristic function values. The list is of limited length - "beam width". This limits the memory requirements, but the compromise risks to pruning out the path to the goal-state.



## Informed search strategies

### A* search algorithm

The A* search algorithm uses a BeFS strategy, and a heuristic function that combines two metrics: the cost from the origin to the current state (or the cost-so-far) and an estimation of the cost from the current state to a goal-state (or the cost-to-goal).

# Metaheuristics and metaheuristic methods



## What are metaheuristics ?

- The term metaheuristic was proposed by Glover at mid-80s as a family of searching algorithms able to define a high level heuristic used to guide other heuristics for a better evolution in the search space.
- The most attractive feature of a metaheuristic is that its application requires no special knowledge on the optimization problem to be solved (see the concept of general problem solving model).



## Types of metaheuristics

- Simulated annealing
- Tabu search
- Evolutionary computation techniques
- Artificial immune systems
- Memetic algorithms
- Particle swarm optimization
- Ant colony algorithm
- Differential evolution
- Harmony search
- Honey-bee colony optimization

etcetera

# Simulated Annealing

Studies on Simulated Annealing (SA) were developed in the 1980s based on the Metropolis algorithm [17], which was inspired by statistical thermodynamics, where the relationship between the probabilities of two states A and B, with energies $E_A$ and $E_B$, at a common tempera-ture $T$, suggests that states with higher energies are less probable in thermodynamic systems. Thus, if the system is in state A, with energy $E_A$, another state B of lower energy ($E_B < E_A$) is always possible. Conversely, a state B of higher energy ($E_B > E_A$) will not be excluded, but it will be considered with probability exp(- ($E_B$ – $E_A$)/$T$).

# Simulated Annealing

Application of Metropolis algorithm in search problems is known as SA and is based on the possibility of moving in the search space towards states with poorer values of the fitness function. Starting from a temperature $T$ and an initial approximation $\boldsymbol{X_i}$, with a fitness function Fitness($\boldsymbol{X_i}$), a perturbation is applied to $\boldsymbol{X_i}$ to generate a new approximation $\boldsymbol{X_{i+1}}$, with Fitness($\boldsymbol{X_{i+1}}$). If $\boldsymbol{X_{i+1}}$ is a better solution then $\boldsymbol{X_i}$, i.e. Fitness($\boldsymbol{X_{i+1}}$) > Fitness($\boldsymbol{X_i}$), the new approximation will replace the old one. Otherwise, when Fitness($\boldsymbol{X_{i+1}}$) < Fitness($\boldsymbol{X_i}$), the new approximation will be considered with probability $\boldsymbol{p_i}$=exp(-[Fitness($\boldsymbol{X_i}$)–Fitness($\boldsymbol{X_{i+1}}$)] / $\boldsymbol{T}$).

The above steps are repeated for a given number of times for a constant value of temperature, then temperature is updated by decreasing its value, and the iterative process continues until a stopping criterion is met.



# Simulated Annealing

- Data: initial approximation $\mathbf{X_0}$, initial temperature **T**, number of iteration for a given temperature **nT**.
- Optimal solution: $\mathbf{X_{best}} \leftarrow \mathbf{X_0}$.
- WHILE {stopping criterion not met}
  - **n** = 0; **i** = 0;
  - WHILE (**n** < **nT**) DO
    - choose a new approximation **Y**
    - accept or reject the new approximation based on the Metropolis rule: $\mathbf{X_{i+1}} = \mathrm{G}(\mathbf{X_i},\mathbf{Y},\mathbf{T})$
    - update optimal solution: if $\mathbf{X_{i+1}}$ is better than $\mathbf{X_{best}}$, then $\mathbf{X_{best}} \leftarrow \mathbf{X_{i+1}}$
    - next **n**-iteration (**n** ← **n**+1)
  - update temperature **T**
  - next **T**-iteration (**i** ← **i**+1)

# Tabu Search

Tabu Search (TS) was introduced by [18], as a search strategy that avoids returning to solutions already visited by maintaining a Tabu list, which stores successive approximations. Since the Tabu list is finite in length, at some point, after a number of steps, some solutions can be revisited. Adding a new solution to a complete Tabu list is done by removing the oldest one from the list, based o a FIFO principle (First In – First Out).

New approximations can be generated in different ways. The pseudocode presented in Table 2 uses the following procedure: at each step a given number of new approximations are generated in the neighborhood of the current solution **X**, but considering as feasible only the ones which are not in the Tabu list. Amongst the new approximations the best one is chosen to replace the current solution, being also introduced in the Tabu list.



