Introduction to Nature Inspired Optimization*.

Vicente Martín

FIM-UPM

v 0.1
Feb. 2014

* Including an intro to Monte Carlo Methods.

- Biologically Inspired Optimization Methods:
  - Evolution: Evolution Strategies, Genetic Algorithms, Genetic Programming...
    - Fuzzy borders... mixing characterstics.
  - Neural Networks (not seen here).
  - Immune Systems.
  - Foraging, herds, flocks strategies...
  - Physically based Strategies: Simulated Annealing
- Plus: A Brief intro to Monte Carlo Methods.

Some Names and Dates.

- 1948, Turing:
  proposes "genetical or evolutionary search"
- 1962, Bremermann
  optimization through evolution and recombination
- 1964, Rechenberg
  introduces evolution strategies
- 1965, L. Fogel, Owens and Walsh
  introduce evolutionary programming
- 1975, Holland
  introduces genetic algorithms
- 1983 S. Kirpatrick, C.D. Gelatt, M.P. Vecchi.
  "Optimization by Simulated Annealing"
- 1983 W.T. Reeves.
  Introduces Particle Systems for modeling fuzzy objects
- 1987, Reynolds
  Modelos de movimiento de bandadas para animación

Some Names and Dates.

- 1992, Koza
  introduces genetic programming
- 1992, Coloni, Dorigo, Maniezzi
  "Distributed optimization by ant colonies"
- 1994, Forrest et al.:
  "A Self-Nonself Discrimination on a Computer"
- 1995 Eberhart, Kennedy .
  Particle swarm optimization

Evolutionary Computing.

- The basic idea is to evolve an initial *population* made up of *individuals*, each one coding a possible solution to the problem, till finding the optimum.
  - How to encode the individual?
  - How to evolve the individual?
    - A „positive“ direction for the evolution has to be defined.
      - Each individual is evaluated.
    - A set of operators and application schemes must be defined to evolve the initial population till reaching the problem's solution.
      - Operators are applied as dictated by the results of the individuals' evaluation..

- The „Canonical“ evolutionary program:

```
generation=0
Initialize Population P(0)
Evaluate Population(0)
  Repeat till termination condition is met:
      generacion <-  generation+1
      Create Population(generation) from Population(generation-1)
      Modify Population(generation)
      Evaluate Population(generation)
  End Repeat
End
```

- Each individual of the population must code a possible solution to the problem.
  - individual ≡ cromosome
- The selected codification is key for the algorithm's success.
  - Binary: The individual is represented as a bit string.
    - Advantages (will be discussed afterwards):
      - Easy to implement „canonical“ operators.
        - But those adapted for a specific problem can be difficult...
      - „Favoured“ by the scheme theorem.
      - Minimal cardinality alphabet.
    - Disadvantages (to be discussed later on):
      - Limited Range (for fixed cromosome lengths)
      - The translation among the genotype (algorithm representation space) and phenotype (problem space) might be costly.

- Real coding: A set of floating points (IEC-589 representation) are used to code an individual.
  - Advantages:
    - A much higher representation range for a fixed number of bits ($\sim\pm 10^{38}$ in SP (32 bits/number), $\sim\pm 10^{307}$ in DP (64 bits)) and related advantages (constant relative error)
    - Translation among coding space and problem space is usually easier.
  - Disadvantages:
    - More complicated canonical operators... but easier to adapt to the specific problem.
- Symbolic: The individual is codified using a character string or a more sophisticated data structure.
  - If the problem requires so... e.g.: Using a tree (genetic programming)

- Codings:
  - Cover the full solution space without bias (except for good reasons...)
  - Avoid to code non feasible individuals or to code several times the same individual (or at least, minimize the chance)
  - Decoding: translation from the algorithmic representation space (genotype) to problem space (phenotype) should be fast (or unnecessary except at the beginning/end)
  - Cromosomes with „near“ genotypes should correspond to individuals with „near“ phenotypes.
  - Should naturally produce „good“ genetic operators: those preserving characteristics with high fitness from one generation to the next and also efficient from a computational viewpoint.

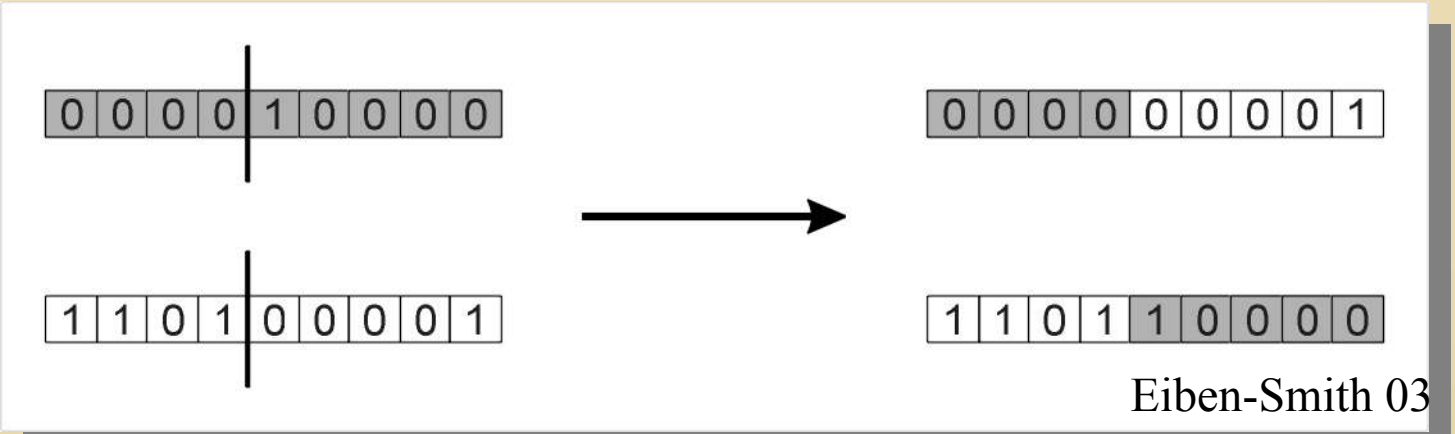
- Population(0) is usually initiallized randomly.
  - Bias sampling can be used... as long as the problem justifies it.
    - It is usual that this produces improvements only at the beginning of the algorithm, but it is many times useless.
  - A systematic test on the generated individual, to test its feasibility before going on, can be applied.
    - The selection of a good coding minimizes the possibility to generate individuals that cannot lead to a solution.
    - Systematic tests are more sensible within evolutionary strategies. In genetic algorithms it is many times worthwhile to allow for invalid candidates, although with a highly reduced fitness value, than to elliminate them.

- Evaluation: Fitness.
  - The fitness function measures how well the individual satisfies the problem's conditions.
  - It is usually defined in a way that higher values indicate a better solution.
    - E.g.: If two different distributions are compared, we could be interested in minimizing some distance definition, such as $\|.\|_2$, hence we could define the fitness function as $1/\|.\|_2$
      - Note that definitions such as the previous one could lead to very big fitness values: the way in which the fitness function scales is key for the correct working of the algorithm, since are these values the ones used to apply the different operators, since it regulates the „evolutionary pressure“.
      - It could be better to use $1/(\varepsilon+\|.\|_2)$, as the fitness, adjusting the ε value for a correct convergence.

- According to its fitness, each individual is assigned a rank order in the population.
- Based in its rank, its fitness or a function of it, the intermediate Population(1) is obtained from Population(0).
  - Examples:
    - From Population(0) a fixed number of the lower ranked individuals are eliminated and those with the best rank are duplicated.
    - Individuals from Population(0) are copied to Population(1), with a (normalized) probability dependent on its fitness. E.g.: If a given individual's fitnness is twice as big as the mean, it will be copied two times on average. If it is only 0.5 it will have just a 50% chance of continuing in Population(1), etc.
      - Note that the intermediate population is an „effective“ population, used to apply the genetic operators. It does not need to be explicitly generated.
      - The intermediate population does not need to be explicitly created: e.g.: in a typical evolution strategy, the population is just a father and a son, product of a mutation of the father. The best of the two survives and mutes again.

- The intermediate population is modified using a set of genetic operators:
  - Crossover: A new Chromosome is formed from two (or more) parents.
  - Mutation: Add variability to the genes composing the chromosome.
  - Reordering: Change the order of the genes to increase the probability of having „constructor blocks“: pieces of the chromosome that carry a high probability of having a high fitness value.
    - Crossover and mutation are the typical ones. Reordering is usually dismissed. The most used variant is the „inversion“, that inverts the order of the genes in a given chromosome segment.
      - A gene is (usually) the piece of the chromosome that codes one of the parameters of the problem.

- Crossover operators : segments of the parents' chromosomes are used to compose a new chromosome. The most usual techniques:
  - One point crossover.
  - Two points crossover.
  - Uniform crossover.
- Example: Typical one point crossover.
  - Randomly select one crossover point in te parents chromosomes.
  - Interchange the parts of the chromosome beyond that point among the parents to produce two new individuals.



- Positional bias: One point crossover will not keep together good genes located in the first and last positions of the chromosome.

- Ex: Uniform crossover.
  - For each gene of the first children, the corresponding gene from the first or second parent, with a 50% probability, is copied. In the second children is copied the gene of the other parent.
- Ex: Crossover that preserves some of the solution properties.
  - Ex: In the travelling salesman problem, the possible solutions are the cities permutations. PMX: Partially mapped crossover:

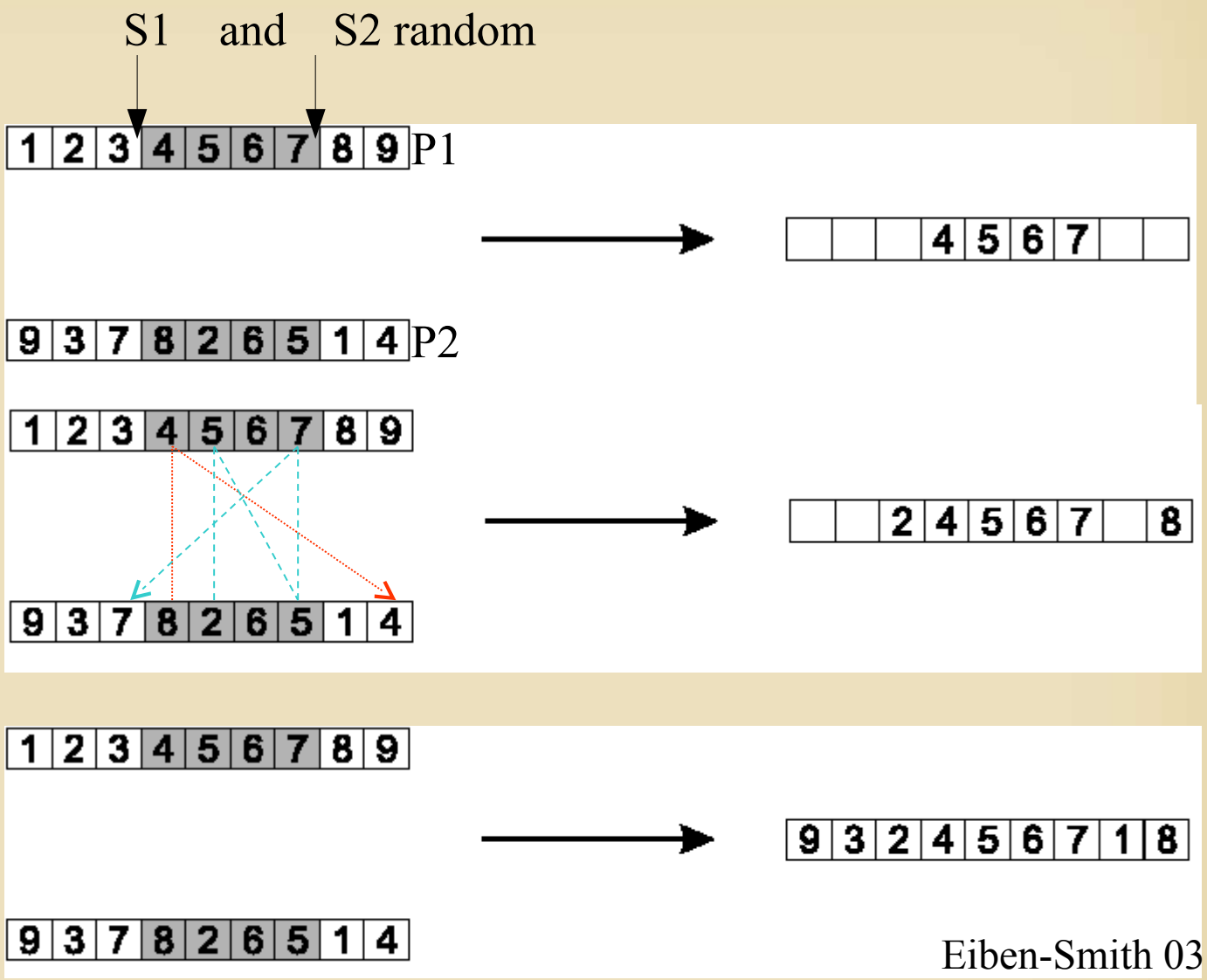
Parents: P1 y P2

children 1

- Step 1: The locations among S1 and S2 will be copied to the first children.

