Computational statistics

Chapter 2: Combinatorial optimization

Thierry Denœux
Université de technologie de Compiègne

August 2026

Overview

Overview

Combinatorial optimization

- In this chapter, we consider a function $g : \mathbf{\Theta} \to \mathbb{R}$, where $\mathbf{\Theta}$ is a finite set of N elements.
- We seek the maximum of $g(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. This is called combinatorial or discrete optimization.
- We consider problems in which $\mathbf{\Theta}$ has a very large cardinality and cannot be searched exhaustively.

Example: traveling salesman problem

- The salesman must visit each of p cities exactly once and return to his point of origin, using the shortest total travel distance. We seek to minimize the total travel distance over all possible routes.
- Any tour corresponds to a permutation of the integers $1, \ldots, p$, which specifies the sequence in which the cities are visited. There are $(p-1)!/2$ possible routes (since the point of origin and direction of travel are arbitrary).

p	$(p-1)!/2$
10	181,440
20	6.08e+16
30	4.42e+30

- Statisticians frequently face combinatorial problems where the search space $\mathbf{\Theta}$ has $p!$ or 2^p elements, where p defines the "size" of the problem.

Complexity

- The traveling salesman problem is NP-complete. There is no known solution in $\mathcal{O}(p^k)$ steps for any k.
- Consider two problems. The first can be solved in $\mathcal{O}(p^2)$ operations, and the second $\mathcal{O}(p!)$ operations.

	Time to solve problem of order...	
p	$\mathcal{O}(p^2)$	$\mathcal{O}(p!)$
20	1 minute	1 minute
21	1.10 minutes	21 minutes
25	1.57 minutes	12.1 years
30	2.25 minutes	207 million years
50	6.25 minutes	2.4×10^{40} years

The big number exceeds by far the age of the universe!

Heuristic search

- The traveling salesman and similar problems are inherently too difficult to be solved exactly.
- Abandon algorithms that are guaranteed to find the global maximum (under suitable conditions) but will never succeed within a practical time limit.
- Turn instead to algorithms that can find a good local maximum within tolerable time.
- We need to trade global optimality for speed.

Overview

Optimization over configuration parameters

- In statistical applications, it is not uncommon for a likelihood function to depend on
  - Configuration parameters $\boldsymbol{\theta}$ that describe the form of a statistical model and for which there are many discrete choices, and
  - A small number of other parameters $\boldsymbol{\beta}$ that are easily optimized if the best configuration were known.
- In such cases, we define the profile log-likelihood of a configuration $\boldsymbol{\theta}$ as

$$\ell(\boldsymbol{\theta}) = \sup_{\boldsymbol{\beta}} \ell(\boldsymbol{\theta}, \boldsymbol{\beta})$$

  i.e., the highest log-likelihood attainable using that configuration.
- We typically want to maximize $\ell(\boldsymbol{\theta})$, or the sum of $\ell(\boldsymbol{\theta})$ and a term depending on the number of free parameters in configuration $\boldsymbol{\theta}$.

Example 1: Variable selection in regression

- Given a dependent variable Y and a set of candidate predictors $x_1,\ x_2, \ldots, x_p$, we must find the best model of the form

$$Y = \beta_0\theta_0 + \sum_{j=1}^{p} \beta_j\theta_j x_j + \epsilon$$

  where $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_p) \in \boldsymbol{\Theta} = \{0,1\}^{p+1}$.
- In the expression above, $\theta_j = 1$ means that covariate x_j is in the model, and $\theta_j = 0$ means that covariate x_j is omitted.
- The cardinality of $\boldsymbol{\Theta}$ is 2^{p+1}, since each variable and the intercept may be included or omitted.

Example 1: Variable selection in regression (continued)

- For $\boldsymbol{\theta}$ fixed, we maximize the log-likelihood $\ell(\boldsymbol{\theta}, \boldsymbol{\beta})$ wrt $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_p)$ and get the profile log-likelihood value $\ell(\boldsymbol{\theta})$.
- The goal is to select the best model according to the Akaike information criterion (AIC), i.e., to minimize

$$\text{AIC}(\boldsymbol{\theta}) = -2\ell(\boldsymbol{\theta}) + 2q,$$

  where $q = \sum_{j=0}^{p} \theta_j$ is the number of parameters in the linear model indexed by $\boldsymbol{\theta}$.
- The variable selection problem requires an optimization over the space of 2^{p+1} possible models.

