STA6171: Statistical Computing for DS 1

Combinatorial Optimization

Ick Hoon Jin

Yonsei University, Department of Statistics and Data Science

2020.09.16

Introduction to Combinatorial Optimization

- Let us assume that we are seeking the maximum of $f(\theta)$ w.r.t $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_p)$, where $\boldsymbol{\theta} = \boldsymbol{\Theta}$ and $\boldsymbol{\Theta}$ consists of N elements for a finite positive integer N.
- Example: Profile Likelihood
  - In statistical applications, it is common for a likelihood function to depend on configuration parameters that describe the form of a statistical model and for which there are many discrete choices, as well as a small number of other parameters that could be easily optimized if the best configuration were known.
  - View $f(\boldsymbol{\theta})$ as the log profile likelihood of a configuration, $\boldsymbol{\theta}$, that is, the highest likelihood attainable using that configuration.

Hard Optimization Problems

- Hard optimization problems are generally combinatorial in nature.
- p items may be combined or sequenced in a very large number of ways. and each choice corresponds to one element in the space of possible solutions.
- Maximization requires a search of this very large space.
- Example: Traveling Salesman Problem
  - Suppose sales man must visit each of p cities exactly once and return to his point of origin using the shortest total travel distance.
  - Seek to minimize the total travel distance over all possible routes.
  - If the distance between two cities does not depend on the direction traveled between them, then there are $(p-1)!/2$ possible routes.
  - The difficulty of optimization depends on p.

Need for Heuristics

- Necessary to abandon algorithms that are guaranteed to find the global maximum under suitable conditions but will never succeed within a practical time limit. Better to use algorithms that can find a good local maximum within tolerable time.
- Heuristics: Intend to find a globablly competitive candidate solution with an explicit trade of global optimality for speed.
  - iterative improvement of a current candidate solution, and
  - limitation of the search to a local neighborhood at any particular iteration.

Example: Variable Selection in Regression

- Consider a multiple linear regression problem with p potential predictor variables and try to select a suitable model.
- Given a dependent variable Y and a set of candidate predictors $x_1, \cdots, x_p$, we must find the best model of the form

$$Y = \beta_0 + \sum_{j=1}^{s} \beta_{i_j} x_{i_j} + \epsilon,$$

  where $\{i_1, \cdots, i_s\}$ is a subset of $\{1, \cdots, p\}$ and ϵ denote a random error.
- The variable selection problem requires an optimization over a space of 2^{p+1} possible models.

Example: Variable Selection in Regression

- Two ways to find the best model
  - Use the Akaike information criterion. Seek to find the subset of predictors that minimizes the fitted model AIC,

    $$\text{AIC} = N \log\{\text{RSS}/N\} + 2(s+2),$$

    where N is the sample size, s is the number of predictors in the model, and RSS is the sum of squared residuals
  - Use Bayesian regression with the normal-gamma conjugate class of priors

    $$\beta \sim N(\mu, \sigma^2 V) \quad \text{and} \quad \nu\lambda/\sigma^2 \sim \chi^2_\nu$$

    and find the subset of predictors corresponding to the model that maximizes the posterior model probability.

Basic Local Search

- It is an iterative procedure that updates a current candidate solution $\theta^{(t)}$ at iteration t to $\theta^{(t+1)}$.
- One or more possible moves (updates) are identified from a neighborhood of $\theta^{(t)}$, $\mathcal{N}(\theta^{(t)})$.
- A neighborhood of the current candidate solution, $\mathcal{N}(\theta^{(t)})$, contains candidate solutions that are near $\theta^{(t)}$.

Basic Local Search

- The advantage of local search over global search is that only a tiny portion of Θ need be searched at any iteration, and large portions of θ may never be examined.
- The disadvantage is that the search is likely to terminate at an uncompetitive local maximum.
- If the neighborhood is defined by allowing as many as k changes to the current candidate solution in order to produce the next candidate, then it is a k-neighborhood.

Local Search - Ascent Algorithm

- *Steepest Ascent*: An obvious strategy at each iteration is to choose the best among all candidates in the current neighborhood.
- *Random Ascent*: To speed performance, one might instead select the first randomly chosen neighbor for which the objective function exceeds its previous value.
- If k-neighborhoods are used for a steepest ascent algorithm, the solution is said to *k-optimal*.
- Although the ascent is not the steepest possible within $\mathcal{N}(\theta^{(t)})$, any local search algorithm that choose $\theta^{(t+1)}$ uphill from $\theta^{(t)}$ is an *ascent algorithm*.