- Step 2: The elements displaced in P2 will be written in the son using the element in the same position of P1 as a pointer (using the values from P2 if the resulting position are in between S1 and S2)

•Step 3: The rest of the elements are copied from P2.



Eiben-Smith 03

- Ex: Floating point crossover (intermediate crossover).
  - If Xi, Yi are the values of the parent's genes, the corresponding son's gene Zi is obtained as:

$$Zi = \alpha\, Xi + (1-\alpha)Yi$$

    - $0 \leq \alpha \leq 1$ can be either randomly chosen, fixed during all of the run, or varying while the algorithm runs, taking into account the upper/lower bounds allowed, etc...
- We are not forced to consider always just two parents. More could be used.
  - „arity“ of the operator.
- Different crossover operators behave differently in regard to some basic properties considered in the EC theory (ex.: disruptivity)

- Mutations: Alter the genes with a given probability.
- Ex: Binary mutation in the „canonical“ algorithm.
  - Modify each parent's bit with a fixed probability, independent of the bit's value.
  - Typical values for the mutation rate are among the inverse of the population size and the inverse of the length of the chromosome.
    - The words „Mutation rate“ are used quite often and care must be taken in its interpretation. Its meaning will be different as used in the canonical algorithm -just described- or in a numerical optimizacion problem with real coding.
- Ex: Mutation for permutations (integer coding):
  - Insertion/interchange mutation: Take two arbitrary points in the chromosome and insert/exchange the value in the first point in the place/the value in the second point.

- Ex: Mutation for genetic programming:
  - Select a point in the tree and delete everything down the point, substituting it by a set of randomly generated branches (till a previously chosen depth)
- Ex: Real coding mutation:
  - Uniform Mutation : take a randomly chosen gene and change its value by a random value (generated among the allowed values for the gene).
  - Non-uniform mutation: The gene k chosen for the mutation (with value v_k in $[l_k,u_k]$) changes to a value v_k' chosen with a 50% probability between:

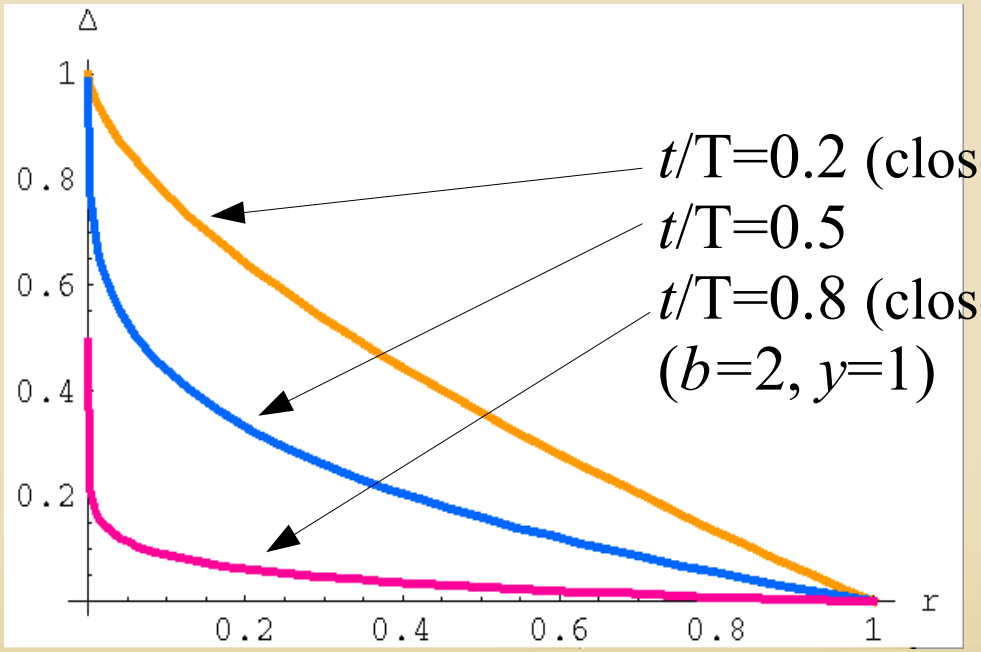
$$v_k' = \begin{cases} v_k + \Delta(t, u_k - v_k) \\ v_k + \Delta(t, v_k - l_k) \end{cases}$$

With the function::

$$\Delta(t, y) = y\left(1 - r^{\left(1-\frac{t}{T}\right)^{b}}\right)$$

b is a given parameter. *t* is the population sequence number among the total *T* allowed. *r* is a random number in [0...1]. *y* is the allowed range.

- This operator tries to avoid the premature convergence phenomenon by making bigger mutations (bigger variation) more probable at the beginning (small *t*) than at the end.



- Selection of the parents:
  - Proportional selection.
  - Tournament selection.
  - Uniform state selection.
- Proportional Selection:
  - It can converge prematurely (high evolutionary pressure): One (or just a few) individual(s) rapidly inherit all the probability of being parents.
  - Ex: Roulette method:

    1) Add up all the fitness values for all the individuals in the population (normalize to $P=\Sigma_i P_i$

    2) Assign to each individual the normalized area corresponding to the roulette.

    3) Use the roulette n times (the population size) to select n individuals.

    - A direct implementation is slow. The worse individuals have a non zero reproduction probability. The number of times that each individual is selected can be different from its expected value(P_i/P).

- Ex: Stocastic remaining:
  - Calculate the expected value of every individual fitness (P_i/P).
  - Copy each individual the number of times that indicate the integer part of the obtained value.
  - The remaining of the integer part is used to bias the rest of the individuals left till complete the population.
    - Without replacement: Each remaining is used individually (using a random number among 0 and 1 and comparing it with the remaining. This is the most used)
    - With replacement: The remains of all of the individuals are used to make a roulette that is used as many times as needed individuals.
  - The stocastic remain reduce the problems with the roulette wheel but they are not elliminated.

- Ej: Sigma Scaling:
  - Tries to control premature convergence by mantaining a constant selection pressure.
  - The selection is done according to the expected value of the i-th individual in generation t, $P_t(i)$, modified as a fnction of the population average and its width (using the standard deviation, f= fitness):

$$P_t(i) = \begin{cases} 1 + \dfrac{f(i) - \bar{f}_t}{2\sigma_t} & if\ \sigma_t \neq 0 \\ 1 & if\ \sigma_t = 0 \end{cases}$$

$$\sigma_t = \sqrt{\frac{n\sum f_t^2 - \left(\sum f_t\right)^2}{n^2}}$$

    - High values of σ (i.e: in a population far from convergence, as at the beginning of the run) will forbid that the highly fitted individuals gather all of the probability (low evolutionary pressure). As it gets closer to convergence, the width will be lower and the better individuals will reproduce more easily, making easier the last steps of the convergence.

- Ex: Boltzmann selection.
  - Related to „simmulated annealing“
  - Again, it redfines the expected value:

$$P_t(i) = \frac{e^{\frac{f_i}{T}}}{\left\langle e^{\frac{f_i}{T}} \right\rangle}$$

T is now the „temperature“, that will be defined according to the problem (and that, in general, will be a function of the convergence)

- Ex: Rank Selection.
  - Rank the population from 1 to N (population size), with 1 the best fitted and N the least.
  - Parameterize the expected value as a function of the rank.
    - A usual (linear) parameterization is:

$$P_t(i) = m + (M - m)\frac{rank_t(i) - 1}{N - 1}$$

Where M is chosen among [1,2] and m=2-M. Again, the *t index* refers to the generation.

    - Non-linear parameterizations can be used if the problem requires so.
  - A proportional rule is used (e.g. roulette) using the expected values.
  - It dilutes the evolutionary pressure, usually avoiding premature convergence. It is also useful when the fitness function provides values with small (non significant) absolute differences (e.g.: with noisy functions)

- Tournament selection: (non proportional)
  - Use *p randomly* chosen (typically 2) elements od the population.
  - Compare fitness..
  - The tournament winner is:
    - The most fitted (determinist strategy)
    - Choose a random [0,1) individual. If it is smaller than a given *x*, a value previously fixed for all the run and also in [0,1), the highest fitted is the winner. If not, choose the least fitted (probabilistic strategy)

Selection pressure can be regulated changing the *p* value. If *p*=1 we have a random selection with a very low selectio pressure. If p=∞ the best individuals are always selected. *p*≥10 values are used for „hard“ selections, values among 2 and 5 for „soft“ selections.

- Uniform state selection: (non proportional)
  - Only a group of individuals is replaced in each generation (non generational algorithm).
  - An explicit memory is kept
    - Elitism strategies.
  - R individuals among the most fitted are selected from the original population M.
    - Apply crossover and mutation to these individuals, choose the best μ.
    - The worst μ individuals are substituted with these.

Typical Example

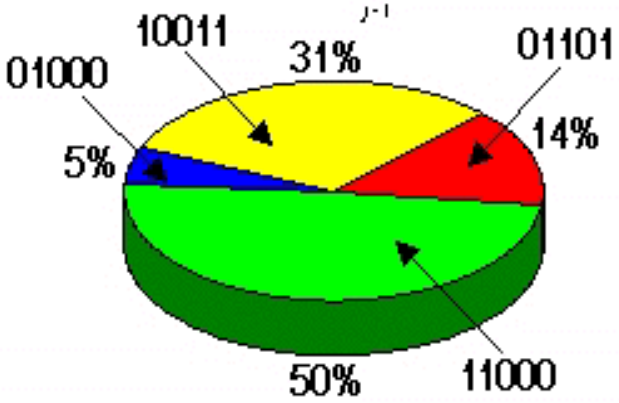
Operators Example

The Problem is to Maximize $f(x) = x^2$

Number	String	Fitness	% of the Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

1.- *Roulette Wheel Selection*

$$P_s = \frac{f_i}{\sum_{j=1}^{n} f_j}$$



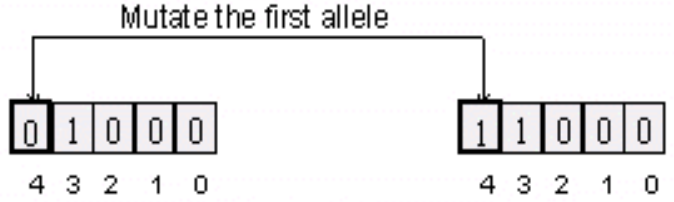
2.- *One-Point Crossover (SPX)*

$P_c \in [0.6 .. 1.0]$

parents	offspring
01|101 (169)	01000 (64)
11|000 (576)	11101 (841)

3.- *Mutation*

$P_m \in [0.001 .. 0.1]$



de Evonet

- Parameters governing an EA... (Only as a guideline. Note that many estimates are based only in very specific algorithms and test problems. De Jong, Schaffer)
  - On Population size:
    - Goldberg (1985), canonical algorithm, binary coding:

$$M = 1.65^{(2^{0.21L})}$$

L=Length, in bits, of the chromosome.

      - Very big... it has been broadly critizised.
    - Empirical rule: at least $2L$.
    - Theoretical support for micropopulations (N=3).
  - Crossover and mutation percentage:
    - Guidelines (binary coding):
      - Crossover: 0.75-0.95
      - Mutation: 0.005 - 0.01
  - Metaalgorithms:
    - Evolve the parameters with the solution.

Examples of functions used to judge GAs performance.