Example 2: Clustering

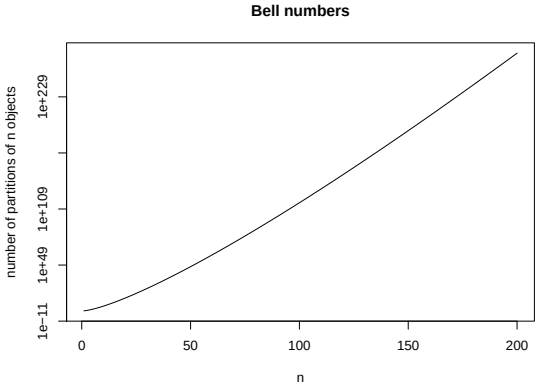
- A partition of a set $\mathcal{O}$ is a grouping of its elements into non-empty subsets, in such a way that every element is included in exactly one subset.
- Problem: Find a partition θ of a set of n objects that optimizes some criterion.
- Let $\mathbf{\Theta}$ be the set of all partitions. Cardinality of $\mathbf{\Theta}$?

Search space cardinality

- Number of partitions on n points: Bell numbers

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

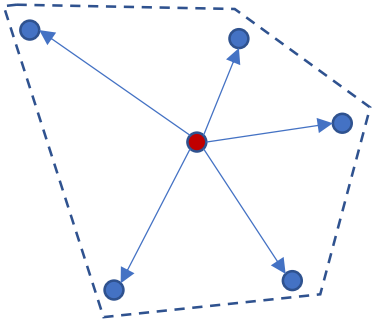
- $B_1 = 1, B_2 = 2, \ldots, B_5 = 52, B_6 = 203, B_{10} = 115975, B_{20} \approx 5 \cdot 10^{13}$



Overview

Basic idea

- A simple, yet efficient approach.
- Relies on
  - iterative improvement of a current candidate solution, and
  - limitation of the search to a local neighborhood at any particular iteration.
- $\boldsymbol{\theta}^{(t)} \longrightarrow \boldsymbol{\theta}^{(t+1)}$ is a move or step.



Definition of neighborhood $\mathcal{N}(\boldsymbol{\theta}^{(t)})$

- Keep it simple. For instance, define $\mathcal{N}(\boldsymbol{\theta}^{(t)})$ by allowing k changes to the current candidate solution. This is a k-neighborhood, and any move is called a k-change.
- In the regression model selection problem, suppose $\boldsymbol{\theta}^{(t)}$ is a binary vector coding the inclusion/exclusion of each potential predictor.
  - A 1-neighborhood is the set of models that either add or omit one predictor from $\boldsymbol{\theta}^{(t)}$
  - A 2-neighborhood is the set of models that either add or omit one or two predictors from $\boldsymbol{\theta}^{(t)}$
  - Etc.

Search strategies and stopping criterion

- Search strategies
  - Simple ascent: $\boldsymbol{\theta}^{(t+1)}$ = any better $\boldsymbol{\theta}$ in $\mathcal{N}(\boldsymbol{\theta}^{(t)})$.
  - Steepest ascent: $\boldsymbol{\theta}^{(t+1)}$ = best $\boldsymbol{\theta}$ in $\mathcal{N}(\boldsymbol{\theta}^{(t)})$.
- The algorithm stops when no better solution is found in the neighborhood of the current solution. We have then found a local maximum of g.

Random starts local search

- Approach:
  - Select many starting points by stratified or completely at random sampling
  - Run simple/steepest ascent from each point
  - Choose best solution as final answer.
- Works surprisingly well compared to more sophisticated local search techniques.
- The random starts strategy can be overlaid on any optimization method to improve the odds of finding a globally competitive final answer, or even the global optimum.

Example: Regression with 27 predictors and 2^{27} possible models

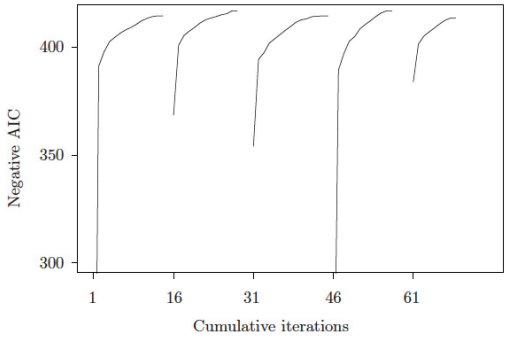
Table: Local search (LS) model selection results using 1-change steepest ascent, for 5 random starts.