# Tabu Search

- Data: length of the Tabu list $\mathbf{L_T}$, number of intermediate solutions **N**.
- Initialization: approximation **X**, Tabu list **TABU**={**X**}.
- Optimal solution: $\mathbf{X_{best}} \leftarrow \mathbf{X}$.
- WHILE {stopping criterion not met}
  - prepare the Tabu list: if Length(**TABU**) = $\mathbf{L_T}$, then delete the oldest item from the list.
  - generate **N** new aproximations in the neighborhood of **X** and select the best candidate-solution **Y** which is not **TABU**.
  - update current approximation $\mathbf{X} \leftarrow \mathbf{Y}$ and add it in the Tabu list: Add(**TABU**,**X**).
  - update optimal solution : if **X** is better than $\mathbf{X_{best}}$, then $\mathbf{X_{best}} \leftarrow \mathbf{X}$

# Evolution Strategy

Evolution Strategy (ES) was first proposed in [13] as a branch of evolutionary computation. It was further developed after 1970's. An ES may be described by two main parameters: number of parents in a generation $\boldsymbol{\mu}$, and number of offsprings created in a generation $\boldsymbol{\lambda}$. A common notation is ES($\boldsymbol{\mu}, \boldsymbol{\lambda}$). The main genetic operator that controls the evolution from one generation to another is mutation.

In a general ES($\boldsymbol{\mu}, \boldsymbol{\lambda}$) model (where $\boldsymbol{\lambda} = \mathbf{k} \cdot \boldsymbol{\mu}$), each generation starts with a population of $\boldsymbol{\lambda}$ individuals. The fitness of each individual is computed to rank them in descending order of their fitness. Amongst the current population, only the first $\boldsymbol{\mu}$ fittest individuals are selected to create the parent population (this selection phase is sometimes called *truncation*). Next, each of the $\boldsymbol{\mu}$ parents will create by repeated mutation $\mathbf{k} = \boldsymbol{\lambda} / \boldsymbol{\mu}$ offsprings. Eventually, the new, mutated population will replace the old one and the algorithm reiterates. The pseudocode for the ES($\boldsymbol{\mu}, \boldsymbol{\lambda}$) model is shown in Table 3.

## Evolution Strategy

- Data: number of parents **μ** and offsprings **λ** (**λ = k· μ**).
- Initialization: create initial population **P** = {$P_i$}, **i**=1 … **λ**, and initialize the best solution **Best** ← **void**.
- WHILE {stopping criterion not met}
  - evaluate **P** and update the best solution, **Best**.
  - reproduction stage: select **μ** fittest individuals from **P** and create parent-population, **R** = {$R_j$}, **j**=1 … **μ**.
  - mutation stage: apply stochastic changes to parents and create **k = λ / μ** offsprings for each parent:

    [ **R** = {$R_j$}, **j**=1… **μ** ] → [ **Q** = {$Q_i$}, **i**=1… **λ** ]
  - Evolution stage: replace the current population with the mutated one:

    [ $P_i = Q_i$, **i**=1 … **λ** ]



## Genetic Algorithms

- Data: population size **N**, crossover rate $\eta_c$ and mutation rate $\eta_m$
- Initialization: create initial population **P**={$P_i$}, **i**=1…**N**, and initialize the best solution **Best** ← **void**.
- WHILE {stopping criterion not met}
  - evaluate **P** and update the best solution **Best**.
  - initialize offspring population: **R** ← **void**.
  - create offsprings:

    FOR **k** = 1 TO **N** / 2 DO
    - selection stage: select parents $Q_1$ and $Q_2$ from **P**, based on fitness.
    - crossover stage: use crossover rate $\eta_c$ and parents ($Q_1$;$Q_2$) to create offsprings ($S_1$; $S_2$).
    - mutation stage: use mutation rate $\eta_m$ to apply stochastic changes to $S_1$ and $S_2$ and create mutated offsprings $T_1$ and $T_2$.
    - add $T_1$ and $T_2$ to offspring population:

      $R \leftarrow R \cup \{ T_1 \text{ and } T_2 \}$.
  - replace current population P with offspring population **R**: **P** ← **R**.
  - elitism: replace the poorest solution in **P** with the best solution in **Best**.

## Differential Evolution

Differential Evolution (DE) was developed mainly by [23] as a new evolutionary algorithm. Unlike other evolutionary algorithms, DE change successive approximations of solutions or individuals based on the differences between randomly selected possible solutions. This approach uses indirectly information about the search space topography in the neighborhood of the current solution. When candidate-solutions are chosen in a wide area, mutations will have large amplitudes. Conversely, if candidate-solutions are chosen in a narrow area, mutations will be of small importance.