(Digalakis et al. „An Experimental Study of Benchmarking Functions for Genetic Algorithms“ Inter. J. Computer. Math. 79 (2002) 403)

3. SET OF BENCHMARK FUNCTIONS

Taking the above under consideration and given the nature of our study, we concluded to a total of 14 optimization functions. Table I summarizes some of the

TABLE I Unimodal and multimodal functions (F1–F8)

Name	*Function*	*Limits*
F1	$f_1 = \sum_{i=1}^{2} x_i^2$	$-5.12 \le x_i \le 5.12$
F2	$f_2 = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2.048 \le x_i \le 2.048$
F3	$f_3 = \sum_{i-1}^{5} int.(x_i)$	$-5.12 \le x_i \le 5.12$
F4	$f_4 = \sum_{i=1}^{30} (ix_i^4 + Gauss(0,1))$	$-1.28 \le x_i \le 1.28$
F5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \left(\frac{1}{j} + \sum_{i=1}^{2} (x_i - a_{ij})^6 \right)$	$-65536 \le x_i \le 6553($
F6	$f_6 = 10V + \sum_{i=1}^{10} (-x_i, \sin(\sqrt{\lvert x_i \rvert})), V = 4189.829101$	$-500 \le x_i \le 500$
F7	$f_7 - 20A + \sum_{i=1}^{20} (x_i^2 - 10\cos(2\pi x_i)), A = 10$	$-5.12 \le x_i \le 5.12$
F8	$f_8 = 1 + \sum_{i=1}^{10} \left(\frac{x_i^2}{4000} \right) - \prod_{i=1}^{10} \left(\cos\left(\frac{x_i}{\sqrt{i}} \right) \right)$	$-600 \le x_i \le 600$

TABLE II Non linear squares problems

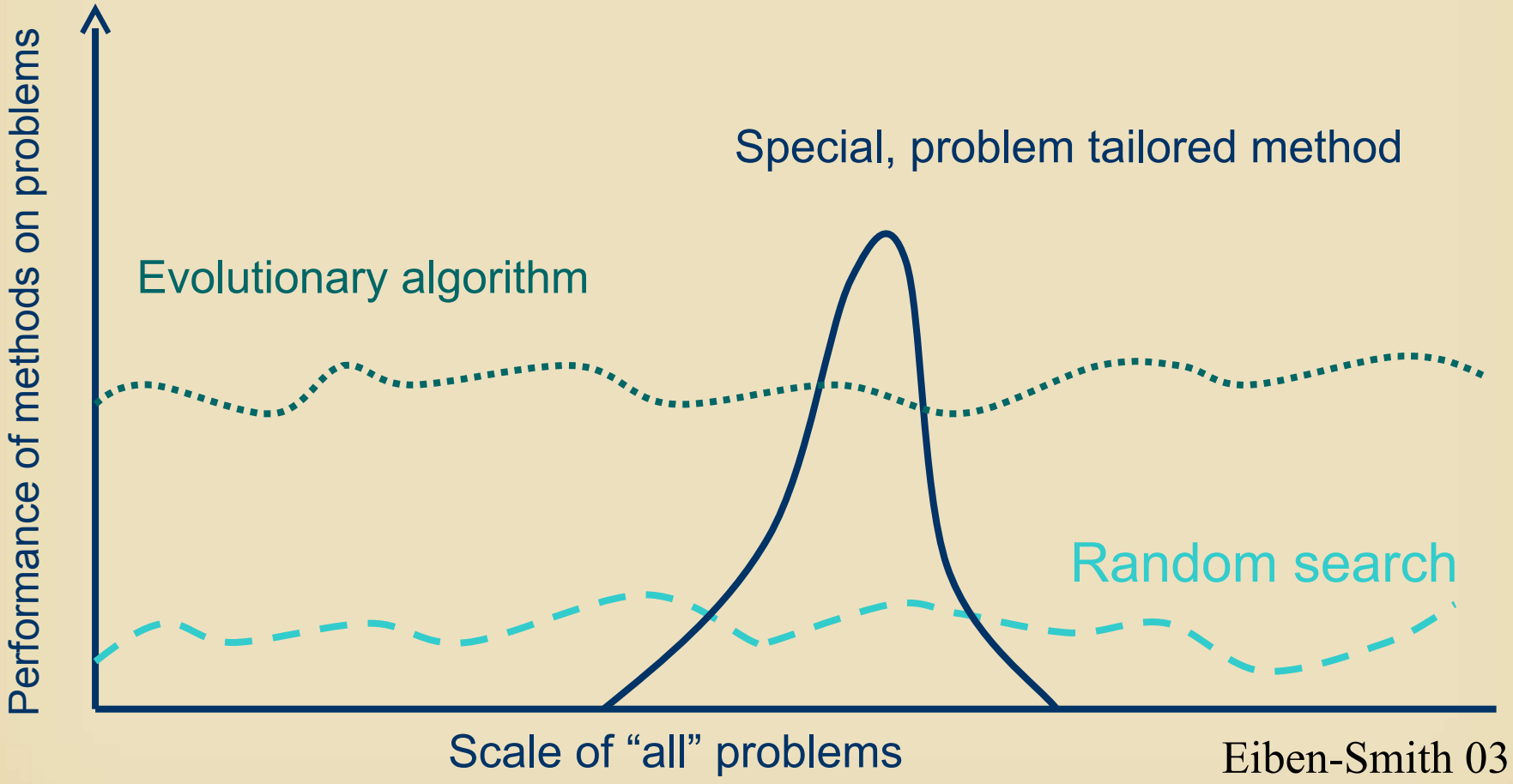
Name	*Function definition*	*Dimensions*
F9	$f_i = \sum_{j=2}^{n} (j-1)x_j t_i^{(j-2)} - \sum_{j=1}^{n} (x_j t_i^{(j-1)})^2 - 1$ where $t_2 = i/29, 1 \le i \le 29$	$2 \le n \le 31, m = 31$
	$f_{30}(x) = x_1,\ f_{31}(x) = x_2 - x_1^2 - 1$	
	Standard starting point : $x_0 - (0, \ldots, 0)$	
F10	$f_{2i} - 1(x) = 10(x_{2i} - x_{(i-1)}^2)$	n variable but even $m = n$
	$f_{2i}(x) = 1 - x_{(2i-1)}$	
	Standard starting point: $x_0 = (\xi_j)$ *where*	
	$\xi_{2j} - 1 = -1.2, \xi_{2j} = 1$	
F11	$f_1(x) = x_1 - 0.2$	n variable, $m = 2n$
	$f_i(x) = \alpha^{[1/2]} \left(\exp\left[\frac{x_i}{10} \right] + \exp\left[\frac{x_{[i-1]}}{10} \right] - y_i \right), 2 \le i \le n$	
	$f_i(x) = \alpha^{[1/2]} \left(\exp\left[\frac{x_{[i-n+1]}}{10} \right] - \exp\left[\frac{-1}{10} \right] \right), n \le i \le 2n$	
	$f_{2n}(x) = \left(\sum_{j=1}^{n} (n-j+1)x_i^2 \right) - 1$ where	
	$\alpha = 10^5, y_i = \exp\left[\frac{i}{10} + \exp\left[\frac{(i-1)}{10} \right] \right.$	
	Standard starting point: $x_0 = (\frac{1}{2}, \ldots, \frac{1}{2})$	
F12	$f_1(x) = 10^4 x_1 x_2 - 1$	$n = 2, m = 2$
	$f_2(x) = \exp[-x_1] + \exp[-x_2] - 1.0001$	
	Standard starting point: $x_0 = (0, 1)$	
F13	$f_i(x) = \exp\left[\frac{-\lvert y_i mi\ x_2 \rvert^{(x_3)}}{x_i} \right] - t_i$	$n = 3, n \le m \le 100$
	$t_i = i/100$	
	$y_i = 25 + (-50\ln(t_i))^{(2/3)}$	
	Standard starting point: $x_0 = (5, 2.5, 0.15)$	
F14	$f_{(4i-3)}(x) = x_{(4i-3)} + 10x_{(4i-2)}$	n variable but a multiple of 4
	$f_{(4i-3)}(x) = 5^{(1/2)}(x_{(4i-1)} - x_{4i})$	$m = n$
	$f_{(4i-1)}(x) = (x_{(4i-2)} - 2x_{(4i-1)})^2$	
	$f_{(4i)}(x) = 10^{(1/2)}(x_{(4i-3)} - x_{(4i)})^2$	
	Standard starting point: $x_0 = (\xi_i)$	
	$\xi_{4j} - 3 = 3, \xi_{4j} - 2 = -1, \xi_{4j} - 1 = 0, \xi_{4j} = 1$	

- Typical considerations:
  - Big populations and high mutation rates do not improve the algorithm...
  - … Neither do it small populations and mutation rates.
  - Big populations aren't necessarily needed.
  - Crossover disruptivity does not appear to play an essential role.
  - Crossover is less important with big populations.

- An example of a evolution strategy:
  - Typical for numerical optimization: Find the minimum (maximum) of a function.
  - Algorithm 1:
    - A point in space (vector) is a chromosome: $(x_1,x_2...x_n)$
    - A random point is chosen (population size=1)
    - For each of the vector components, a gaussian random number with varianza σ is added to the component.
    - If the value of the function in the modified vector is less than the previous one, use the new one and repeat the process from this new point. If not, use the old one and repeat.

- Algorithm 2:
  - The chromosome is composed by the vector and an associated (to each component) varianza value $(x_1,x_2...x_n, \sigma_1,\sigma_2,...\sigma_n)$
  - The mutation parameters (varianzas) coevolve with the solution.
    - First, the varianza is mutated: The coordinate corresponding to the mutation with the mutated varianza is obtained.
  - The rest of the algorithm is equal to Algorithm 1.

The expected performance of an EA:



- Typical evolution over time (generations) of an EA.

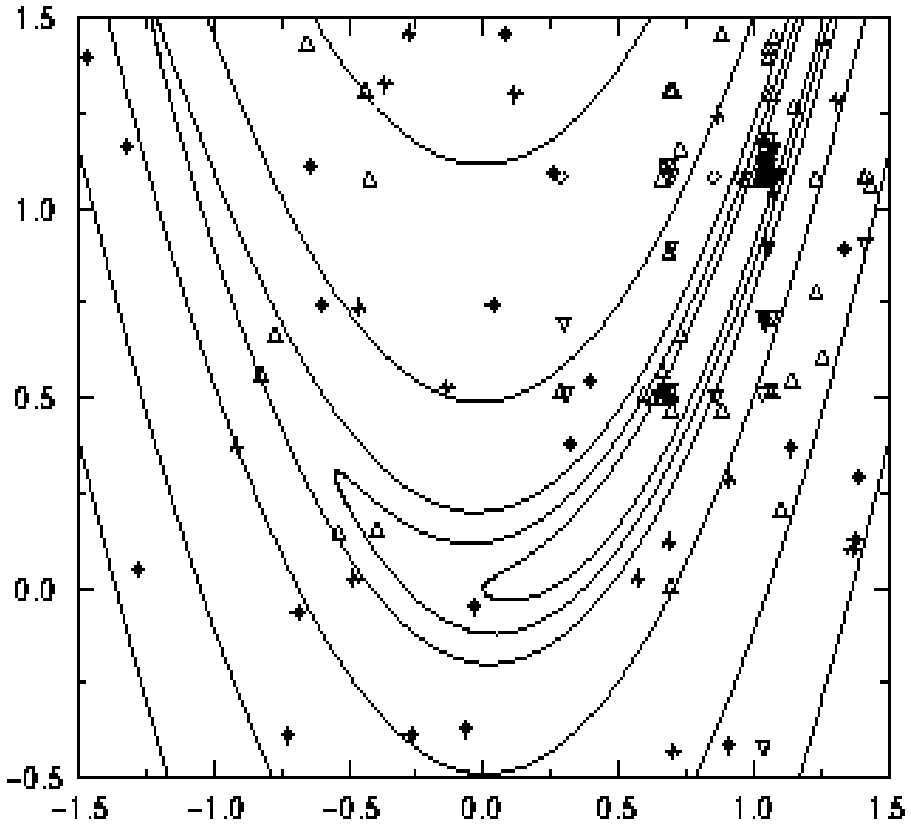
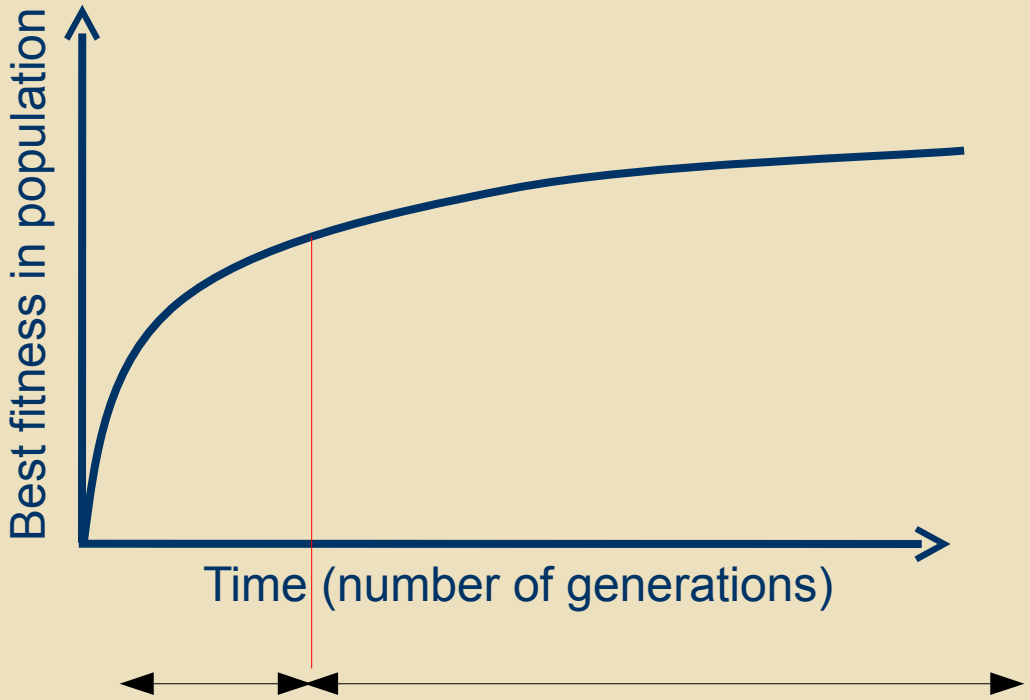


Figure 21: Minimization of two-dimensional Rosenbrock function by a genetic algorithm---population distributions of the first, tenth, twentieth, and thirtieth generations.