	Predictors selected																							
Method	1	2	3	6	7	8	9	10	12	13	14	15	16	18	19	20	21	22	24	25	26	AIC
LS (2,4)		●	●	●		●		●		●	●	●	●						●	●	●	−416.95
S-Plus	●		●	●		●		●		●	●	●	●						●	●	●	−416.94
LS (1)			●	●		●		●	●	●	●					●	●	●				−414.60
LS (3)			●	●		●		●		●	●					●	●	●	●	●		−414.16
LS (5)			●			●	●	●		●	●				●	●	●	●				−413.52
Efroymson			●		●	●		●		●	●			●					●		●	−400.16

The S-Plus (function `step()`) and Efroymson's methods are greedy stepwise procedures.

Example: Regression (continued)

- Results of random starts local search by 1-change steepest ascent for the regression example, for up to 15 iterations from each of five random starts.
- Only AIC values between -300 and -420 are shown.



Overview

Medoid-based clustering

- Let $\mathcal{O} = \{o_1, \ldots, o_n\}$ be a set of objects described by a dissimilarity matrix $D = (d_{ij})$. No numerical attributes are needed.
- A medoid is an object chosen to represent a cluster. A set of K medoids

$$M = \{m_1, \ldots, m_K\} \subset \{1, \ldots, n\}$$

  defines a partition: each object is assigned to its nearest medoid.
- Quality of the partition:

$$W(M) = \sum_{i=1}^{n} \min_{1 \leq k \leq K} d_{im_k} \quad \text{(to be minimized)}.$$

- Search space: $\binom{n}{K}$ subsets, e.g., $7.5 \cdot 10^7$ for $n = 100$ and $K = 5$.
- Only partitions of the nearest-medoid form are reachable: this is a restriction of the problem.

The PAM algorithm

- Partitioning Around Medoids (Kaufman and Rousseeuw, 1990): the reference heuristic for the k-medoids problem.
- BUILD (initialization): take as first medoid the most central object,

$$m_1 = \arg\min_i \sum_{j=1}^{n} d_{ij},$$

  then add medoids one at a time, each time choosing the object whose addition most decreases $W(M)$, until $|M| = K$.
- SWAP (local search): for every pair (m, o) with $m \in M$ and $o \notin M$, compute

$$\Delta(m, o) = W\big((M \setminus \{m\}) \cup \{o\}\big) - W(M),$$

  and perform the swap with the most negative Δ. Repeat until no swap improves W.
- SWAP is a steepest ascent in the 1-neighborhood, of size $K(n-K)$. It therefore stops at a local optimum, and BUILD provides a single starting point.

Overview

Overview

Physical analogy

- Motivated by analogy to the physics of slowly cooling solids. The method models the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy.
- Traditionally problems are posed as minimizations. We adopt this convention for this algorithm only.
- Relies on a sequence of numbers, τ_0, τ_1, $\tau_2 \ldots$, called temperatures. Temperature parameterizes how willing the algorithm is to make non-improving steps.

Basic idea

- Starts at time $t = 0$ with an initial point $\boldsymbol{\theta}^{(0)}$ and a temperature τ_0. The algorithm is run in stages indexed by $j = 0, 1, 2, \ldots$, and each stage consists of several iterations. The length of the jth stage is m_j.
- The new candidate solution is
  - always adopted when it is better than the current solution, and
  - sometimes adopted even when it is worse (with a probability depending on the temperature).
- This is stochastic descent designed to enable escape from uncompetitive local minima.

Algorithm

1. Select a candidate solution $\boldsymbol{\theta}^*$ within the neighborhood of $\boldsymbol{\theta}^{(t)}$, say $\mathcal{N}(\boldsymbol{\theta}^{(t)})$, according to a proposal density $f^{(t)}(\cdot \mid \boldsymbol{\theta}^{(t)})$.
2. Randomly decide whether to adopt $\boldsymbol{\theta}^*$ as the next solution or to keep another copy of the current solution. Specifically, we have $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^*$ with probability equal to

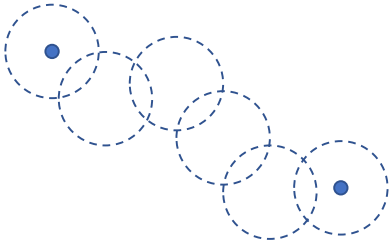
$$p^{(t)} = \begin{cases} 1 & \text{if } g(\boldsymbol{\theta}^*) \leq g(\boldsymbol{\theta}^{(t)}) \\ \exp\left\{ \dfrac{g(\boldsymbol{\theta}^{(t)}) - g(\boldsymbol{\theta}^*)}{\tau_j} \right\} & \text{if } g(\boldsymbol{\theta}^*) > g(\boldsymbol{\theta}^{(t)}) \end{cases}$$

   Otherwise, let $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$.
3. Repeat steps 1 and 2 a total of m_j times.
4. Increment j. Update $\tau_{j+1} = \alpha(\tau_j)$ and $m_{j+1} = \beta(m_j)$. Go to step 1.

