UiO : Matematisk institutt
Det matematisk-naturvitenskapelige fakultet

STK-4051/9051 Computational Statistics Spring 2022
Combinatorial Optimization

Instructor: Odd Kolbjørnsen, oddkol@math.uio.no

Optimization and decision

- Optimization: Solve $\max_\theta f(\theta)$
- Decision: Is there a $\theta \in \Theta$ for which $f(\theta) > c$?
- Optimization problem can be solved by repeatedly solving decision problems for different values of c.
- Decision problems that can be solved in polynomial time ($\mathcal{O}(p^k)$ operations) are generally considered to be efficiently solvable. Called P problems
- Decision problems that can be checked in polynomial time called NP problems
- P$\subset$NP
- NP hard: solution to one such problem can be used to solve any NP problem
- NP complete: problem is both NP and NP hard

NP - hard

- **P** (**P**olynomial-time)**:** decision problems that can be solved in polynomial time
- **NP** (**N**on-deterministic **P**olynomial-time)**:** decision problems that can be checked in polynomial time
- We do not know if **NP** problem can be solved in polynomial time
- **NP-hard:** (**N**on-deterministic **P**olynomial-time hard) problems that are "at least as hard as the hardest problems in NP“ Solution to a NP-hard problem can be used to solve any NP problem
- **NP-complete:** subclass which are both NP and NP-hard. The hardest problems among NP.

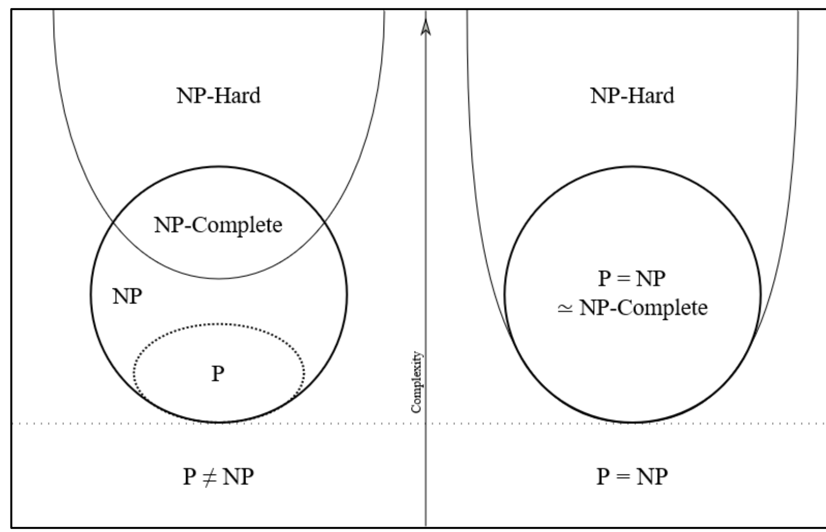
Most people think this →

Which universe do we live in?

$P \neq NP$

or

$P = NP$



Check versus solve

- How do you «check» a problem without solving it?
  - If someone propose a solution θ^* you can check it
    - evaluate the function $f(\theta)$ for θ^*
    - Is $f(\theta^*) > c$?
    - We still do not know how they got the value [lucky guess??]
    - We still do not know if it is the global optimum

- It is harder to find the solution $\arg\max f(\theta)$
  - solve it, find the global optimum with guarantee

NP-complete problems (how hard can it be?)

- Consider two problems:
  - The first can be solved in $\mathcal{O}(p^2)$ operations
  - The second $\mathcal{O}(p!)$ operations.
  - They both require 1 minute of computing time when $p = 20$.

	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

- There are optimization problems that are inherently too difficult to solve exactly by traditional means.
- Many problems in bioinformatics, experimental design, and nonparametric statistical modeling, for example, require combinatorial optimization.
- (The content of this slide was kindly provided by Givens & Hoeting)

Model selection

- Genetic association studies: Which genes influence a certain phenotype (presence of cancer, size, etc)
- Linear model including all possible variables:

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \varepsilon_i$$

- Reasonable to assume that some x_{ij}'s do not influence the response, modification:

$$Y_i = \beta_0 + \sum_{j=1}^{p} \gamma_j \beta_j x_{ij} + \varepsilon_i$$

where $\gamma_j \in \{0, 1\}$.