Local Search - Local Optimal Values

- The sequential selection of steps that are optimal in small neighborhoods, disregarding the global problem, is reminiscent of a *greedy algorithm*.
- Wise selection of a new candidate solution from a neighborhood of the current candidate must balance the need for a narrow focus enabling quick moves against the need to find a globally competitive solution.
- To avoid entrapment in poor local maxima, it might be reasonable to eschew some of the best neighbors of $\theta^{(t)}$ in favor of a direction whose rewards are later realized.

Local Search - Variable Depth Local Search

- When k is greater than 1 or 2, searching within the current neighborhood for a k-change steepest ascent move can be difficult because the size of the neighborhood increases rapidly with k.
- For larger k, it can be useful to break the k-change up into smaller parts, sequentially selecting the best candidate solutions in smaller neighborhoods.
- This *variable-depth local search* permit a potentially better step away from the current candidate solution, even though it will not likely be optimal within k-neighborhood.
- *Random starts local search* run a simple ascent algorithm repeatedly with a large number of starting points.

Example: Baseball Salaries

- Application of the random starts local search method to a regression model selection problem.
- There are 27 baseball performance statistics, which were collected for 337 players in 1991 and players' 1992 salaries may be related to these variables. Use the log of the salary variable as the response variable.
- Find the best subset of predictors to predict log salary using a log linear regression model. There are $2^{27} = 134,217,728$ possible models.

Example: Baseball Salaries

- The application of a random starts local search algorithm to minimize the AIC w.r.t. regression variable selection.
- The problem can be posed as maximizing the negative of the AIC, thus preserving our preference for uphill search.
- Neighborhoods were limited to 1-changes generated from the current model by either adding or deleting one predictor.
- Search was started from 5 randomly selected subsets of predictors and 14 additional steps were allocated to each start.

Introduction to Simulated Annealing

- Simulated annealing is a popular technique for combinatorial optimization
- Advantage: Simulated annealing is generic and easily implemented in its simplest form and this limiting behavior is well studied.
- Disadvantage: The limiting behavior of a simulated annealing is not easily realized in practice, the speed of convergence can be slow.

Introduction to Simulated Annealing

- Annealing is the process of heating up a solid and then cooling it slowly.
- When a stressed solid is heated, its internal energy increases and its molecules move randomly.
- If the solid is then cooled slowly, the thermal energy generally decreases solwly, but there are also random increases governed by Boltzmann's probability.
- At temperature τ, the probability density of an increase in energy of magnitude ΔE is $\exp\{-\Delta E/k\tau\}$ where k is Boltzmann's constant.
- If the cooling is slow enough and deep enough, the final state is unstressed, where all the molecules are arranged to have minimal potential energy.

Simulated Annealing Algorithm

- For consistency with the motivating physical process, we pose optimization as minimization, so the minimum of $f(\theta)$ is sought over $\theta \in \Theta$. (If we set $f(\theta)$ as $-\log L(\theta)$, then the problem will change finding a minimum.)
- Possible to draw an analogy between the physical cooling process and the process of solving a combinatorial minimization problem.
- For simulated annealing algorithms, θ corresponds to the state of the material, $f(\theta)$ corresponds to its energy level, and the optimal solution corresponds to θ that has minimum energy.
- Random changes to the current state (i.e., moves from $\theta^{(t)}$ to $\theta^{(t+1)}$) are governed by the Boltzmann distribution, which depends on parameter called temperature.

Simulated Annealing Algorithm

- When the temperature is high, acceptance of uphill moves are more likely to be tolerated. This discourages convergence to the first local minimum that happens to be found, which might be premature if the space of candidate solution has not been adequately explored.
- As search continues, the temperature is lowered. This forces increasingly concentrated search effort near the current local minimum, because few uphill moves will be allowed.
- If the cooling schedule is determined appropriately, the algorithm will hopefully converge to the global minimum.