$\alpha(\cdot)$ should slowly decrease temperatures to zero.
$\beta(\cdot)$ should increase stage lengths as temperatures drop.

Neighborhoods for simulated annealing

- Small and easily computed. E.g., in variable selection, add or remove one variable.
- Neighborhood structure must allow all elements of $\boldsymbol{\Theta}$ to communicate (any two candidate solutions must be connectable via a sequence of neighborhoods).
- Common proposal distribution: discrete uniform over $\mathcal{N}(\boldsymbol{\theta}^{(t)})$.



Overview

Cooling schedule and convergence

- Simulated annealing produces a Markov chain $\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ because $\boldsymbol{\theta}^{(t+1)}$ depends only on $\boldsymbol{\theta}^{(t)}$.
- The distribution of $\boldsymbol{\theta}^{(t)}$ converges to some stationary distribution as $t \to \infty$.
- In principle, we would like to run the chain at each fixed temperature long enough that the Markov chain is approximately in its stationary distribution before the temperature is reduced.
- Theoretical analysis of cooling schedules leads to "recommendations" requiring impracticably long run lengths, sometimes longer than exhaustive search.

Theoretical results

Theorem (Stationary distribution)

- *Let τ be the (fixed) temperature. Suppose that proposing $\boldsymbol{\theta}_i$ from $\mathcal{N}(\boldsymbol{\theta}_j)$ has the same probability as proposing $\boldsymbol{\theta}_j$ from $\mathcal{N}(\boldsymbol{\theta}_i)$ for any pair of solutions $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ in $\boldsymbol{\Theta}$.*
- *Then the sequence of $\boldsymbol{\theta}^{(t)}$ generated by simulated annealing is a Markov chain with stationary distribution*

$$\pi_\tau(\boldsymbol{\theta}) \propto \exp\{-g(\boldsymbol{\theta})/\tau\}$$

- *Consequently,*

$$\lim_{t\to\infty} P(\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}) = \pi_\tau(\boldsymbol{\theta})$$

Theoretical results (continued)

Theorem (Convergence to a global minimum)

Suppose there are M global minima and the set of these solutions is $\mathcal{M}$. Then,

$$\lim_{\tau \downarrow 0} \pi_\tau(\boldsymbol{\theta}_i) = \begin{cases} 1/M & \text{if } i \in \mathcal{M} \\ 0 & \text{otherwise.} \end{cases}$$

Interpretation: If, for each τ, we wait long enough to reach the stationary distribution, and if we let τ tend to 0, then we are sure to reach a global minimum (after a possibly infinite amount of time!).

Overview

Cooling schedule

- Popular approach: set $m_j = 1$ for all j and reduce temperature very slowly according to

$$\alpha(\tau_j) = \frac{\tau_j}{1 + a\tau_j}$$

for a small value of a.

- A second option is to set

$$\alpha(\tau_j) = a\tau_j \quad \text{for} \quad a < 1.$$

(usually $a \geq 0.9$). Increase stage lengths as temperatures decrease, using, e.g.,

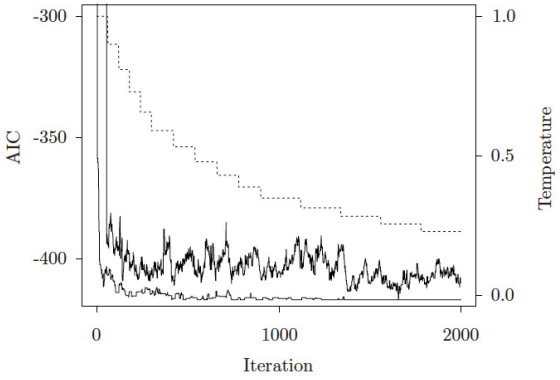
$$\beta(m_j) = bm_j \quad \text{for} \quad b > 1.$$

Initial temperature

- Selection of the initial temperature, τ_0, is usually problem dependent.
- Try using a positive τ_0 value so that $\exp\{(g(\boldsymbol{\theta}_i) - g(\boldsymbol{\theta}_j))/\tau_0\}$ is close to one for any pair of solutions $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ in $\boldsymbol{\Theta}$. This provides any point in the parameter space with a reasonable chance of being visited in early iterations of the algorithm.
- Long runs at high temperatures are unnecessary. Decrease temperature rapidly at first but slowly thereafter.