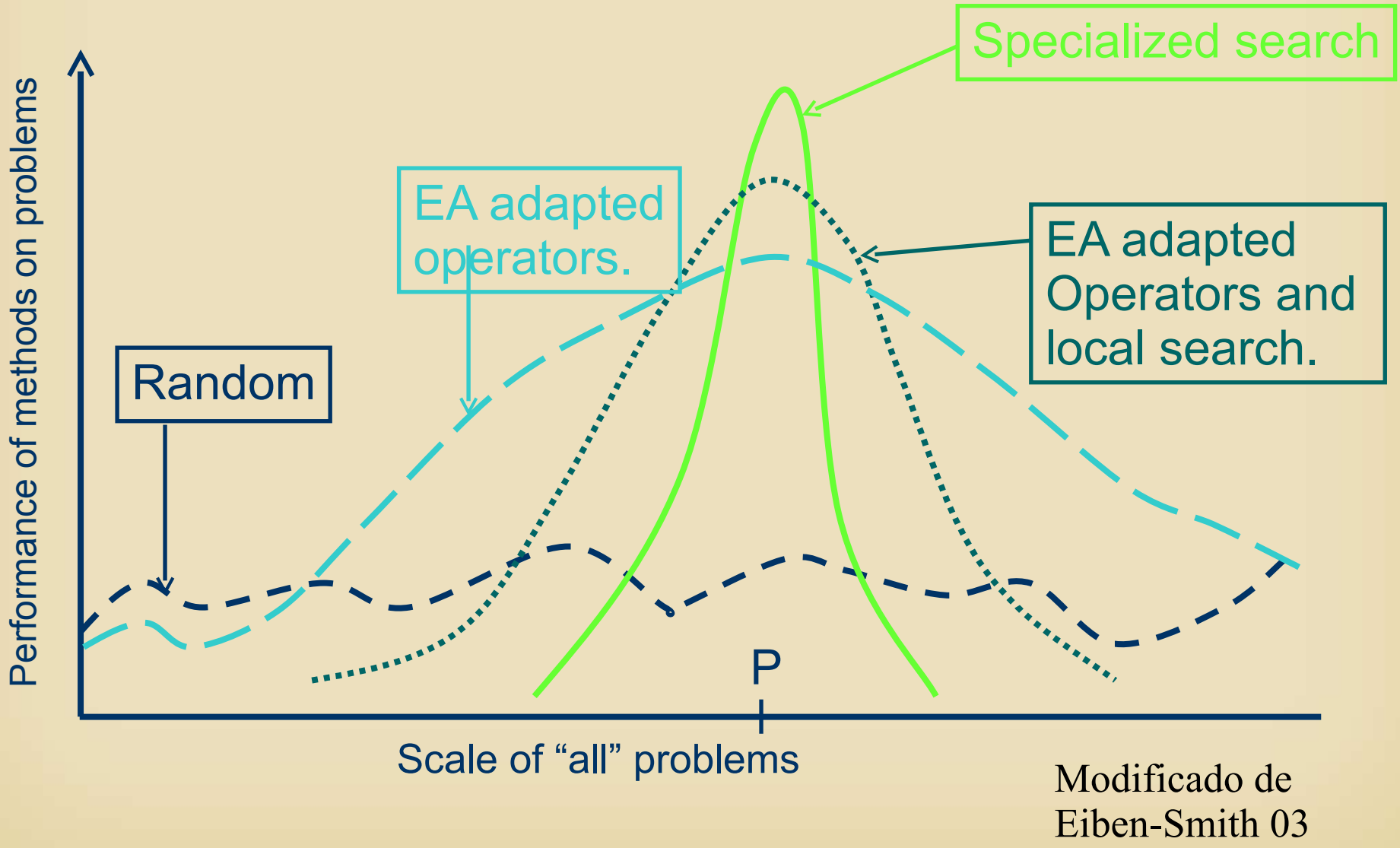
Computational Science Education Project
http://www.phy.ornl.gov/csep/

- Typical evolution over time (generations) of an EA.



After a fast increase in fitness, many generations are needed to increase it slightly (are sophisticated initializations really needed?)

- Adapted EAs.



Modificado de Eiben-Smith 03

- Un GA adaptado para un problema difícil.

VOLUME 75, NUMBER 2 PHYSICAL REVIEW LETTERS 10 JULY 1995

Molecular Geometry Optimization with a Genetic Algorithm

D. M. Deaven and K. M. Ho

Physics Department, Ames Laboratory U.S. DOE, Ames, Iowa 50011

(Received 19 January 1995)

We present a method for reliably determining the lowest energy structure of an atomic cluster in an arbitrary model potential. The method is based on a genetic algorithm, which operates on a population of candidate structures to produce new candidates with lower energies. Our method dramatically outperforms simulated annealing, which we demonstrate by applying the genetic algorithm to a tight-binding model potential for carbon. With this potential, the algorithm efficiently finds fullerene cluster structures up to C_{60} starting from random atomic coordinates.

- Codificación:
  - Un vector con las coordenadas de cada átomo del cluster (números reales)
- Cruce:
  - Se elige aleatoriamente un plano que pase por el centro de masas y se divide en dos conjuntos el cluster.
  - De dos padres así divididos se obtiene un hijo ensamblando la parte que cae por encima del plano de uno de los padres con la parte que cae por debajo de la del otro.
    - Si el número de átomos resultante no es correcto para formar el cluster del que se quiere determinar la estructura, se mueven los planos de corte en la dirección normal a estos hacia arriba en un padre y hacia abajo en el otro hasta que el hijo tenga el número correcto.
  - El hijo resultante es relajado usando un método de gradiente conjugado hasta el mínimo local más cercano.

- Mutación:
  - Mutación 1:
    - Mueve los átomos aleatoriamente en una dirección aleatoria un número de veces aleatorio entre 5 y 50. Tiene en cuenta casos no factibles (e.g.: átomos demasiado juntos)
  - Mutación 2:
    - Busca una cuenca próxima en direcciones que pueden ir „cuesta arriba“. El resultado puede tener una energía mayor (ser peor) pero añadir „genes“ inexistentes.
      - Después de la mutación se realiza una minimización local que puede conceder una mejor aptitud al hijo.
- Selección:
  - De Boltzmann, con una temperatura dada por el promedio de los átomos que forman el cluster.

- El algoritmo:
  - Los padres son elegidos continuamente de la población (que puede ser de un tamaño tan pequeño como de cuatro elementos) con una probabilidad dada por la distribución de Boltzmann.
  - De cada dos padres se obtiene un hijo que reemplaza a uno de los padres en la población si su energía es menor (mayor aptitud)
  - Con una probabilidad μ se muta el hijo (ej.: μ=0.05) que sufre también una minimización local (GC) y reemplaza a algún elemento de la población si su aptitud es mayor.
  - La aptitud se asume „escalonada“: No se consideran significativas diferencias de aptitud por debajo de un determinado umbral y, por tanto, no hay reemplazo en estos casos.
  - El algoritmo ha resultado muy resistente frente a cambios en el tamaño de la población, temperatura para la selección, minimizador local, etc.

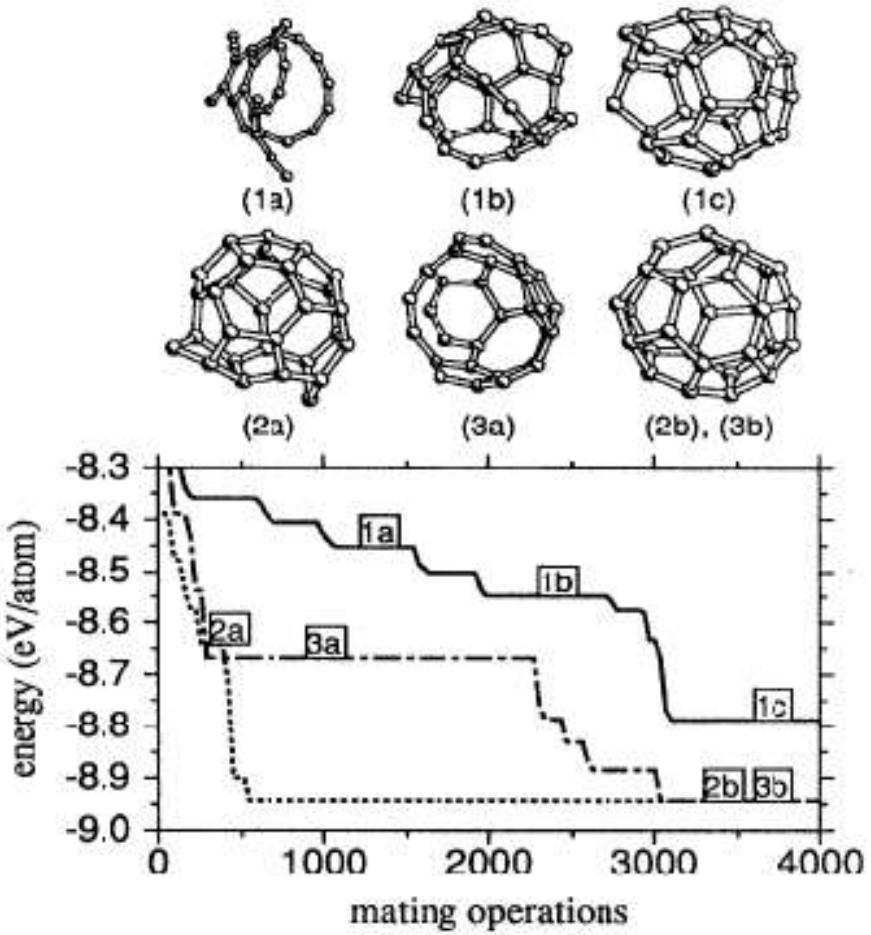
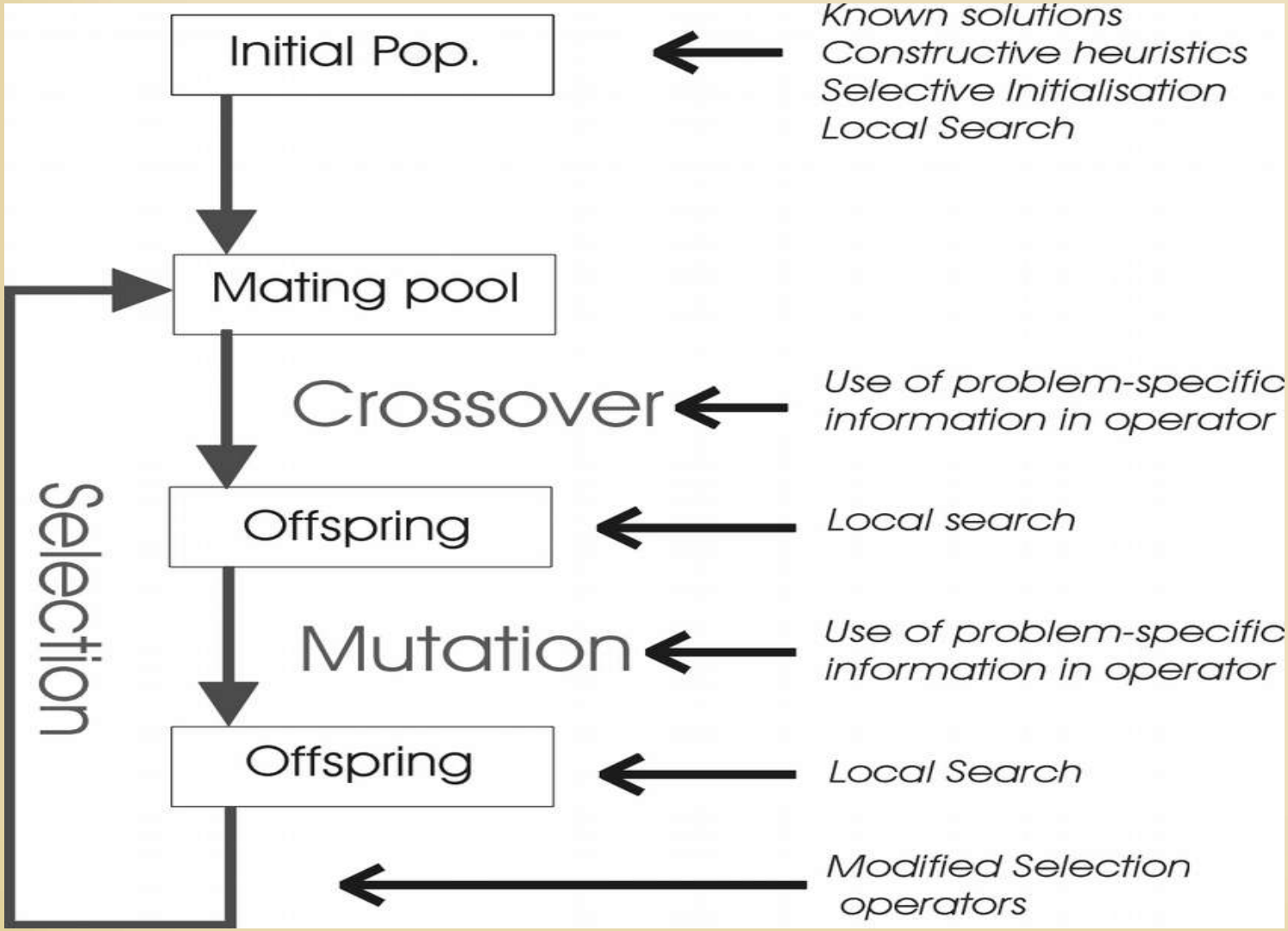