Simulated Annealing Algorithm

An iterative procedure started at time $t = 0$ with an initial point $\theta^{(0)}$ and a temperature τ_0. Iterations are indexed by t. The algorithm is run in stages, which we index by $j = 0, 1, 2, \cdots$, and each stage consists of several iterations. The length of the j-th stage is m_j. Each iteration proceeds as follows:

1. Select a candidate solution θ^* within the neighborhood of $\theta^{(t)}$, say $\mathcal{N}(\theta^{(t)})$, according to a proposal density $g(\cdot|\theta^{(t)})$.
2. Randomly decide whether to adopt θ^* as the next candidate solution or to keep another copy of the current solution. Specifically, let $\theta^{(t+1)} = \theta^*$ with probability equal to $\min\left(1, \exp\left[\left\{f\left(\theta^{(t)}\right) - f\left(\theta^*\right)\right\}/\tau_j\right]\right)$. Otherwise, let $\theta^{(t+1)} = \theta^{(t)}$.
3. Repeat steps 1 and 2 a total of m_j times.
4. Increment j. Update $\tau_j = \alpha(\tau_{j-1})$ and $m_j = \beta(m_{j-1})$. Go to step 1.

Simulated Annealing Algorithm

- If the algorithm is not stopped according to a limit on the total number of iterations or a predetermined schedule of τ_j and m_j, one can monitor an absolute or relative convergence criterion.
- After stopping, the best candidate solution found is the estimated minimum.
- The function α should slowly decrease the temperature to zero.
- The number of iterations at each temperature (m_j) should be large and increasing in j.
- Ideally, the function β should scale the m_j exponentially in p, but in practice some compromises will be required in order to tolerable computing speed.

Example: Baseball Salaries

- To implement simulated annealing for variable selection, we establish a neighborhood structure, a proposal distribution, and a temperature.
- The simplest neighborhoods contains 1-change neighbors generated from the current model by adding or deleting one predictor. We assigned equal probabilities to all candidates in a neighborhood.
- The cooling schedule had 15 stages, with lengths of 60 for the first 5 stages, 120 for the next 5, and 220 for the final 5.
- Temperatures were decreased according to $\alpha(\tau_{j-1}) = 0.9\tau_{j-1}$ after each stage.

Example: Baseball Salaries

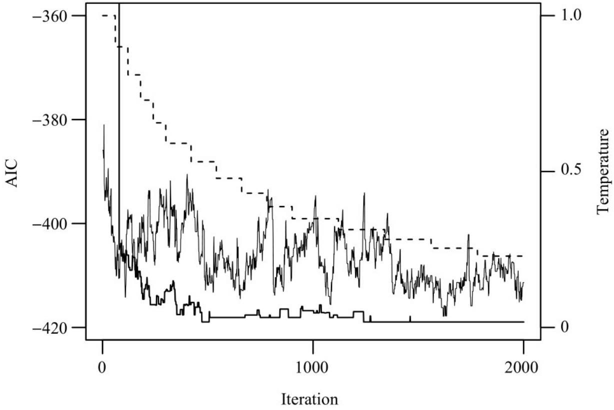

FIGURE 3.4 Results of two simulated annealing minimizations of the regression model AIC for Example 3.4. The temperature for the bottom curve is shown by the dotted line and the right axis. Only AIC values between −360 and −420 are shown.

Overview

- Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA.
- Genetic algorithms are based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals using the following foundations:
    - Individuals in a population compete for resources and mates.
    - Those individuals most successful in each ‘competition’ will produce more offspring than those individuals that perform poorly.
    - Genes from ‘good’ individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
    - Thus each successive generation will become more suited to their environment.

Search Space

- A population of individual is maintained within search space for a genetic algorithm, each representing a possible solution to a given problem.
- Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet $\{0, 1\}$.
- To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables).
- A fitness score is assigned to each solution representing the abilities of an individual to 'compete'. The individual with the optimal (or generally near optimal) fitness score is sought.

Search Space

- The genetic algorithm aims to use selective ‘breeding’ of the solutions to produce ‘offspring’ better than the parents by combining information from the chromosomes.
- The genetic algorithm maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan.
- Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent.
- As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted.

Implementation Details

After an initial population is randomly generated, the algorithm evolves the through three operators:

1. *selection* which equates to survival of the fittest;
2. *crossover* which represents mating between individuals;
3. *mutation* which introduces random modifications.

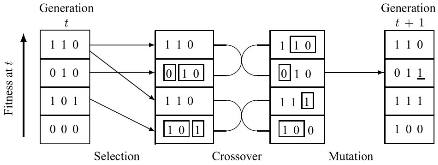

FIGURE 3.5 An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Selection Operator

- Key Idea: Give preference to better individuals, allowing them to pass on their genes to the next generation.
- The goodness of each individual depends on its fitness.
- Fitness may be determined by a function $f(\theta)$.