Example: regression

- Temperature for the bottom curve is shown with the dotted line and the right axis. Neighborhoods given by adding or deleting one predictor.
- Discrete uniform proposal. 15-stage cooling schedule with stage lengths of 5×60, 5×120, and 5×220.
- Cooling given by $\alpha(\tau_{j-1}) = 0.9\tau_{j-1}$ after each stage.
- The bottom curve corresponds to $\tau_0 = 1$, where we observe sticking because there is little tolerance for uphill moves.
- A second run with $\tau_0 = 10$ (top solid line) yielded considerable mixing, with many uphill proposals accepted as moves.



Simulated annealing in R

- Function `optim`, select `method="SANN"`.
- For continuous optimization.
- Parameters:
  - `control$temp`: τ_0 (default $= 10$)
  - `control$tmax`: number m_j of function evaluations at each temperature (default $= 10$)
  - `gr`: function to generate a new candidate point.
- Cooling schedule:

$$\tau_{j+1} = \frac{\tau_0}{\log(j \, \text{tmax} + e)}$$

Overview

Overview

Population-based search

- Local search and simulated annealing explore $\mathbf{\Theta}$ along a single trajectory: only one candidate solution is kept at any time, and new solutions are obtained by perturbing it.
- Genetic algorithms (GAs) instead maintain a whole population of candidate solutions, and new solutions are obtained by recombining pairs of existing ones.
- The diversity of the population is what drives the search: good "building blocks" discovered in different solutions can be combined into better ones.

Biological analogy

- GAs mimic the process of Darwinian natural selection.
- Candidate solutions are seen as biological organisms represented by their genetic code; the fitness of an organism is analogous to the quality of a candidate solution.
- Breeding among highly fit organisms makes it possible to pass along desirable attributes to future generations, while breeding among less fit organisms (and rare genetic mutations) ensures population diversity.
- Over time, the organisms in the population evolve to become increasingly fit, thereby providing a set of increasingly good candidate solutions to the optimization problem.

Chromosomes, genes

- Every candidate solution corresponds to an individual or organism, and every organism is completely described by its chromosome, a sequence of C symbols

$$\gamma = (\gamma_1, \ldots, \gamma_C).$$

- The C elements of the chromosome are the genes, and the symbols that can be stored in a gene form a pre-determined alphabet, whose elements are called alleles.
- Most often the alphabet is binary, i.e., $\gamma_c \in \{0, 1\}$, and a chromosome is a bit string such as '100110001' (here, $C = 9$).

Genotype vs. phenotype

- The information encoded in an individual's chromosome is its genotype, γ. The expression of the genotype in the organism itself is its phenotype, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$.
- Designing a GA thus requires an encoding of the candidate solutions as chromosomes, together with the corresponding decoding map $\gamma \mapsto \boldsymbol{\theta}$.
- The fitness of an organism is the value $g(\boldsymbol{\theta})$ of the objective function at the corresponding phenotype.
- Example: regression model selection. Assume 9 predictors, the intercept being always included (hence, $C = 9$).
  - The genotype $\gamma =$'100110001' is decoded into the phenotype of a model containing only the fitted parameters for predictors 1, 4, 5, and 9.
  - The fitness of this individual is $g(\boldsymbol{\theta}) = -\text{AIC}(\boldsymbol{\theta})$, the negative AIC of that model.

Generations, breeding

- GAs consider many candidate solutions simultaneously. Let the t-th generation consist of a collection of P organisms $\boldsymbol{\theta}_1^{(t)}, \ldots, \boldsymbol{\theta}_P^{(t)}$ having corresponding genotypes (encodings) $\boldsymbol{\gamma}_1^{(t)}, \ldots, \boldsymbol{\gamma}_P^{(t)}$; the fitness of individual i in that generation is $g(\boldsymbol{\theta}_i^{(t)})$.
- An offspring is a new organism inserted in the $(t+1)$-th generation to replace a member of the t-th generation. The offspring's chromosome is determined from two parent chromosomes belonging to the t-th generation.
- Whether the parents themselves survive into generation $t+1$ depends on the replacement scheme, defined later by the generation gap.
- If fit parents are predominantly selected for breeding, then, as generations progress, organisms inherit from their parents bits of genetic code that are associated with high fitness, and the population drifts towards the regions of $\boldsymbol{\Theta}$ where g is large.