FIG. 3. Running the genetic algorithm on C_{30}. The solid line shows the lowest energy structure when the algorithm is run with no mutation ($\mu = 0$) for an ecology that failed to find the minimum energy configuration (a fullerene cage) within 4000 genetic operations. The structures (1a)–(1c) are present in the population at the times indicated. The structure (1c) resulting after 4000 genetic operations is a cage, and is eventually reduced to the perfect fullerene cage even with $\mu = 0$. The broken lines illustrate two $\mu = 0.05$ ecologies which arrive at the perfect cage (2b) via distinct routes (2a), (3a).

- Este es un ejemplo de un algoritmo híbrido: Incluyen búsquedas locales y/o conocimiento específico del problema.



Eiben-Smith 03

Cruce específico

Paso de GC

Mutación específica

Paso de GC

Selección de Boltz-mann, aunque de discutible aportación

- Algo de teoría:
  - Esquemas (schemata):
    - El esquema (schema): (1*1*010) asimila los cromosomas {(1111010),(1110010),(1011010),(1010010)}
    - Un esquema con *r* símbolos „*“ asimila 2^r cromosomas.
    - Un cromosoma de longitud m, es asimilado por 2^m esquemas distintos.
    - En un esquema, el número de posiciones distintas de „*“ define el orden del mismo.
      - (1*1*010) es de orden 5.
    - La longitud de definición es el número de posiciones más uno que están entre la primera y última posición distinta de „*“
      - (1*1*010) tiene longitud de definición 6.
    - Se define la aptitud de un esquema como el promedio de las aptitudes de todos los cromosomas que son asimilados por dicho esquema.

- Un GA procesa como poco 2^m esquemas y como mucho N 2^m (N=tamaño población, m= longitud cromosoma).
  - Algunos de ellos son procesados de manera que son muestreados de manera exponencialmente creciente.
  - Holland mostró que al menos N^3 son muestreados exponencialmente: paralelismo implícito.
  - Los operadores de cruce y mutación establecen una manera ordenada pero aleatoria de (inter)cambio de información entre los cromosomas. Aunque su efecto disruptivo sobre los esquemas es controlado si los esquemas son cortos y de orden bajo.
- Teorema de Esquemas: Los esquemas cortos, de orden bajo y aptitud por encima de la media reciben un muestreo exponencialmente creciente con el paso de las generaciones de un GA.
- Hipótesis de los bloques constructores: Un GA busca el rendimiento óptimo a través de la yuxtaposición de esquemas cortos de orden bajo y elevada aptitud (los llamados bloques constructores)

- Paralelización de GAs:
  - Manteniendo una única población con interacción completa:
    - Adecuado para un modo de trabajo Master/worker.
    - Muchas comunicaciones: El master tienen que reunir a toda la población en cada generación.
    - Aptitud evaluada en paralelo sin problemas.
    - Conceptualmente es idéntico al secuencial, de modo que es aplicable el mismo cuerpo de conocimiento.
  - Manteniendo una única población con interacción limitada:
    - La limitación se impone para mantener las comunicaciones bajo control: Solo se permite interaccionar entre los vecinos próximos de la población o dentro de las subpoblaciones asignadas a cada procesador
  - GA de grano grueso:
    - Subconjuntos de la población total son distribuidos entre los procesadores. Cada subpoblación es evaluada de manera completamente independiente. Se produce mezcla entre las subpoblaciones solo mediante intercambio de los mejores individuos entre subconjuntos de procesadores (e.g.: en una anillo)

- Artificial Immune Systems.
  - Simulate the pattern matching mechanisms used by the immune system to solve problems.
  - Use the immune system model has desirable characteristics:
    - Is a diverse model, hence robust: a individual is immune to different pathogens and different individuals have a different immune response.
    - It is distributed: no central control is needed. It allows the distribution of load and is more robust against component failure.
    - Error tolerant: A few errors are not catastrophic.
    - Dynamical: System components are continuosly updated and destroyed, increasing the possibility of being more diverse, hence, more adapted. At the same time it keeps a memory that improves its response.
    - Self-protected: Its same working mechanism protects it.

- Basic ideas of a immune system (from our own perspective):
  - Immune system discriminates between self/nonself signals.
  - Antibodies (Abs) mature :
    - Negative selection: The cells responsible for the immune response mature in isolated environments where they are exposed to self stimuli. If they react to self stimuli, they are killed.
    - Positive selection: at the same time, they must react against non-self stimuli (antigen, Ags).
  - Mature detectors that are activated, multiply and improve.

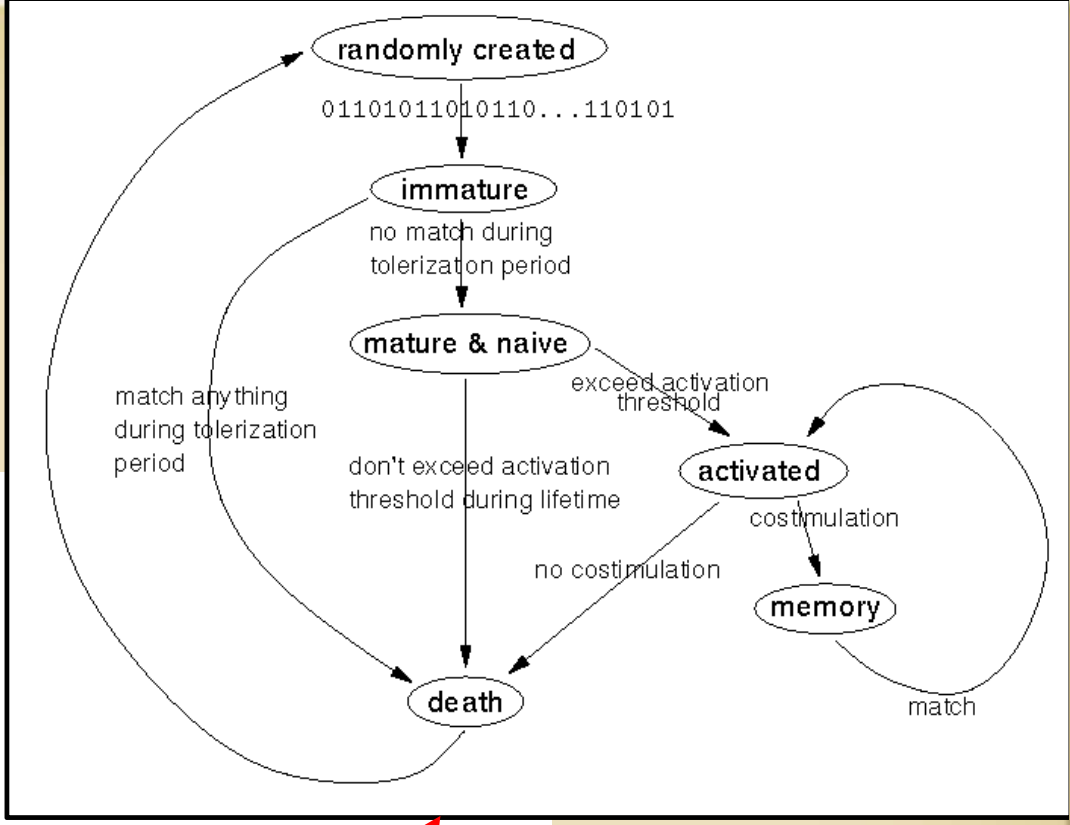
- Typical Algorithm:
  - Generate detectors (Abs) that do not react (recognise) to self patterns but do react (recognise), at least a part, of the non-self patterns.
    - How to represent and code the detectors: Abs?
    - How to represent the self? (hence, the non self)
    - Implement „recognition“ operation.
  - Use the detectors to monitor the „target“ patterns.
  - When a detector is activated, a response is produced and the information used to improve the detector response (and of the detector network in general)
    - How to dynamically refine the immune system response?

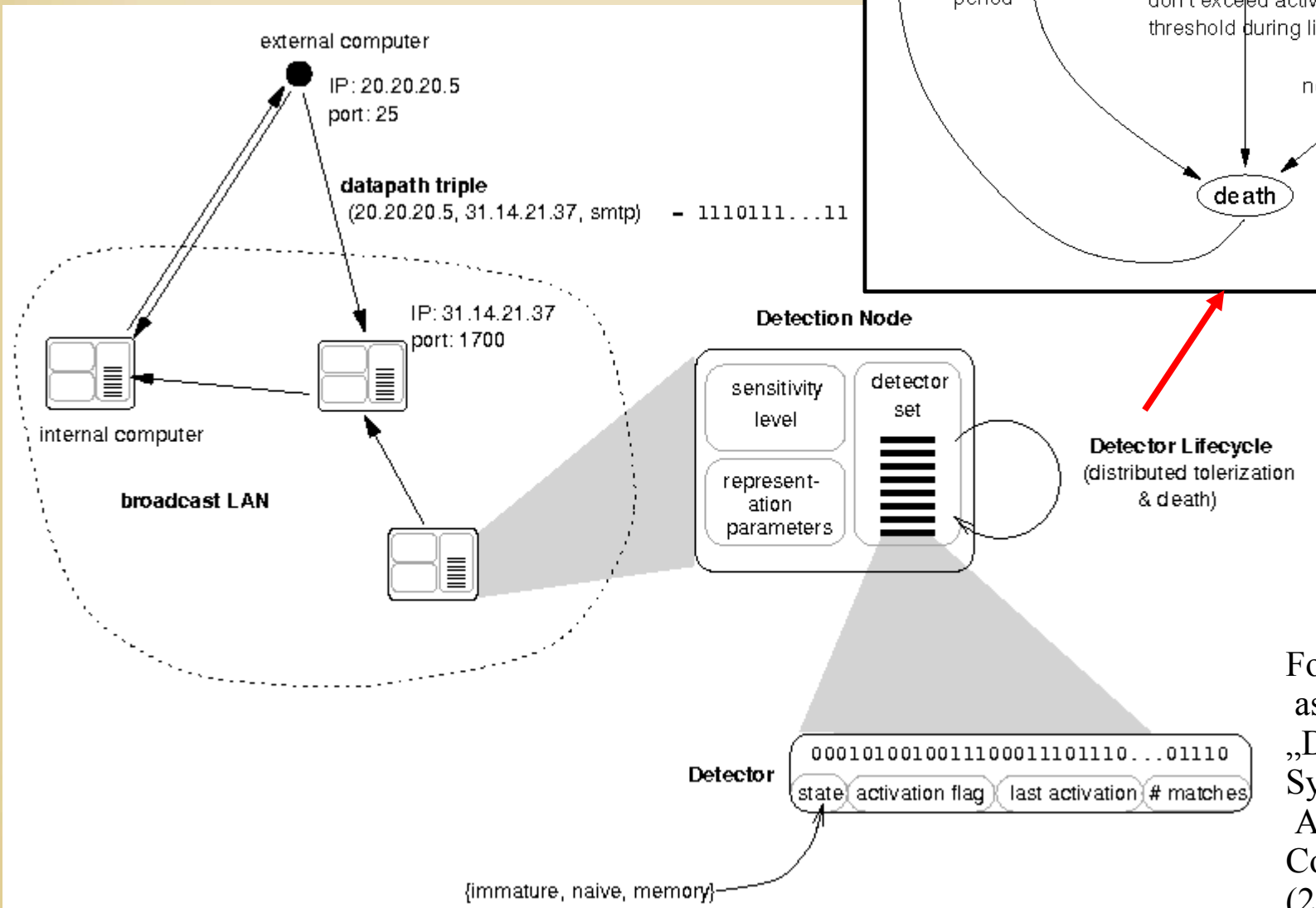
- Examples of the basic concepts:
  - If the important patterns for the process can be represented as a string of symbols, the self can be implemented as a set of equal length strings.
  - A detector (Ab) can be implemented as a string of the same length than one of the chains of the set that represents the self.
  - A „positive“ is a partial match of enough characters among the target chain (Ag) and that of the detector (Ab).
  - The initial set of detectors can be randomly built.
    - Detector „mature“ matching with the „self“ and elliminating those that produce a „positive“.
    - The initial set is refined through the correct matching of the Ags.