## Differential Evolution

For each generation, all current individuals that describe possible solutions are considered as reference solutions to which the mechanisms of DE are applied. Thus, for a reference individual $X_i$, two different individuals $X_{r1}$ and $X_{r2}$, other than $X_i$, are randomly selected and an arithmetic mutation is applied to $X_i$ based on the difference between $X_{r1}$ and $X_{r2}$, to produce a mutant $X_i'$. Then an arithmetic crossover, based on the difference between current and mutated solutions, is applied to generate the new estimation $X_i''$. $X_i''$ will replace the reference solution and the best one anytime when its fitness function is better.



## Differential Evolution

- Data: population size parents **N**, weighting factors **α, β**.
- Initialization: create initial population **P**={P$_i$}, **i**=1…**N**.
- Evaluate current population **P** and store the fittest individual as **Best**.
- WHILE {stopping criterion not met}
  - FOR **i** = 1 TO **N** DO
    - select two different individuals $\mathbf{X_{r1}}$ and $\mathbf{X_{r2}}$, other than $\mathbf{X_i}$ .
    - apply mutation: $\mathbf{X_i}' = \mathbf{X_i} + \boldsymbol{\alpha}\cdot(\mathbf{X_{r1}} - \mathbf{X_{r2}})$.
    - apply crossover: $\mathbf{X_i}'' = \mathbf{X_i} + \boldsymbol{\beta}\cdot(\mathbf{X_i} - \mathbf{X_i}')$.
    - evaluate $\mathbf{X_i}''$ and replace $\mathbf{X_i}$ with $\mathbf{X_i}''$ anytime when Fitness ($\mathbf{X_i}''$) > Fitness ($\mathbf{X_i}$).
    - update the best solution **Best**.

## Immune Algorithms

The Immune Algorithm (IA) was proposed first be [19], to simulate the learning and memory abilities of immune systems. The IA is a search strategy based on genetic algorithm principles and inspired by protection mechanisms of living organisms against bacteria and viruses. The problem coding is similar for both GA and IA, except that chromosomes in GA are called antibodies in IA, and problem formulation, i.e. objective or fitness functions are coded as antigens in IA.

The basic difference between AG and IA lies in the selection procedure. Instead of fitness functions, IA computes affinities between antibodies and / or between antibodies and antigens.

# Immune Algorithms

Based on the affinities between antibodies and antigens a selection and reproduction pool (the proliferation pool), is created using antibodies with greatest affinities. The proliferation pool is created by clonal selection: the first **M** antibodies, with highest affinities relative to antigens, are cloned (i.e. copied unchanged) in the proliferation pool. Using a mutation rate inversely proportional to the affinity of each antibody to antigens, mutations are applied to the clones. Then affinities for new, mutated clones and affinities between all clones are computed, and a limited number of clones $\mathbf{N_{rep}}$ (with lowest affinities) are replaced by randomly generated antibodies, to introduce diversity. Elitism can be applied to avoid losing best solutions.



# Immune Algorithms

- Data: population, clonal and replacement size **N, M**, $\mathbf{N_{rep}}$.
- Initialization: create initial population **P**.
- Evaluate affinities for antibodies in current population **P**.
- WHILE {stopping criterion not met}
  - clonal selection: clone the first **M** antibodies from **P**, with highest affinity. Number of clones for an antibody is proportional to its affinity. Number of clones in the proliferation pool **Q** is **N**.
  - mutation: apply stochastic changes to clones from the proliferation pool **Q**, with mutation rate inversely proportional to their affinity.
  - replacement: evaluate affinities for mutated antibodies and replace the worst $\mathbf{N_{rep}}$ clones from population **Q** with randomly generated antibodies.
  - elitism: evaluate new created antibodies and replace the worst antibody from **Q** with the best one from **P**.
  - next generation: replace current population with the one from the proliferation pool: $\mathbf{P} \leftarrow \mathbf{Q}$.

# Particle Swarm Optimization

Particle Swarm Optimization (PSO) was developed by Kennedy and Eberhart in the mid-1990s [21]. PSO is a stochastic optimization technique which emulates the “swarming” behavior of animals such as birds or insects. Basically, PSO develops a population of particles that move in the search space through cooperation or interaction of individual particles. PSO is basically a form of directed mutation.