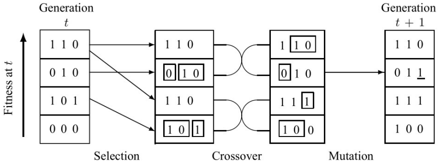

FIGURE 3.5 An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Crossover Operator

- Prime distinguished factor of genetic algorithm from other optimization techniques.
- Two individuals are chosen from the population using the selection operator.
- A crossover site along the bit strings is randomly chosen.
- The values of the two strings are exchanged up to this point.

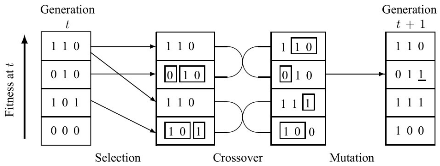

FIGURE 3.5 An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Crossover Operator

- If $S_1 = 000000$ and $S_2 = 111111$ and the crossover point is 2 then $S_1' = 110000$ and $S2' = 001111$.
- The two new offspring created from this mating are put into the next generation of the population.
- By recombining portions of good individuals, this process is likely to create even better individuals.

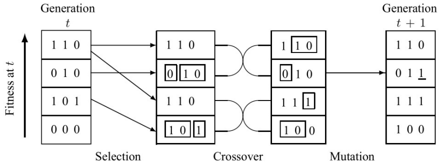

FIGURE 3.5 An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Mutation Operator

- With some low probability, a portion of the new individuals will have some of their bits flipped.
- Its purpose is to maintain diversity within the population and inhibit premature convergence.
- Mutation alone induces a random walk through the search space.
- Mutation and selection (without crossover) create a parallel, noise-tolerant, hill-climbing algorithms.

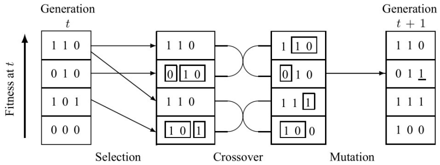

FIGURE 3.5 An example of generation production in a genetic algorithm for a population of size $P = 4$ with chromosomes of length $C = 3$. Crossovers are illustrated by boxing portions of some chromosomes. Mutation is indicated by an underlined gene in the final column.

Effect of Genetic Operator

- Using selection alone will tend to fill the population with copies of the best individual from the population.
- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution.
- Using mutation alone induces a random walk through the search space.
- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm.

Algorithm

1. Randomly initialized population
2. Determine fitness of population.
3. Repeat
    1. Select parents from population.
    2. Perform crossover on parents creating population.
    3. Perform mutation of population.
    4. Determine fitness of population.

Example: Baseball Salaries

- One hundred generations of size $P = 20$ were used.
- Binary inclusion-exclusion alleles were used for each possible predictor yielding chromosomes of length $C = 27$.
- The starting generation consisted of purely random individuals.
- A rank-based fitness function was used. One parent was selected with probability proportional to this fitness.
- The other parent was selected independently, purely at random.
- Breeding employed simple crossover.
- A 1% mutation rate was randomly applied independently to each locus.

Example: Baseball Salaries

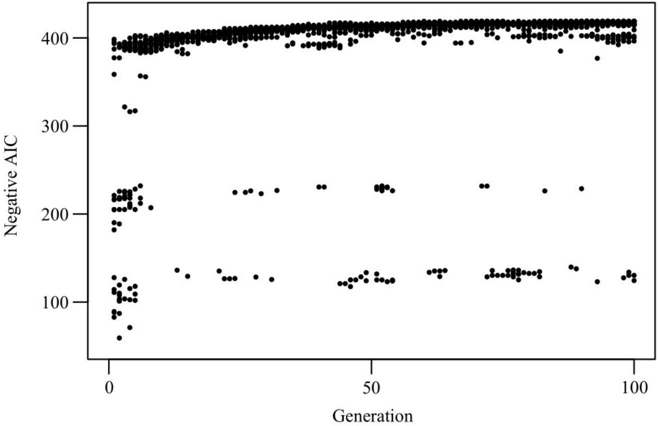

FIGURE 3.6 Results of a genetic algorithm for Example 3.5.