- Example: Protecting a LAN with an immune algorithm.
  - Self: Typical connetions point to point (Level 2, transport) among computers, including those belonging to a external network. A self element is the triplet (origin IP, destination IP, port) stored as a 49 bits string corresponding to a connection identified as „normal“. A nonself (Ag) element will be one of the connections that are rarely observed.
  - An Ab is a similar string plus an small amount of storage to indicate its state (e.g.: if it has been recently activated, it it is immature, etc.)
  - A detection happens when the 49 bits piece of the Ab representing the connection matches the 49 bits string of the Ag according to a given matching rule (e.g.: if there are *r* contiguos bits that are equal)

- The Abs are grouped in sets, one set per computer in the network.
  - Asuming that we are in an Ethernet network , all of the computers hear all of the communications among their members, hence every computer can update its own definition of self independently.
  - Each computer produces its own Abs randomly and expose them to its own self to reach its maturity.
  - Each mature Ab can act independently of any other Ab's. If one of them has recognized an Ag a certain number of times (there is a tolerance to elliminate false positives) a flag is raised.
  - If a mature Ab does not recognize anything during some time, it is deleted.
  - If a mature Ab has recognized a certain number of Ags, it enters a competition phase with others Abs in the same state to reach the state of a long lived „memory“ Ab with a low activation threshold that can react quickly to a problem in the network.
  - If several mature Abs recognize the same Ag, a competition among them is done and a fitness function is used to improve its response.

Architecture of the artificial immune system.

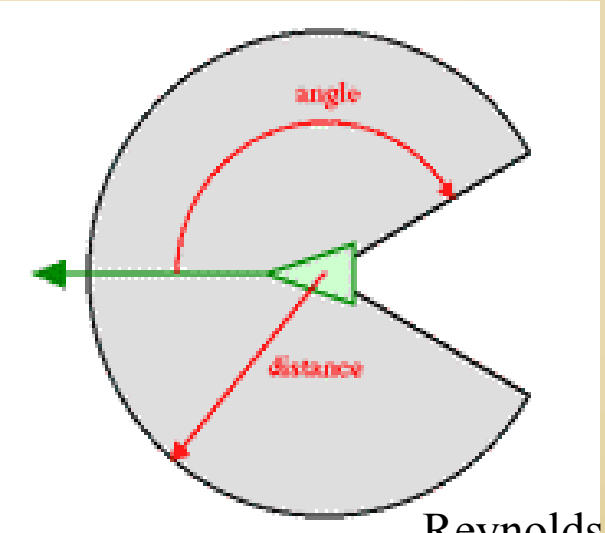




Forrest, Hofmeyr. „Immunology as Information Processing“ en „Design Principles for Immune Systems & Other Distributed Autonomous Systems“. Segel, Cohen, eds. Oxford U. Press (2000)

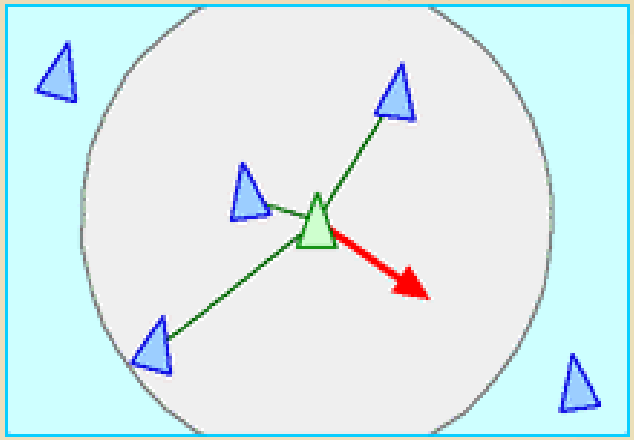
Swarms: Reynolds' Boids

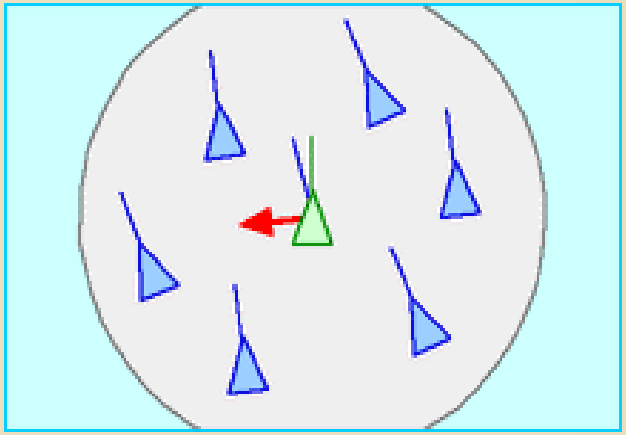
- How to simulate the behaviour of a flock of birds?
- Reynolds called „boid“ to each element.
  - A boid will behave in the next step depending on:
    - Its own velocity.
    - The speed and distance of those individuals that it can perceive. The perceptual system of a boid is defined by an angle and a distace.

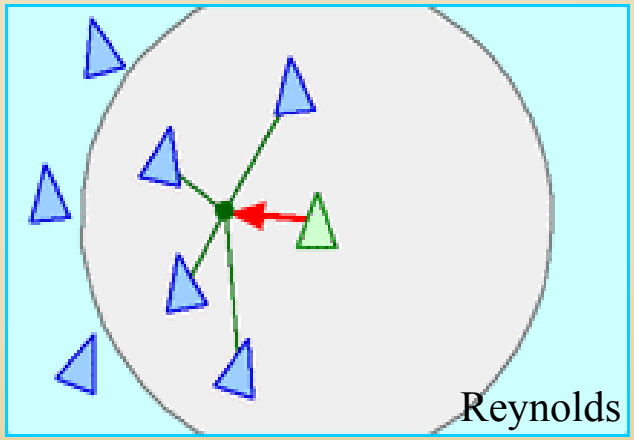


Reynolds

- In each step of the algorithm, the boid obeys the following rules:
  - Separation: Do not get too close to the neighbours.
  - Alignment: Adjust its speed to the average speed of the neighbours.
  - Cohesion: Adjust the speed to get closer to the neighbours' centroid.







- The precise mixture of the three rules and appropiate values for distances and angles requires some experimentation to get realistic movements.
- A modification of these ideas can be used to find the optimum.

Particle Swarm Optimization.

- A set (swarm) of particles searchs for the optimum of a function.
  - Each particle explores its surrounds, to do so, a speed is assigned to it.
  - Each particle stores the parameters (where it was in phase space) associated to its bet result.
  - All the particles know the best results of its neighbours.
  - At each step, every particle changes its speed by adding:
    - A random amount (usually weighed) in the direction of its own best result.
    - A random amount (usually weighed) in the direction of its neighbours best result.

Particle Swarm Optimization Pseudocode.

http://www.swarmintelligence.org/

```
For each particle
    Initialize particle
END
Do
 For each particle
        * Calculate fitness value
        *  If the fitness value is better than the best fitness value (pBest) in history
           set current value as the new pBest
    End
    * Choose the particle with the best fitness value of all the particles as the gBest
    For each particle
       *  Calculate particle velocity v[] = v[] + c1 * rand() * (pbest[] - present[]) +
                                            c2 *  rand() * (gbest[] - present[])
        * Update particle position according present[] = present[] + v[]
    End
```

(Pbest= particle best; gbest= global (neighbourhood) best)

- Parameters:
  - Number of Particles: Typically 10-50.
    - Relative weight of local (c1 value) and global (c2) minima.
  - If the speed is in excess of a prestablished maximum, it is limited (Vmax is another extra parameter. If too small, convergence is slow. If too big, the method becomes unstable).

Foraging Algorithms.

- Ant optimization.

Stigmergy is: indirect communication via interaction with the environment.

  - Optimization is performed by a set of individuals (ants) that communicate leaving messages in the visited places.
  - Ants leave a pheromone trace wherever they go.
    - If many ants have travelled the path, the pheromone trace is more intense.
    - Pheromones evaporate with time.

- E.g.: Travelling salesman problem:
  - An ant starts in a given city and travels all the nodes.
    - Its path is the solution represented by that particular ant.
  - Depending on the quality of the solution, the path represented by that ant receives more or less pheromones.
  - Pheromones evaporate with time.
  - At each node, the ant decides which node will be next depending on the amount of pheromones in the diferent available paths and a heuristic (in the TSP is usually the inverse of the distance between the node and the city to which the path goes to)

Monte Carlo Methods

Monte Carlo methods allow:

1) Generate samples from a given probability distribution $P(x)$ (usually called target density).

2) Estimate the expected value of a function under a given probability distribution :

$$\Phi = \langle \phi(\boldsymbol{x}) \rangle \equiv \int d^N \boldsymbol{x}\, P(\boldsymbol{x}) \phi(\boldsymbol{x})$$

Obviously, useful when $P(x)$ is complicated enough such that these expectation functions cannot be directly evaluated.

Why is Monte Carlo so popular?

- Monte Carlo Methods are almost the default choice for high dimensional problems:
  - The accuracy with which an average such as the previous formula is calculated, depends only on the variance of Φ and not on the dimension of the space. The variance of the average Φ goes as σ^2/R, where R is the size of the sample.
  - The average is calculated using a sample $\{\boldsymbol{x}^r\}_{r=1}^{R}$ obtained from the probability distribution $P(x)$ as:

$$\Phi = \frac{1}{R}\sum_{r=1}^{R} \phi(\boldsymbol{x}^r)$$

The problems of Monte Carlo:

- How to sample from $P(x)$?
  - Correct samples from $P(x)$ should come from places where $P(x)$ is big. How do we know which places are this without actually sampling $P(x)$ at every point?
    - If we choose to sample in a regular grid, the number of sample points grow as the power of the dimension of the space: If we use a basic size of 50 sample points in each dimension and have a space of dimension 100, 50^{100} points have to be sampled... more than the total number of atoms in the universe.
    - And systems of such dimensions are quite common...

Example: Calculate the average concentration of plankton in a lake assuming that the concentration at a point (x,y) is $\varphi(x,y)$ and that the depth of the lake is $P(x,y)$.

The required value is: $$\langle \phi(x,y) \rangle \equiv \frac{1}{Z} \int dx\, dy\, P(x,y)\, \phi(x,y)$$

with

$$Z = \int dx\, dy\, P(x,y)$$

- How to sample from $P(x,y)$?
  - If we don't know nothing about the depth of the lake, we have to sample at every possible (x,y) [within a given spatial resolution]... we have to discover every submerged valleys even in order to estimate Z.

Sampling techniques:

- Uniform sampling.
- Importance sampling.
- Rejection sampling.
- Metropolis-Hastings: Markov Chain Monte Carlo MCMC

Uniform Sampling:

- Let us try to solve the problem 2): estimate the expectation value of $\phi(x)$
- Since we cannot visit all possible points in space state, lets do it in the simplest way by drawing random samples uniformly and evaluating P($\boldsymbol{x}$) in those points:
    - The normalizing constant could be approximated by:

$$Z_R = \sum_{r=1}^{R} P(\boldsymbol{x}^{(r)})$$

$$\Phi = \langle \phi(\boldsymbol{x}) \rangle \equiv \int d^N \boldsymbol{x} \, P(\boldsymbol{x}) \phi(\boldsymbol{x})$$

- And then the estimate of would be calculated as:

$$\hat{\Phi} = \sum_{r=1}^{R} \phi(\boldsymbol{x}^{(r)}) \frac{P(\boldsymbol{x}^{(r)})}{Z_R}$$

Uniform Sampling:

- Is this of any help?
  - Only if the samples hit the "typical set", where the $\boldsymbol{x}$ corresponding to the highest valued $P(\boldsymbol{x})$ lie, the estimate is good.
    - For highly dimensional $P(\boldsymbol{x})$, the probability of drawing x from the typical set is very low... unless $P(\boldsymbol{x})$ is uniform itself.
    - Uniform sampling, hence it is not likely to be very useful.