## Particle Swarm Optimization

Any particle $\boldsymbol{i}$ is considered in two parts: particle's location $\boldsymbol{X_i}$ and its velocity $\boldsymbol{V_i}$. At any moment, the position of a particle $\boldsymbol{i}$ is computed based on its prior position $\boldsymbol{X_i}$ and a correction term proportional with its velocity $\boldsymbol{\varepsilon \cdot V_i}$. In its turn, the velocity assigned to each particle is computed using four components: (i) the influence of the previous value of velocity $\boldsymbol{V_i}$; (ii) the influence of the best personal solution for particle $\boldsymbol{i}$, $\boldsymbol{X_i^P}$; (iii) the influence of the best local solution so far for informants of particle $\boldsymbol{i}$, $\boldsymbol{X_i^L}$ and (iv) the influence of the best global solution so far for the entire swarm, $\boldsymbol{X^G}$. These components are taken into consideration using three weighting factors, denoted by $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$.



## Particle Swarm Optimization

- Data: population size **N**, personal-best weight **α**, local-best weight **β**, global-best weight **γ**, correction factor **ε**.
- Initialization: create initial population **P**.
- WHILE {stopping criterion not met}
  - select the best solution from the current **P**: **Best**.
  - select global-best solution for all particles: $\mathbf{B^G}$.
  - apply swarming:

    FOR **i** = 1 TO **N** DO
    - select personal-best solution for particle $\mathbf{X_i}$: $\mathbf{B_i^P}$.
    - select local-best solution for particle $\mathbf{X_i}$: $\mathbf{B_i^L}$.
    - compute velocity for particle $\mathbf{X_i}$:

      FOR **j** = 1 TO **DIM** DO
      - generate correction coefficients: $\mathbf{a} = \boldsymbol{\alpha} \cdot \text{rand}()$ ; $\mathbf{b} = \boldsymbol{\beta} \cdot \text{rand}()$ ; $\mathbf{c} = \boldsymbol{\gamma} \cdot \text{rand}()$.
      - update velocity of particle $\mathbf{X_i}$ along dimension **j**:

        $$\mathbf{V_{ij}} = \mathbf{V_{ij}} + \mathbf{a} \cdot (\mathbf{B_{ij}^P} - \mathbf{x_{ij}}) + \mathbf{b} \cdot (\mathbf{B_{ij}^L} - \mathbf{x_{ij}}) + \mathbf{c} \cdot (\mathbf{B_j^G} - \mathbf{x_{ij}})$$
    - update position of particle $\mathbf{X_i}$: $\mathbf{X_i} = \mathbf{X_i} + \boldsymbol{\varepsilon} \cdot \mathbf{V_i}$



## Ant Colony Optimization

- Data: population size **N**, set of components $\mathbf{C} = \{C_1, \ldots, C_n\}$, evaporation rate **evap**.
- Initialization: amount of pheromones for each component $\mathbf{PH} = \{PH_1, \ldots, PH_n\}$; best solution **Best**
- WHILE {stopping criterion not met}
  - initialize current population, **P** = **void**.
  - create current population of virtual solutions **P**:

    FOR **i** = 1 TO **N** DO
    - create feasible solution **S**.
    - update the best solution, **Best** ← **void**.
    - Add solution **S** to **P**: $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{S}$
  - Apply evaporation:

    FOR **j** = 1 TO **n** DO

    $PH_j = PH_j \cdot (1 - \mathbf{evap})$
  - Update pheromones for each component:

    FOR **i** = 1 TO **N** DO

    FOR **j** = 1 TO **n** DO
    - if component $C_i$ is part of solution $P_j$, then update pheromones for this component: $PH_j = PH_j + \text{Fitness}(P_j)$

## Honey Bee Colony Optimization

The Honey Bee Colony Optimization (HBCO) algorithm was first proposed in [24]; it is a search procedure that mimics the mating process in honey-bee colonies, using selection, crossover and mutation.

A honey bee colony houses a queen-bee, drones and workers. The queen-bee is specialized in egg laying; drones are fathers of the colony and mate with the queen-bee. During the mating flight the queen mates with drones to form a genetic pool. After the genetic pool was filled with chromosomes, genetic operators are applied.

## Honey Bee Colony Optimization

During the crossover stage, drones are randomly selected from the current population and mate with the queen using a Simulated Annealing-type acceptance rule based on the difference between the fitness functions of the selected drone and the queen. The final stage of the evolutionary process consists in raising the broods generated during the second stage, and creating a new generation of $\mathbf{N_B}$ broods, based on mutation operators. A new generation of $\mathbf{N_D}$ drones is created based on a specific selection criterion.