Overview

Generic genetic algorithm

1. Initialization: draw P chromosomes at random to form generation $t = 0$, and evaluate their fitness.
2. Selection: choose pairs of parents in generation t, using a mechanism that favors the fittest individuals.
3. Crossover: combine the chromosomes of each pair of parents to produce offspring chromosomes.
4. Mutation: randomly alter a few genes in the offspring chromosomes.
5. Replacement: form generation $t + 1$ by inserting the offspring in the population, in place of some of the current individuals; evaluate the fitness of the new individuals.
6. Increment t and return to step 2.

Stopping rule: maximum number of generations, or no improvement of the best fitness over a given number of consecutive generations.

Selection mechanisms

- The process by which parents are chosen to produce offspring is called the selection mechanism. It must balance selection pressure (favoring fit individuals) against diversity (avoiding premature convergence).
- Fitness-proportional selection (simplest): select one parent with probability proportional to fitness and select the other parent completely at random, or select both parents with probability proportional to fitness. This requires $g > 0$ and is sensitive to the scale of g: a single very fit individual may quickly invade the population.
- Rank-based selection (better): select individual i with probability

$$\frac{r_i^{(t)}}{P(P+1)/2},$$

where $r_i^{(t)}$ is the rank of $g(\boldsymbol{\theta}_i^{(t)})$ among generation t, sorted in increasing order ($r = 1$ for the least fit individual, $r = P$ for the fittest), and $P(P+1)/2 = \sum_{i=1}^{P} r_i^{(t)}$ is the normalizing constant. Only the ordering of the fitness values matters.

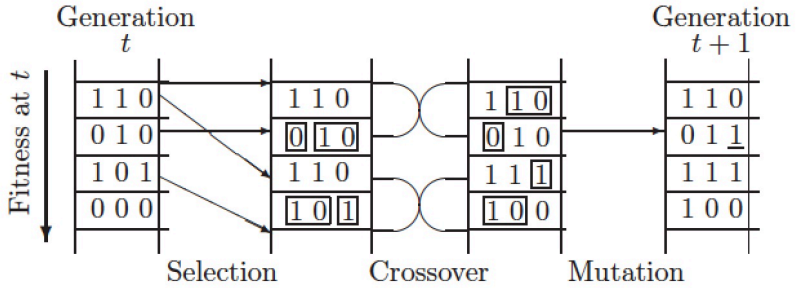
Selection mechanisms (continued)

- Tournament selection (popular):
  - Randomly partition the set of chromosomes in generation t into k disjoint subsets of equal size.
  - Select the best individual in each group as a parent.
  - Each partition yields k parents, so the operation is repeated until enough parents have been obtained.
  - Finally, randomly pair parents for breeding.
- The selection pressure is controlled by the size P/k of the subsets: the smaller k, the larger the subsets, and the more the fittest individuals are favored.

Crossover and mutation

- After two parents from the t-th generation have been selected for breeding, their chromosomes are combined in some way that allows attributes from each parent to be inherited by their offspring, who become members of generation $t+1$. The most widely used method is crossover.
- Crossover (See next slide):
  - Select a random position c, uniformly in $\{1, \ldots, C-1\}$, and split both parent chromosomes between genes c and $c+1$.
  - Glue the left chromosome segment from one parent to the right segment from the other parent to form an offspring chromosome.
  - the two complementary offspring may both be kept.
- Mutation:
  - Usually applied after crossover, it changes each gene of the offspring into another allele, independently, with a small probability μ called the mutation rate.
  - It is the only mechanism able to restore alleles that have disappeared from the population.

Example: production of one generation



Production of a new generation in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Replacement

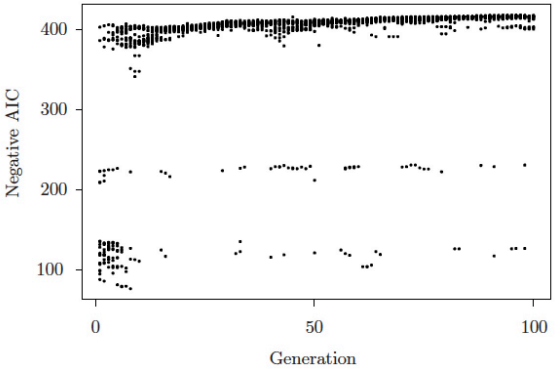
- Define the generation gap, G, to be the proportion of the current generation that is replaced by offspring at each iteration.
- $G = 1$ corresponds to a canonical GA with distinct, non-overlapping generations: the whole population is renewed at once.
- At the other extreme, $G = 1/P$ corresponds to incremental updating of the population one offspring at a time. Such a steady state GA produces one offspring at a time to replace the least fit (or some random relatively unfit) individual.
- When $G < 1$, part of the current generation survives; we can then ensure that the k best chromosomes are always kept in the next generation (elitist strategy), so that the best fitness never decreases.