- 2^p possible models, how to choose the best one?
  - $p = 20, 2^p = 1\,048\,576$, $p = 100, 2^p = 1.267651 * 10^{30}$
- Combinatorial problem, discrete optimisation

Need for heuristics

- When no algorithm guaranties a global maximum (within a time frame)

- Heuristics: Algorithms that find a good local optima within tolerable time
  - Local search
  - Simulated annealing
  - Tabu algorithm
  - Genetic algorithm

Local search

- Iterative improvement: $\theta^{(t)} \rightarrow \theta^{(t+1)}$ (Move or step)
- Limiting the search to a **local neighborhood** $\mathcal{N}(\boldsymbol{\theta}^{(t)})$ at any particular iteration
  - Example model selection : (change only one component)
    $\mathcal{N}\left(\boldsymbol{\theta}^{(t)}\right) = \{\boldsymbol{\theta}\colon \exists\, l \text{ such that } \theta_j = \theta_j^{(t)} \text{ for } j \neq l\,\}$
- Steepest ascent

$$\theta^{(t+1)} = \operatorname{argmax}_{\boldsymbol{\theta}\in\mathcal{N}(\boldsymbol{\theta}^{(t)})} g(\boldsymbol{\theta})$$

- Random ascent:
  - Test random samples $\boldsymbol{\theta}_S$from $\mathcal{N}\left(\boldsymbol{\theta}^{(t)}\right)$
  - $\theta^{(t+1)}$ first sample such that $g(\boldsymbol{\theta}_S) > g\left(\boldsymbol{\theta}^{(t)}\right)$
- Balance: neighborhood size vs speed

Random starting points combined with local search

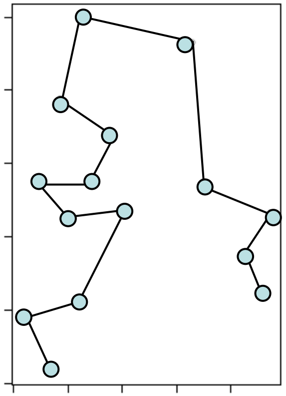
- Select many starting points
  - Stratified or random sampling
- Run local search from each staring point
  - Random or steepest ascent
- Select best final answer

- Works very well in many cases

- Random starting point can be used for any optimization method. (Build confidence in optimum)

Example – Traveling salesperson problem

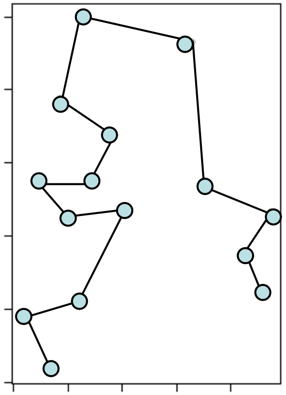
- A salesman needs to visit p cities
- Each city visited only once
- What is the minimum distance needed in order to visit all the cities?



- `Travel_salesman_greedy.R`

Example – Traveling salesperson problem

- A salesman needs to visit *p* cities
- Each city visited only once
- What is the minimum distance needed in order to visit all the cities?



- Example: Traveling salesperson

  Check:

  Compute the travel time along one specific path

  N – operations (N = number of cities)

  Solve:

  Find the optimal route for the traveling salesman

  N!   possibilities (number of orderings)

Simulated annealing

- Annealing (chemistry)
  - Heating up a solid (increasing energy) and then cooling down (decreasing energy)
  - Slow cooling: State with minimal energy
  - Fast cooling: Local minima
- Simulated annealing: Numerical algorithm resembling annealing

$$\min_{\theta} f(\theta)$$

1: Start with $\theta^{(0)}$

2: At stage j: Repeat m_j times

- Generate a candidate solution $\theta^* \in \mathcal{N}(\theta^{(t)})$
- Put

$$\theta^{(t+1)} = \begin{cases} \theta^* & \text{with probability } \min(1, \exp\{[f(\theta^{(t)}) - f(\theta^*)]/\tau_j\} \\ \theta^{(t)} & \text{otherwise} \end{cases}$$

3: Update $\tau_{j+1} = \alpha(\tau_j)$ and $m_{j+1} = \beta(m_j)$

= Cooling schedule

- If $f(\theta^*) \leq f(\theta^{(t)})$, we always move to candidate solution
- If $f(\theta^*) > f(\theta^{(t)})$, we may move to candidate solution
  - For τ_j large, high probability for moving ("high temperature")
  - For τ_j small, small probability for moving ("low temperature")
- Makes it possible to move out of modes
- Store «best so far» in addition to iterations (for the end game)