## Honey Bee Colony Optimization

- Data: size of populations: - drones ($\mathbf{N_D}$), broods ($\mathbf{N_B}$) and genetic pool ($\mathbf{N_P}$); initial queen's speed $\mathbf{S_{max}}$; crossover rate $\boldsymbol{\eta_c}$ and mutation rate $\boldsymbol{\eta_m}$
- Initialization: create initial population **Drones** with $\mathbf{N_D}$ individuals, and select the best drone as the **Queen**.
- WHILE {stopping criterion not met}
    - create the genetic pool: use **Drones** population and select $\mathbf{N_P}$ individuals using a Simulated Annealing-type acceptance rule, based on the queen's speed **S**, and gradually reduce **S**.
    - crossover: apply arithmetic crossover between **Queen** and successively selected drones from the genetic pool, until a population of $\mathbf{N_B}$ broods (offsprings) is created.
    - mutation: apply arithmetic mutation to randomly selected broods (offsprings).
    - update the **Queen**: if any brood is better than the **Queen**, update the **Queen**.
    - selection: use broods and, based on a selection criterion, create the new population of **Drones**.

## Applications of (meta)heuristic methods in power systems



## Types of applications

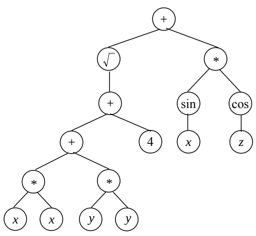
- Load assessment and profiling
- Network reconfiguration
- Reactive power planning
- System security analysis
- State estimation
- Distributed generation

any many others ….



## Load assessment and profiling

### Peak load estimation




- **Dataset**: hourly load profiles
- **Problem**: find the approximant (analytical) that best fits the daily peak load using as input different combinations of data from the dataset.
- **Approach**: **Genetic Programming & Symbolic Regression**.
- **Solution-inputs**: peak loads form days d-7, d-6 and d-1 and the number of the reference day in the year.

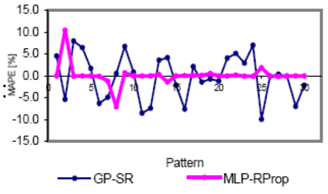
Applications 3/36

## Comparison of the GP&SR approach to an ANN-MLP model

Function approximation (GP&SR):

$$f1 = \cos(\cos(\sin x_3)) - \cos(x_3)$$
$$f2 = \cos(\cos(\cos x_3)) - \cos(x_3)$$
$$f = 2 \cdot \cos(x_3) + x_1 - \log(x_2) - 2 \cdot \log(\log(x_4)) - - \log(f1) - 2 \cdot \log(f2)$$

Estimation errors (GP&SR and ANN-MLP):




Applications 4/36

# Load assessment and profiling

## Load profiling (1)

- **Dataset**: hourly load profiles
- **Problem**: load profile clustering and typical load profiles generation.
- **Approach**: Affinity Control inspired from Immune Algorithms and SOFM.
- **Solution**: portfolio of typical load profiles.

Ctrl =

Applications 5/36

## Affinities between LPs and affinity degrees

Affinities between TLPs and metered LPs:

$$A_{i,j}^{T-T} = \sqrt{\left\| W_i - W_j \right\|} = \sqrt{\sum_{k=1}^{NK} \alpha_k \cdot \left( w_{i,k} - w_{j,k} \right)^2}$$

$$A_{i,p}^{T-M} = \sqrt{\left\| X_p - W_i \right\|} = \sqrt{\sum_{k=1}^{NK} \alpha_k \cdot \left( x_{p,k} - w_{i,k} \right)^2}$$

Affinity degrees:

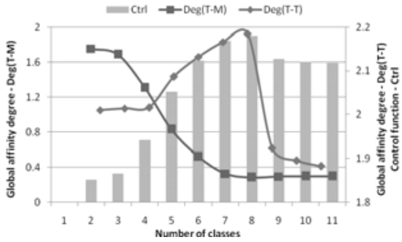
$$Deg^{T-T} = \frac{2}{NC \cdot (NC-1)} \cdot \sum_{i=1}^{NC-1} \sum_{j=1}^{NC} A_{i,j}^{T-T}$$

$$Deg^{T-M} = \frac{1}{NC} \cdot \sum_{i=1}^{NC} \sum_{p=1}^{NP} \left( A_{i,p}^{T-M} \Big/ \sum_{p=1}^{NP} \delta_{i,p} \right)$$

Control function:

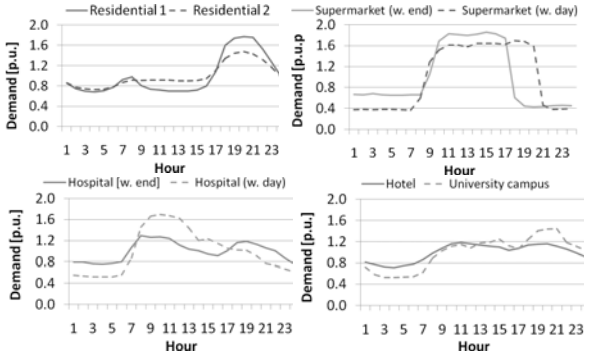
$$Ctrl = \beta \cdot Deg^{T-T} - (1-\beta) \cdot Deg^{T-M}$$

## Global affinity degrees and the control function during the growing phase






## Results






# Load assessment and profiling

## Load profiling (2)

- **Dataset**: hourly load profiles
- **Problem**: load profile clustering and typical load profiles generation.
- **Approach**: Honey bee mating optimization algorithm.
- **Solution**: portfolio of typical load profiles.