Implementation guidelines

- Initial population: Purely random chromosomes, so as to cover $\mathbf{\Theta}$ as widely as possible.
- Generation size:
  - Larger provides greater search diversity, but also greater computing burden, as the fitness must be evaluated for every new individual.
  - For binary encoding of chromosomes, try $C \leq P \leq 2C$, where C is the chromosome length.
  - Real applications often have $10 \leq P \leq 200$, but a review of empirical studies suggests that P can often be as small as 30.
- Mutation rate: the probability μ that a given gene mutates. It is typically very low, in the neighborhood of 1%. Studies have suggested a rate of $1/C$ or a rate roughly proportional to $1/(P\sqrt{C})$.
- Too low a mutation rate leads to premature convergence, while too high a rate turns the search into a random walk.

Example: regression model selection

- Results of a genetic algorithm for regression model selection, with 100 generations of size $P = 20$, using rank-based selection, simple crossover, and a 1% mutation rate.
- Darwinian 'survival of the fittest' is clearly seen.
- The twenty random starting individuals quickly coalesce into three effective subspecies, with the best of these slowly overwhelming the rest.
- The best model was first found in generation 87.



Genetic algorithms in R

- Package `GA`.
- Main function: ga
- Some arguments:
  - `type`: set to `"binary"` (other values: `"real-valued"`, `"permutation"`)
  - `fitness`: the fitness function, which is maximized
  - `popSize`: the population size P (default = 50)
  - `pcrossover`: the probability of crossover between pairs of chromosomes. Default = 0.8.
  - `pmutation`: the probability of mutation in a parent chromosome. Default = 0.1. This is a rate per chromosome: the default binary operator then flips one gene chosen at random, i.e., a per-gene rate of about $0.1/C$.
  - `elitism`: the number of best fitness individuals to survive at each generation. By default the top 5% individuals will survive at each iteration.
  - `maxiter`: the maximum number of generations. Default = 100.
  - `run`: the number of consecutive generations without any improvement in the best fitness value before the GA is stopped.

Overview

Application to clustering

- The PAM algorithm is a local search procedure: it can easily get stuck in a local optimum.
- A GA could overcome this limitation, at the cost of greater computing time.
- First problem: encoding of a solution as a binary vector (chromosome).
- Two situations will be considered:
  - the number of clusters K is fixed: the encoding is constrained, and the genetic operators must be adapted;
  - the number of clusters is free: the encoding is unconstrained and the standard operators apply, but the fitness must be changed.

Encoding

- Chromosome: binary string of length $C = n$, with

$$\gamma_i = 1 \iff \text{object } i \text{ is a medoid.}$$

  E.g., for $n = 8$, '01001010' encodes $M = \{2, 5, 7\}$.
- Every medoid set has exactly one encoding: the decoding map is injective.
- If K is fixed, the chromosomes are subject to the constraint $\sum_{i=1}^{n} \gamma_i = K$: the feasible set is not $\{0, 1\}^n$, and the standard operators must be adapted.
- Why not a shorter chromosome of length K, gene k holding the index of the k-th medoid? Because crossover may then produce duplicate medoids, and each medoid set has $K!$ encodings: two parents encoding the same solution may breed a poor offspring.

Decoding

- The dissimilarity matrix D is computed once, before running the GA.
- Decoding of a chromosome γ:
  1. $M \leftarrow \{i : \gamma_i = 1\}$, $K \leftarrow |M|$;
  2. assign each object i to the cluster $k(i) = \arg\min_k d_{im_k}$, with cost $c_i = d_{im_{k(i)}}$.
- Cost: $\mathcal{O}(nK)$. The assignment itself need not be stored during the search: the chromosome carries all the information.
- The fitness is then computed from the partition; it is

$$g(\gamma) = -W(M) = -\sum_{i=1}^{n} c_i$$

if K is fixed, but not if K is free (see below).