Importance Sampling:

- Lets assume that we are able to ***evaluate*** the target density $P(\boldsymbol{x})$, at least to within a multiplicative constant Z, i.e.: we can evaluate $P^*(\boldsymbol{x})$ such that:

$$P(x) = P^*(x)/Z$$

- But we cannot sample from it. Instead we can sample from a simpler *sampler density function* $Q(x)$ and that we can evaluate it up to a constant, i.e: we can evaluate $Q^*(\boldsymbol{x})$ such that:

$$Q(x) = Q^*(x)/Z_Q$$

Importance Sampling:

- Then we generate samples $\{x^{(r)}\}_{r=1}^{R}$ L from $Q(\boldsymbol{x})$ Where they obtained from $P(\boldsymbol{x})$, then we could estimate $\boldsymbol{\Phi}$ directly as:

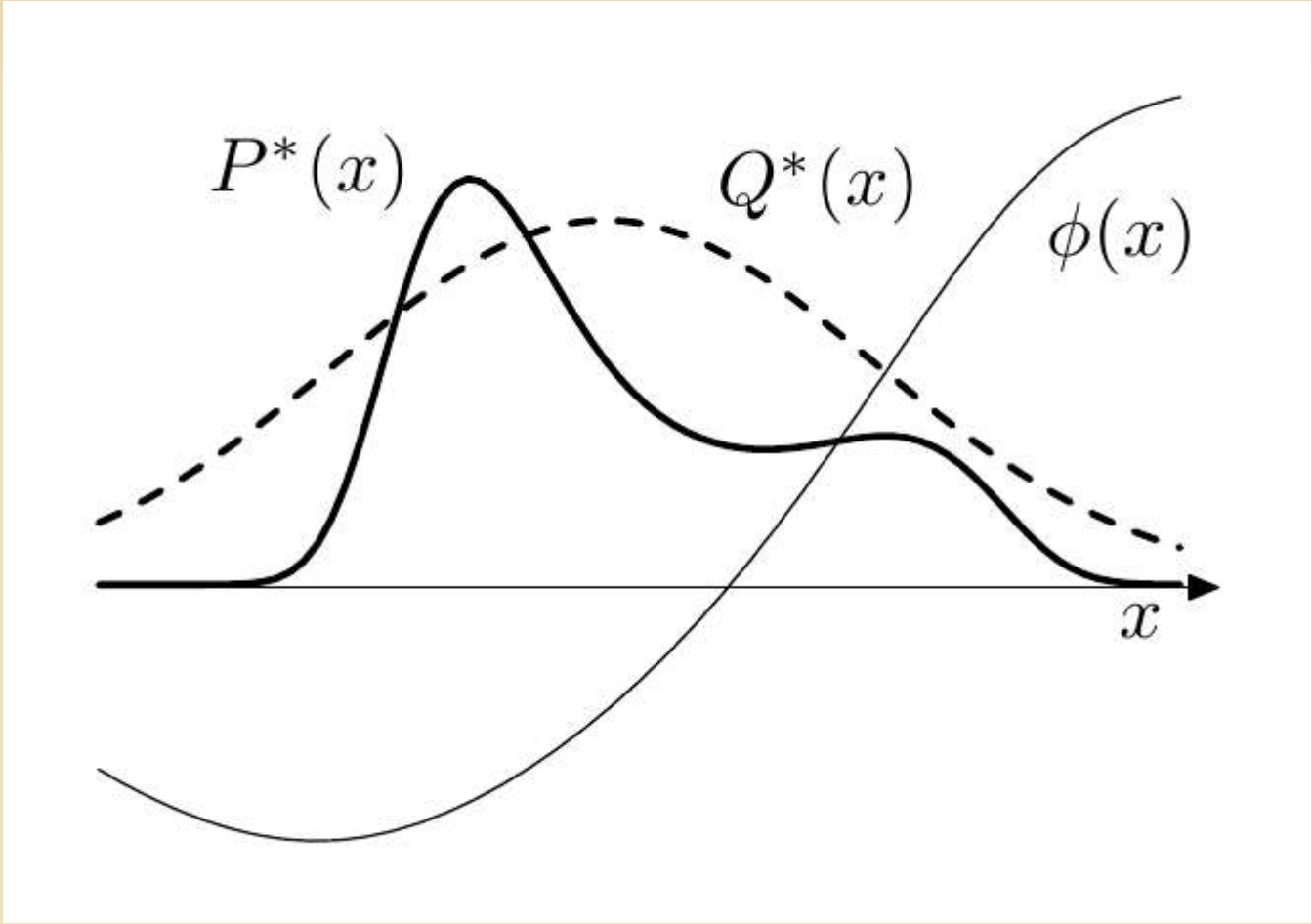
$$\Phi = \frac{1}{R}\sum_{r=1}^{R} \phi(\boldsymbol{x}^{\boldsymbol{r}})$$

- However, by sampling $Q(\boldsymbol{x})$ now we will have cases in which $Q(\boldsymbol{x}) > P(\boldsymbol{x})$ and those would be over weighed if we use directly the formula above. Cases in which $P(\boldsymbol{x}) > Q(\boldsymbol{x})$ would be under weighed. So, now we define the weights:

$$w_r = \frac{P^*(x^{(r)})}{Q^*(x^{(r)})}$$

- These weights adjust the importance of each sample and we can use as the estimate:

$$\hat{\Phi} = \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r}$$



D. Mackay.

Functions involved in importance sampling. The function for which we have to estimate the average $\varphi(x)$ over the probability function $P(\boldsymbol{x})$. The function $P^*(\boldsymbol{x}) = Z\ P(\boldsymbol{x})$ that we can evaluate and the function $Q^*(\boldsymbol{x}) = Z_Q\ Q(\boldsymbol{x})$, where $Q(\boldsymbol{x})$ is the sampler density function that we can Sample from.

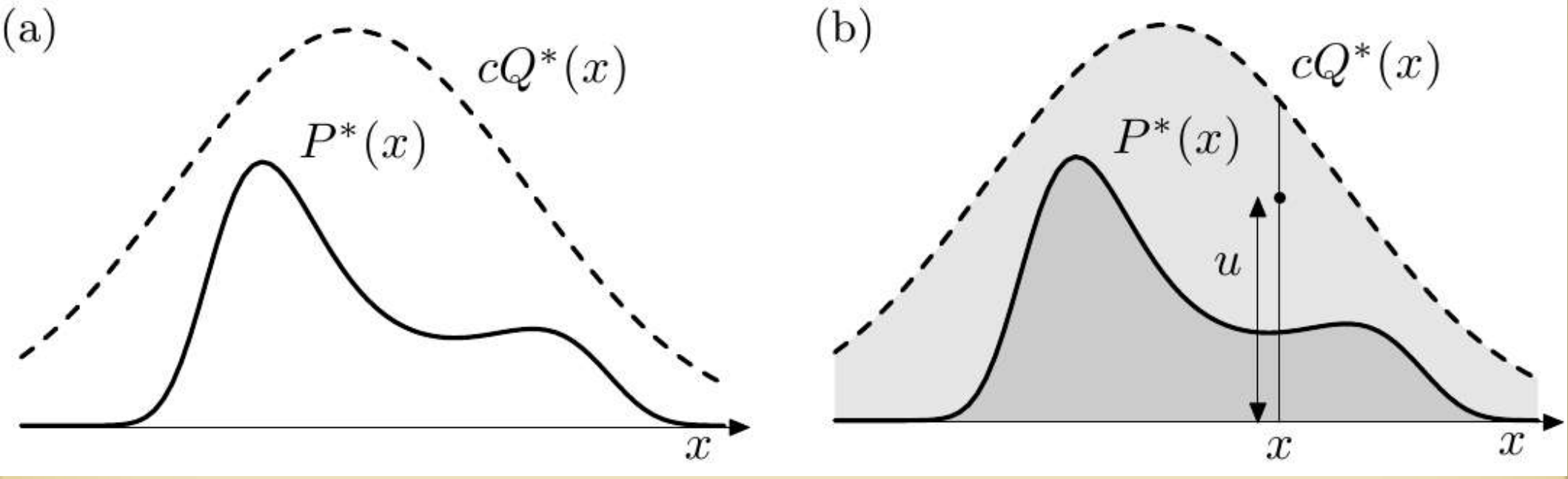
- In practise it is difficult to estimate the reliability of the result. The weighs are not necessarily a good guide.
- Unless $Q(x)$ is a good approximation of $P(x)$ most of the samples will have a negligible weight.

Rejection Sampling:

- Again, we assume that we have a difficult to sample $P(x)$ and a simpler $Q(x)$ that we can sample and evaluate up to the usual multiplicative constant Z_Q .
- We also assume that we know a constant c such that:

$$c\,Q^*(x) > P^*(x)\;\forall\; x$$

- Procedure:
  - Generate one random number, x, from $Q(x)$ and evaluate $cQ^*(x)$.
  - Generate another random number, u, but now in $[0, cQ^*(x)]$.
  - Evaluate $P^*(x)$. If $u > P^*(x)$, then reject x, accept otherwise.
    - Accept means to add x to $\{x^{(r)}\}$
    - u is discarded.
  - Continue till a big enough number of samples $\{x^{(r)}\}$ has been obtained.
    - In high dimensional problems it is likely that c will become large in order to make $cQ^*(x) > P^*(x)$ for all x and the frequency of rejection will be large unless $Q(x)$ is a good approximation to $P(x)$
    - The rejection ratio is proportional to the volume of $P(x)$ over $cQ(x)$. For normalized $P(x)$ and $Q(x)$, this is $1/c$. Hence, if c



D. Mackay

Functions involved in rejection sampling: $P^*(\boldsymbol{x})$ and $Q^*(\boldsymbol{x})$ are as in importance sampling. c is the constant that guarantees that $cQ^*(x) > P^*(x)$. x is the first random number generated from $Q(\boldsymbol{x})$ and u the second, uniform in $[0, cQ^*(x)]$. x is accepted or rejected depending on wheter is inside the dark grey region (accepted) or outside (rejected).

- **Metropolis-Hastings method (1).**

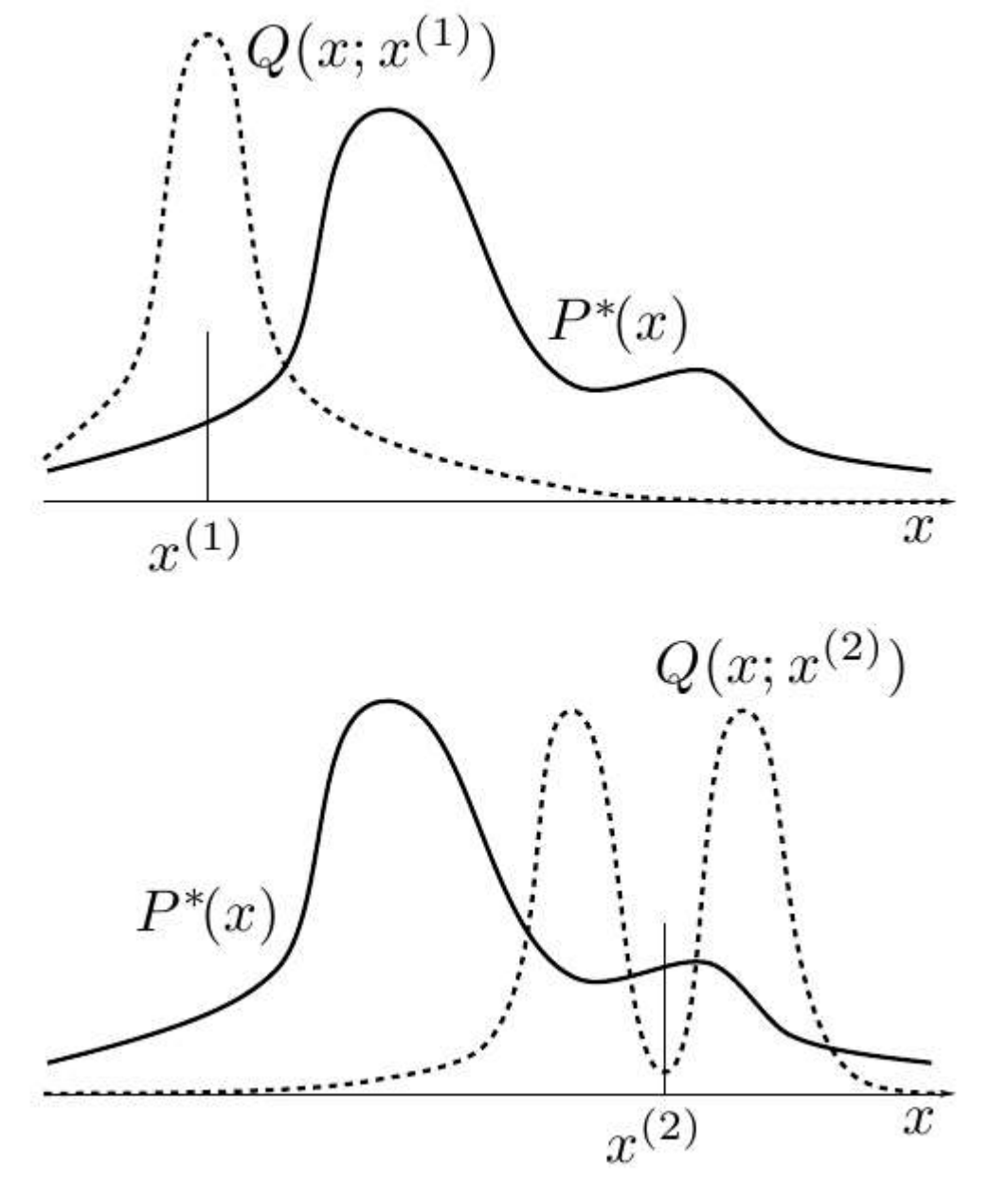
  Importance and rejection sampling work badly unless $Q(x)$ closely tracks $P(x)$. M-H adjusts $Q(x)$ at each step t : $Q(x) \equiv Q(x;x^{(t)})$