## Average distances between LP vs. Affinities

Average distances LP-TLP

$$\sigma_t = \frac{1}{NP_t \cdot NH} \cdot \sum_{i=1}^{NP} \| LP_i - TLP_t \| \quad Class(i) \equiv t \quad t = 1, \ldots, NT$$

$$\sigma = \frac{1}{NT} \cdot \sum_{t=1}^{NT} \sigma_t$$

Average distances TLP-TLP

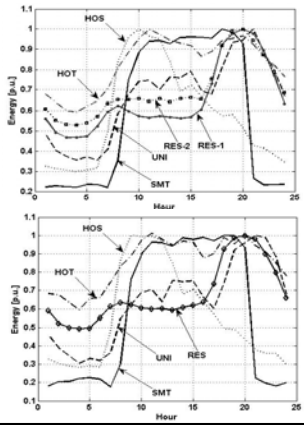
$$\rho_{s,t} = \frac{1}{NH} \cdot \| TLP_s - TLP_t \| \quad s,t = 1, \ldots, NT$$

$$r_s = \frac{1}{NT-1} \cdot \sum_{t=1}^{NT} \rho_{s,t} \quad s = 1, \ldots, NT \qquad R = \frac{1}{NT} \cdot \sum_{s=1}^{NT} r_s$$

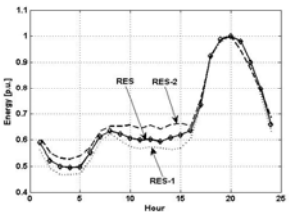
Fitness functions:

$$F_1 = 1/\sigma = 1/(\sum_{t=1}^{NT} \sigma_t / NT) \qquad F_2 = R - \sigma = \frac{1}{N_T} \cdot \sum_{s=1}^{NT} r_s - \frac{1}{N_T} \cdot \sum_{t=1}^{NT} \sigma_t$$






## Results






# Network reconfiguration

The network reconfiguration problem arises usually in distribution systems and aims at changing the network topology by altering the position and status of sectionalizing switches.

A complex combinatorial problem.

# Network reconfiguration

## Problem formulation (RPP included)

$$Min\left(\Delta P_{network}\right) = Min\left(\sum_{l=1}^{NL} \Delta P_l^{Line} + \sum_{t=1}^{NT} \Delta P_t^{Transf}\right)$$

subject to:

- Voltage constraints: $V_t^{\min} \leq V_t \leq V_t^{\max} \quad t=1,\dots,NT$
- Line capacity constraints: $I_{l,1} \leq I_{\max\, adm} \quad l=1,\dots,NL$
- Capacitor constraints: $N_t \cdot \Delta Q_K \leq Q_t^{\max} \quad t=1,\dots,NT$ ; $\sum_{t=1}^{NT} N_t \leq K_T$



## Network reconfiguration (1)

- **Dataset**: network topology, network data, load data.
- **Problem**: determine the optimal combination of opened sectionalizing switches and the optimal RPP.
- **Approach**: Ant Colony Optimization.
- **Solution**: optimal configuration of the network that minimize power losses.



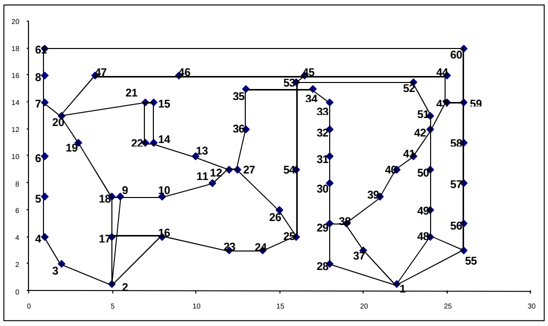
## Network reconfiguration (1)

**Solution representation:**

- Reconfiguration problem:
  - A candidate solution will be represented by a vector consisting of the load sections that will be opened to obtain the radial configuration of the network.
- RPP problem:
  - A candidate solution will be a vector, which shows how many capacitor banks are placed in each node of the network.