Practical issues – Simulated annealing

- Neighborhoods
  - Problem dependent, but small neighborhoods typically most efficient
  - Need a neighborhood so that all solutions in Ω communicate:
    - For all θ, θ^*, there exist a finite set $\theta_1, \ldots, \theta_k$ such that $\theta_1 \in \mathcal{N}(\theta), \theta_{j+1} \in \mathcal{N}(\theta_j)$ for $j = 1, \ldots, k-1$, $\theta^* \in \mathcal{N}(\theta_k)$
- Proposals
  - Most common to choose uniformly within $\mathcal{N}(\theta)$
- Efficiently calculation of $f(\theta^*)$
  - In many cases $f(\theta^*)$ can be efficiently updated from $f(\theta)$
- Cooling schedule: $\tau_{j+1} = \alpha(\tau_j)$ and $m_{j+1} = \beta(m_j)$
  - If $m_j = 1$, then $\tau_j = c/\log(1+j)$ guarantees asymptotic convergence to global minimum
  - c is the depth, the smallest increase needed to escape the deepest local minima.
  - In practice, $\tau_j = c/\log(1+j)$ results in too slow convergence, faster cooling schedules typically used

Traveling salesperson Simulated annealing

- Neighborhood: Swap the order of two components
  - Will lead to that all solutions communicate
- Proposal: Draw two indices within $\{1, \ldots, p\}$ randomly
- Cooling schedule: $m_j = 1$, $\tau_j = 1/\log(1+j)$ or $= 10/i$
- Updating $f(\boldsymbol{\theta}^*)$ from $f(\boldsymbol{\theta})$: Assume $j < k$ are swapped

$$
\begin{aligned}
f(\boldsymbol{\theta}^*) =& \sum_{l=1}^{p-1} d(\mathbf{p}(\theta_l^*), \mathbf{p}(\theta_{l+1}^*)) \\
=& f(\boldsymbol{\theta}) + d(\mathbf{p}(\theta_{j-1}^*), \mathbf{p}(\theta_j^*)) + d(\mathbf{p}(\theta_j^*), \mathbf{p}(\theta_{j+1}^*)) + \\
& d(\mathbf{p}(\theta_{k-1}^*), \mathbf{p}(\theta_k^*)) + d(\mathbf{p}(\theta_k^*), \mathbf{p}(\theta_{k+1}^*)) - \\
& d(\mathbf{p}(\theta_{j-1}), \mathbf{p}(\theta_j)) - d(\mathbf{p}(\theta_j), \mathbf{p}(\theta_{j+1})) - \\
& d(\mathbf{p}(\theta_{k-1}), \mathbf{p}(\theta_k)) - d(\mathbf{p}(\theta_k), \mathbf{p}(\theta_{k+1}))
\end{aligned}
$$

- `Travel_salesman_SA.R`

Simulated annealing for continuous function

- Simulated annealing can equally be used for continuous functions
- Main change: Define neighborhood in continuous space
  - Example: $\mathcal{N}(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* : \exists j \text{ such that } \theta_k^* = \theta_k, k \neq j\}$
- Can choose $f(\boldsymbol{\theta}) = L(\boldsymbol{\theta})$ or $f(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta})$
- Typically prefer $f(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta})$ because
  - The depth parameter c will usually be smaller
  - It is typically easier to update $\ell(\boldsymbol{\theta})$

Genetic algorithm background

- Mimics the process of Darwinian natural selection
- Candidate solutions to a maximization problem are envisioned 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 provides the best opportunity to pass along desirable attributes to future generations
- Breeding among less fit organisms (and mutations) ensures population diversity
- Darwin: The population evolve to become increasingly fit
- Consider again maximization of $f(\theta)$

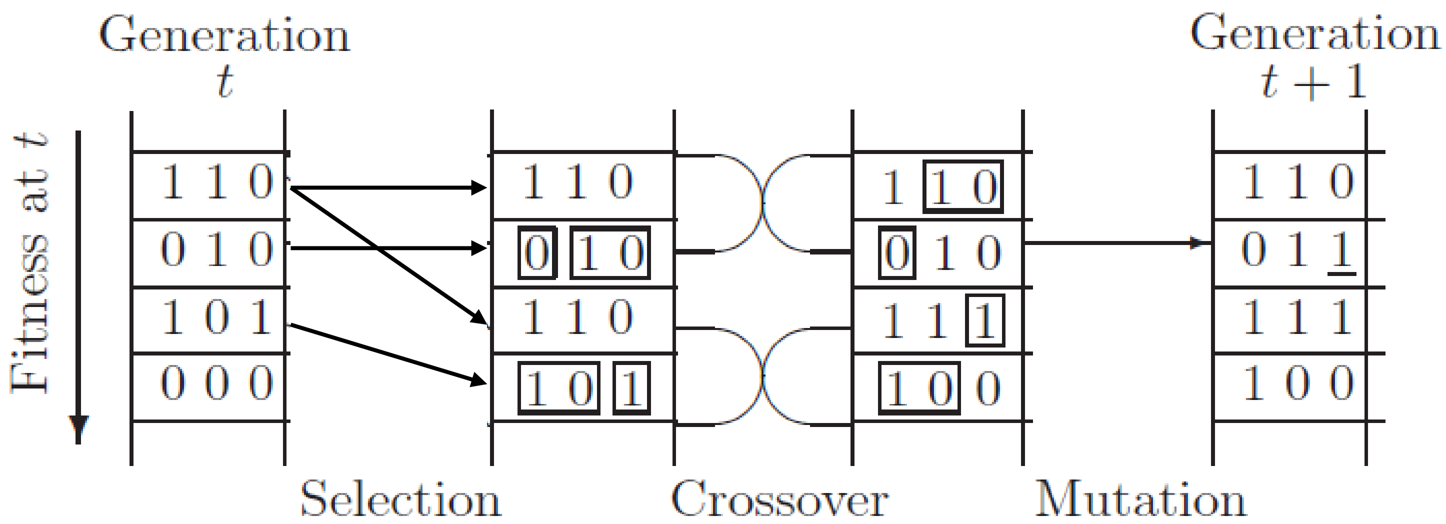
Genetic algorithm (iterations)