    - $Q(x;x^{(t)})$ could be, for example, a gaussian centered around $x^{(t)}$

  - As before, we assume that we can evaluate $P^*(\boldsymbol{x})$ for any $\boldsymbol{x}$. From $Q(x;x^{(t)})$ we obtain a new x.
  - The new x is accepted if $a \geq 1$, where

$$a = \frac{P^*(x)}{P^*(x^{(t)})} \times \frac{Q(x^{(t)};x)}{Q(x;x^{(t)})}$$

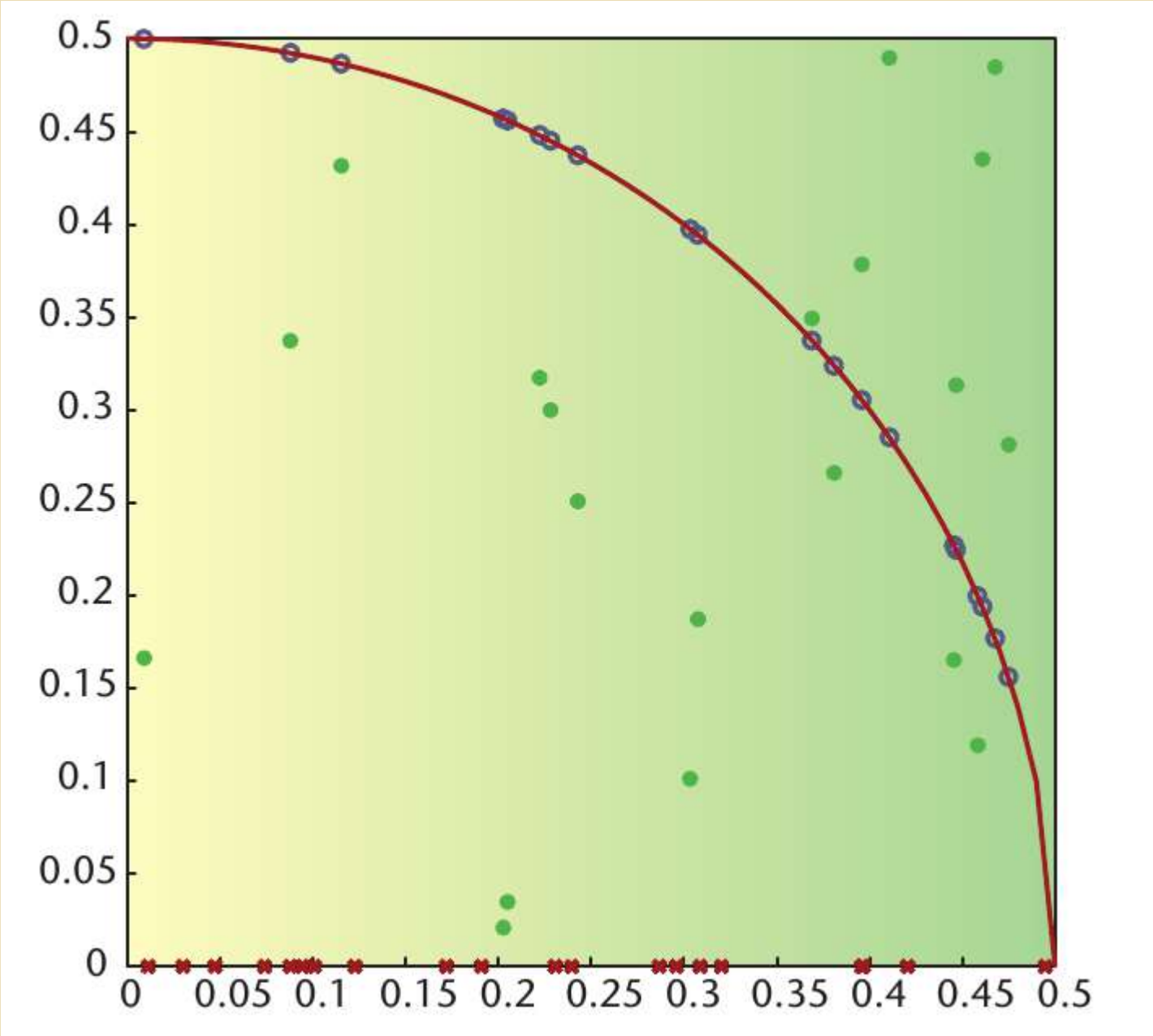
- **Metropolis-Hastings method (2).**
  - Otherwise, it is accepted with probability a
    - If it is accepted $x^{(t+1)} = x$
    - And if not, $x^{(t+1)} = x^{(t)}$
    - Note that, differently from rejection sampling, rejected points do influence on the list of samples since they are not discarded if rejected. The current state is rewritten in the list if rejected.
    - If Q is symmetric, the second term in the quotient is 1. This is what has been traditionally called a Metropolis method.



Functions involved in the Metropolis-Hasting method. Note how Q changes with the current state.

Mackay.

- For any $Q(x';x) > 0$, for all x,x', it can be demonstrated that the distribution of $x^{(t)}$ tends to $P(x)$ as $t \rightarrow \infty$
- M-H method is an example of a Markov Chain Monte Carlo (MCMC).
  - In rejection sampling the accepted points are independent samples of the desired distribution.
  - In MCMC a sequence of states is generated. Each state is dependent on the previous one, hence a MCMC must run for some time in order to generate samples that are effectively independent.
  - It is possible to greatly increase the efficiency of the MC methods described. See Hamiltonian MC, overrelaxation and Simulated annealing.


0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

A system based on ideas from physics: Simulated Annealing

- Use an algorithm that emulates how a physical system reaches the state of minimum energy, as a series of equilibrium states very close together.
  - E.g.: A melt solid cristalizes when cold enough. If cooling is done very slowly, a single crystal is obtained, that is: the minimum energy state..

– A system in thermal equilibrium at a temperature T has its energy distributed among all possible states with a Boltzmann probability distribution:

$$Prob(E) \sim e^{\frac{-E}{kT}}$$

- Hence, high energy states are possible, although with a very low probability, this happens even for systems with a very low temperature

- Algorithm:
  - Start from an approximate solution. Evaluate the objective function to be minimized in that point.
  - Propose a change to the original solution and evaluate the objective function in the new point.
    - If the value of the objective function in the new point is smaller, accept it and change it again.
    - If the value of the objective function is bigger, accept the new point with probability: $e^{\frac{-\left(E_{new}-E_{old}\right)}{T}}$

      for a given „temperature“ value, T, defined ad hoc.
  - Continue till convergence is achieved.

- The procedure is a Markov chain with transition probabilities given by the Boltzmann distribution.
- Boltzmann's distribution is not the only possible:

$$\textit{Threshold accepting}: P_{TA}(\Delta E) = \begin{cases} 1 & si \quad \Delta E \le T \\ 0 & si \quad \Delta E > T \end{cases}$$

$$\textit{Tsallis annealing}: P_{TS}(\Delta E, q) = \begin{cases} 1 & si \quad \Delta E \le T \\ \left[1-(1-q)\frac{\Delta E}{T}\right]^{1/(1-q)} & si \quad \Delta E > 0 \quad y \quad (1-q)\frac{\Delta E}{T} \le 1 \\ 0 & si \quad \Delta E > 0 \quad y \quad (1-q)\frac{\Delta E}{T} > 1 \end{cases}$$

In fact, for many problems, threshold accepting is the best strategy.

- The two main problems to solve when this algorithm is implemented is:
  - Find a good way to generate new states in which to apply the accept/reject test.
  - Find a good „cooling“ strategy: How to define adequately the „temperature“ and how to vary it during the calculation.

- Example: Travelling Salesman.
  - Configuration: The list of the pairs of coordinates (xi,yi) for each city. A configuration is a permutation of the original list (cities are visited in the order of the list)
    - Generation of new configuration: As an example, a section of the old path is travelled in inverse order, or a piece of the old path is deleted from its position and inserted in a new one obtained randomly.
  - Objective function: Total length of the path.

$$L = \sum \sqrt{\left(x_i - x_{i+1}\right)^2 + \left(y_i - y_{i+1}\right)^2}$$

  - Cooling strategy: Selected heuristically. As an example, randomly generated configurations are used to determine the typical range of variations in length path ΔL. Choose an initial T substantially higher than the biggest one previously found. Produce a predetermined number of configurations with that value and them decrease the T value by a 10%.

- The objective function can be adequately weighed to avoid undesirable configurations.
  - Eg.: If we are considering the total length of the paths joining a set of Integrated Circuits that could be in different boards, we could assing a value μ=1 to those in one of the boards and μ=-1 to those in the other. To reduce the communication among boards, a possible objective function could be:

$$L=\sum\left[\sqrt{\left(x_i-x_{i+1}\right)^2+\left(y_i-y_{i+1}\right)^2}+\lambda\left(\mu_i-\mu_{i+1}\right)^2\right]$$

Bibliography

- Z. Michalewitz. „Genetic Algorithms+Data Structures= Evolution Programs“. Springer (1996)
- A.E. Eiben and J.E. Smith, „Introduction to Evolutionary Computing“ Springer (2003) Material adicional en http://www.cems.uwe.ac.uk/~jsmith/ecbook/ecbook.html
- K.A. De Jong, „Evolutionary Computation: A unified aproach“. The MIT Press (2006)
- C.A. Coello. „Introducción a la Computación Evolutiva.“ http://delta.cs.cinvestav.mx/~ccoello/compevol/apuntes.pdf
- EVONET http://evonet.lri.fr/
- Evolutionary Computation Education Center http://evonet.lri.fr/CIRCUS2/node.php?node=1

Bibliography

- S. Kirpatrick, C.D. Gelatt, M.P. Vecchi. „Optimization by Simulated Annealing“. Science 220 (1983) p. 671
- L.N. De Castro, J.I. Timmis „Artificial immune systems as a novel soft computing paradigm“ Soft Computing 7 (2003) p. 526
- D. Mackay “Information Theory, Inference, and Learning Algorithms”. Cambridge University Press. Non-printable version available online at: http://www.inference.phy.cam.ac.uk/itprnn/book.html
- D. O'Leary “Multidimensional Integration: Partition and conquer” Computing in Science and Enginering Nov./Dec. 2004 p. 58
- S. Forrest, S.A. Hofmeyr. „Immunology as Information Processing“ en „Design Principles for Immune systems & Other Distributed Autonomous Systems“ L.A. Segel and I.R. Cohen, eds. Oxfort Univ. Press (2000)

Bibliography

- Marco Dorigo et al. (2008) „Particle Swarm Optimization“ Scholarpedia 3(11):1486
- W.T. Reeves. „Particle Systems -A technique for modeling a class of fuzzy objects“. ACM Transactions on Graphics, 2(2):91-108,1983
- Reynolds, C. W. (1987) “Flocks, Herds, and Schools: A Distributed Behavioral Model”, in *Computer Graphics*, **21**(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.
- Kennedy, J. and Eberhart, R. C. “Particle swarm optimization”. Proc. IEEE int'l conf. on neural networks Vol. IV, pp. 1942-1948. IEEE service center, Piscataway, NJ, 1995.
- M. Dorigo and T. Stutzle “Ant Colony Optimization” The MIT press, 2004.