## Network reconfiguration (1)

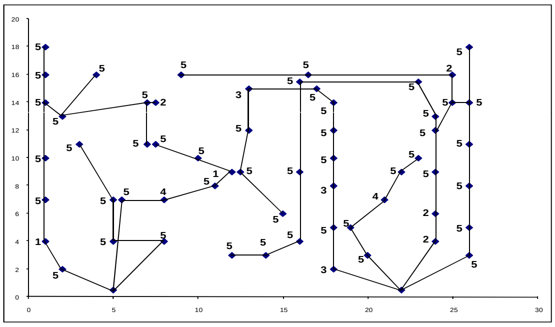
**Test system:**





## Network reconfiguration (1)

**Solution:**





## Network reconfiguration (2)

- **Dataset**: network topology, network data, load data.
- **Problem**: determine the optimal combination of opened sectionalizing switches and the optimal RPP.
- **Approach**: Particle Swarm Optimization (network reconfiguration) and Immune Algorithm (RPP).
- **Solution**: optimal configuration of the network that minimize power losses.

## Network reconfiguration (1)

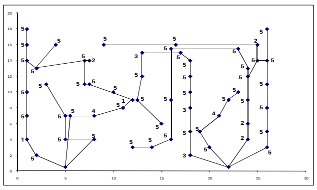
**Solution representation:**

A candidate solution encodes two *m*-tuples of indices. The first *m*-tuple indicates the feeders in the system where the sectionalizing switches should be opened, and the second *m*-tuple shows the line sections on each feeder where the switches must be turned off.



## Network reconfiguration (2)

**Example**



- The same test system.
- The same optimal solution
- Comparable execution time.



# Reactive Power Planning

Reactive power control:

- generator voltage control,
- transformer tap control and
- fixed or controllable VAR sources.

In distribution systems: Reactive Power Compensation (RPC) through power factor correction.

## Reactive Power Planning

- **Dataset**: network topology, network data, load data.
- **Problem**: determine the optimal location and number of capacitors in the network.
- **Approach**: An enhanced Particle Swarm Optimization algorithm.
- **Solution**: optimal reactive power compensation that minimize power losses.



## Reactive Power Planning

The enhanced Particle Swarm Optimization algorithm.

At each iteration a particle should try to mimic not only the best positions so far, but also other successful positions of itself and other particles in the current population.

$$v_i^{t+1} = c_0 \cdot v_i^t + c_1 \cdot \sum_{k=1}^{NT} r_{1,k} \cdot \left(B_{i,k}^L - X_i^t\right) + \\ + c_2 \cdot \sum_{k=1}^{NT} r_{2,k} \cdot \left(B_k^G - X_i^t\right)$$



**Table 1** – Values of objective functions after 10 runs of three searching algorithms.

## Reactive Power Planning

Comparison with other 2 algorithms.

| Run # | GA | IA | PSO |
|---|---|---|---|
| 1 | 2.6421 | 2.6206 | 2.6161 |
| 2 | 2.6368 | 2.6206 | 2.6155 |
| 3 | 2.6376 | 2.6217 | 2.6153 |
| 4 | 2.6343 | 2.6173 | 2.6126 |
| 5 | 2.6279 | 2.6196 | 2.6136 |
| 6 | 2.6280 | 2.6198 | 2.6126 |
| 7 | 2.6302 | 2.6209 | 2.6071 |
| 8 | 2.6275 | 2.6197 | 2.6209 |
| 9 | 2.6400 | 2.6203 | 2.6128 |
| 10 | 2.6416 | 2.6215 | 2.6120 |
| Mean value | 2.6346 | 2.6202 | 2.6139 |

# System security analysis

System equivalents are a good solution to simplify the on-line analysis of present day wide-area power systems used in assessing system security.



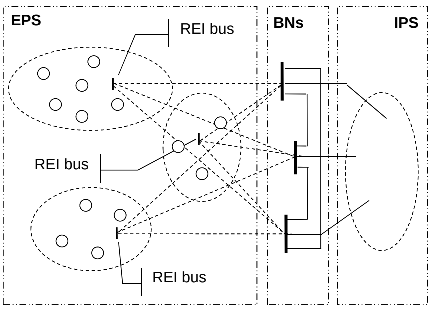
## System security analysis

- **Dataset**: network topology, network data, load data.
- **Problem**: determine the optimal structure of the REI equivalent.
- **Approach**: Genetic Algorithm.
- **Solution**: How to group nodes in the REI equivalent ?