- Each iteration *t* contain a collection/population of solutions, $\theta_1^{(t)},\ldots,\theta_P^{(t)}$
- Individuals of next generation $\theta_j^{(t+1)}$ are based on two parents and a stochastic component:

$$\theta_j^{(t+1)} = g(\theta_k^{(t)},\theta_l^{(t)})$$

- Selection mechanism
  - Parents selected with probabilities related to fitness $f(\theta)$
- Genetic operators
  - $\theta_k^{(t)} = (100110001),\ \theta_l^{(t)} = (110101110) \Rightarrow \theta_j^{(t+1)} = (1?01?????)$? =random
  - $\theta_k^{(t)} = (100110001),\ \theta_l^{(t)} = (110101110) \Rightarrow \theta_j^{(t+1)} = (100101110)$ crossover
- Mutations Randomly change one (or a few) components
  - $(10\mathbf{0}101110) \Rightarrow (10\mathbf{1}101110)$
  - Assures that the solution is not limited by the initial population

Schematic example (fig 3.5)



Population size: P = 4
Chromosome length: C=3 (= # of parameters, i.e. p=3)

Genetic algorithm – Practical issues

- Size of population, P
  - For binary components, suggestion: $p \leq P \leq 2p$
  - For permutations, suggestion $2p < P < 20p$
- Mutation rate, μ
  - Low, typically 1%
  - Theoretical results: $\mu = 1/p$ or $\mu = 1/(P\sqrt{C})$
- Selection of parents
  - Probability proportional to $f(\theta_t^{(k)})$
  - Probability proportional to $\exp(f(\theta_t^{(k)}))$
  - Probability proportional to rank of $f(\theta_t^{(k)})$
  - One parent completely random
  - Tournament selection
    - Individuals at iteration t randomly divided into k clusters
    - Best fitted individuals within each cluster used as parents at iteration $t+1$
- Introducing population gap
  - Only a proportion, G, is replaced between each generation

Genetic algorithm baseball salaries

- Salaries for $n = 337$ baseball players
- $p = 27$ possible covariates, $2^{27} = 134\,217\,728$ possible models

Covariates are statistics collected during a season
  - # runs scored
  - batting average
  - on pace percentage
  - …

- Genetic algorithm (for model selection)
  - Starting with $P = 100$ models selected randomly
  - Choose two parents with probabilities proportional to $\exp(-AIC)$
  - For each component choose the state from one of the parents randomly
  - Allow mutation (change) with probability $\mu = 0.01$
  - `Baseball_genetic.R`

Tabu algorithms

- Local (random) search weakness
  - Next move will in many cases reverse previous move
- Tabu idea:
  - Allow downhill move when no uphill move is possible
  - Make some moves temporarily forbidden or tabu
  - Early form: steepest ascent /mildest decent
    - Move to least unfavorable when there is no uphill move

Traveling salesperson Tabu

- Neighborhood: Swap the order of two components
- Move: To the best state in the neighborhood even if it is worse
- Tabu: Do not allow to pick two components that have been selected in the last k iterations
- Implementation:
  - Make a table of all possible pairs that can be picked, a $p(p-1) \times 2$ table
  - Make a list H containing the last k pairs that have been picked (references to the rows in the table above)
  - When searching within neighborhood, do not consider those pairs contained in H
  - When found the best pair, remove the first element of H and add the new pair to the end of H
- `Travel_salesman_tabu.R`

Tabu additional rules

- Aspiration criterion:
  - Allow a tabu move if it is better than the best found state so far
  - Allow a tabu move if it gives a large change
- Diversification
  - Penalize moves to a worse state if such a move has happened many times before
- Intensification
  - Reward moves that retain features that have shown to be important earlier
    - Variable selection: If inclusion of component j correspond to many good solutions, reward moves including this component