Fixed K: initialization and mutation

- Initial population: for each of the P individuals, draw K distinct objects uniformly at random in $\{1, \ldots, n\}$.
- Swap mutation: draw one gene equal to 1 and one gene equal to 0, and exchange their values, i.e., replace one medoid by a non-medoid.
- The constraint $\sum_i \gamma_i = K$ is preserved by construction, whereas the standard bit-flip mutation would violate it.
- This is exactly the elementary move of the PAM algorithm.

Fixed K: crossover

- One-point crossover does not preserve the constraint. Instead, let M_1 and M_2 be the medoid sets of the two parents, and
  - keep the common medoids $I = M_1 \cap M_2$, with $a = |I|$;
  - draw the $K - a$ remaining medoids uniformly at random in the symmetric difference $S = (M_1 \cup M_2) \setminus I$, with $|S| = 2(K - a)$.
- The offspring has exactly K medoids, all inherited from a parent; two identical parents produce that same individual.
- Example ($n = 8$, $K = 3$): $M_1 = \{2, 5, 7\}$ and $M_2 = \{1, 5, 8\}$ give $I = \{5\}$ and $S = \{1, 2, 7, 8\}$; drawing two elements of S yields, e.g., $M = \{2, 5, 8\}$.
- Simpler alternative: apply the standard operators, then repair the offspring by removing or adding ones at random.

Free K: choice of the fitness

- $-W(M)$ can no longer be used: $W(M)$ decreases as medoids are added, and the GA would converge to n singletons.
- As there is no likelihood here, AIC or BIC do not apply directly: an internal validity index is used instead, which balances the compactness of the clusters against their separation.
- Davies-Bouldin index. Let S_k be the mean dissimilarity between the objects of cluster k and its medoid,

$$S_k = \frac{1}{|C_k|} \sum_{i \in C_k} d_{i m_k},$$

and let $d_{m_k m_l}$ be the dissimilarity between the medoids of clusters k and l. Then

$$DB = \frac{1}{K} \sum_{k=1}^{K} \max_{l \neq k} \frac{S_k + S_l}{d_{m_k m_l}}.$$

- Each term compares the dispersion of two clusters to their separation: DB is small when the clusters are compact and well separated.

Free K: choice of the fitness (continued)

- Fitness: $g(\gamma) = -DB$, to be maximized (DB is defined for $K \geq 2$).
- It is not monotone in K: splitting a genuine cluster produces two clusters that are close to each other relative to their dispersions, while merging two clusters inflates the S_k's.
- Cost: $\mathcal{O}(nK + K^2)$ per evaluation. The S_k are by-products of the decoding step, and the $d_{m_k m_l}$ are read in D: no additional pass over the n^2 dissimilarities is needed.
- Different indices may lead to different numbers of clusters: the choice of the index is part of the definition of the problem.

Free K: operators

- Any $\gamma \in \{0,1\}^n$ with at least two ones is now a valid solution: the standard operators can be used, as in the regression example.
- Bit-flip mutation adds or removes a medoid: the number of clusters changes during the search.
- Only the initial population deserves attention: purely random chromosomes contain about $n/2$ medoids. It is preferable to draw, for each individual, a number of medoids uniformly in $\{2, \ldots, K_{\max}\}$, then their positions at random.
- The problem is then formally the same as variable selection (Example 1), with objects in place of predictors: the ones of the chromosome select a subset, and the fitness penalizes subsets that are too large or too small.

Hybridization

- Pure GAs converge slowly on this problem.
- Hybrid (or memetic) GA: apply one or two PAM swap steps — steepest ascent in the neighborhood of size $K(n-K)$ — to each offspring, before evaluating its fitness.
- The GA provides the global exploration that PAM lacks, as PAM stops at the first local optimum; local search provides the refinement that the GA is bad at.
- Close to random starts local search, with recombination between the starts.

Clustering GA in R

- Dissimilarities: `dist`, or `daisy` (package `cluster`) for mixed-type data. Local search step: `pam`. The index DB is computed directly from D; the silhouette is available as `cluster::silhouette`.
- Free K: a call to `ga(type="binary", nBits=n, fitness=...)` with the default operators is enough.
- Fixed K: the constraint-preserving operators must be supplied through the arguments `population`, `crossover` and `mutation`:
  - `population(object)`: returns the initial `popSize` $\times$ `nBits` matrix;
  - `crossover(object, parents, ...)`: `parents` indexes the two parents in `object@population`; returns `list(children=..., fitness=rep(NA,2))`;
  - `mutation(object, parent, ...)`: `parent` is an index; returns the mutated chromosome.