## System security analysis

REI equivalent optimization



## System security analysis

**Solution representation (IEEE 57 test system):**

| **Genes** | 2 | 3 | 1 | 3 | 2 | 2 | 2 | 3 | 1 | 2 | 4 | 1 | 4 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bus #** | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 58 |

**REI node # 1: group (4, 11, 14)**

**REI node # 2: group (2, 6, 8, 9, 12, 17)**

**REI node # 3: group (3, 5, 10, 58)**

**REI node # 4: group (13, 16)**

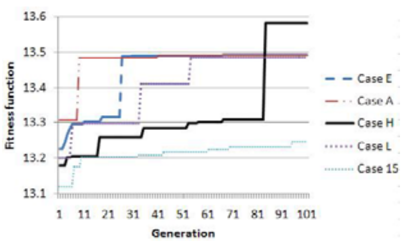
**Objective function:**

$$FO = \frac{1}{NC \cdot NI} \cdot \sum_{i=1}^{NI} \sum_{j=1}^{NC} \frac{\left| U_{i,j}^{ref} - U_{i,j}^{eq} \right|}{U_{i,j}^{ref}} \cdot 100$$



## System security analysis

**Results (IEEE 57 test system):**




| Case | # REI nodes |
|---|---|
| A | 2 |
| E | 3 |
| H | 4 |
| L | 5 |
| 15 | 15 |

**Optimal solution:**

**REI node # 1: group (4, 11, 14)**
**REI node # 2: group (2, 6, 8, 9, 12, 17)**
**REI node # 3: group (3, 5, 10, 58)**
**REI node # 4: group (13, 16)**



## State estimation

- New applications for SE are largely based on Synchronized Phasor Measurement Units (SPMU).
- A hybrid solution which uses both traditional and SPMU is better in terms of costs.

## State estimation

- **Dataset**: network topology, network data, SCADA measurements, PMU measurements.
- **Problem**: determine the optimal location of PMU measurements
- **Approach**: Genetic Algorithm.
- **Solution**: Buses where PMU measurements must be located.



## State estimation

**Solution representation (IEEE 14 test system):**

A chromosome consists of M blocks, each one associated to a PMU. A block describes the binary code of the bus where the PMU is placed.

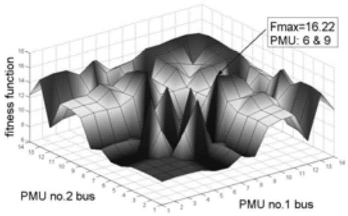
A possible solution for a 2 PMUs placement-problem may use bus number 6 and 9, and the chromosome describing this solution will have the structure:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |



## State estimation

**Error surface for the case of 2 PMUs**

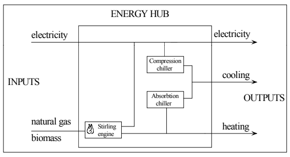


# Distributed generation

Present day distribution systems are facing deep changing that transforms traditional design for passive operation into new concepts centered on distributed generation (DG) and a more active role of end-users.



## Optimal design of multiple energy hubs

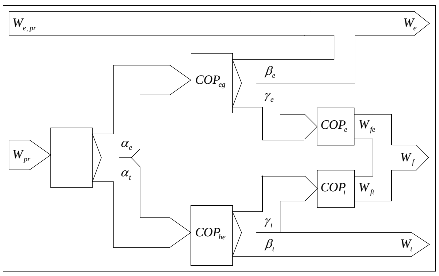



- **Dataset**: inputs and outputs of the energy hub, output loads, parameters of conversion units.
- **Problem**: determine the optimal input structure to supply the output loads.
- **Approach**: Genetic Algorithm.
- **Solution**: Inputs of the energy hub.



## Optimal design of multiple energy hubs

**Solution representation**




A chromosome with 3 real coded genes:

| | | |
|---|---|---|
| $\alpha_e$ | $\beta_e$ | $\beta_t$ |

## Optimal design of multiple energy hubs

The objective function:

$$F_{obj} = k \cdot (W_{pr} + W_{e,pr}) + (1-k) \cdot dW_{pr}$$

where:

$F_1 = W_{pr} + W_{e,pr}$ - the primary energy function

$F_2 = dW_{pr}$ - error term equal to the difference between the primary energy computed on two independent ways

$k$ - weighting coefficient

# Conclusions



## Conclusions

During the last two decades numerous (meta)heuristic approaches have been devised and developed to solve complex optimization problems.

Their success is due largely to their most important features:

- simplicity,
- the need of minimal additional knowledge on the optimization problem and
- a highly numerical robustness of algorithms.

# Thank you for your attention!

Contact information:

Mihai Gavrilas
Faculty of Electrical Engineering
21 D. Mangeron Blvd.,
Iasi, 700050, ROMANIA

E-mail: mgavril@ee.tuiasi.ro
Phone: +40 723 625279
Fax: +40 232 237687