# Heuristic Search

Lucia Moura

Winter 2021

# Heuristic Search vs Exhaustive Search

## Exhaustive Search

- Backtracking (backtracking with bounding):
  - Find all feasible solutions.
  - Find one optimal solution.
  - Find all optimal solutions.
- Branch-and-Bound:
  - Find one optimal solution.

## Heuristic Search

Types of problem it can be applied to:

- Find 1 optimal solution (when optimum value is known)
- Find a “close to” optimal solution (the best solution we manage).

Heuristics methods we will study:

- Hill-climbing, Simulated annealing, Tabu search, Genetic algorithms.

# Characteristics of heuristic search

- The state space is not fully explored.
- Randomization is often employed.
- There is a concept of neighbourhood search.
- **Heuristics** are applied to explore the solutions.
  The word "heuristics" means "serving or helping to find or discover" or "proceeding by trial and error".

# A general framework for heuristic search

**Generic Optimization Problem (maximization):**

Instance: A finite set $\mathcal{X}$.
an objective function $P : \mathcal{X} \to Z$.
$m$ feasibility functions $g_j : \mathcal{X} \to Z$, $1 \leq j \leq m$.
Find: the maximum value of $P(X)$
subject to $X \in \mathcal{X}$ and $g_j(X) \geq 0$, for $1 \leq j \leq m$.

Exercise: pick your favorite combinatorial optimization problem and write it in this framework.

# A general framework for heuristic search (cont'd)

**Designing a heuristic search:**

1. Define a **neighbourhood function** $N : \mathcal{X} \to 2^{\mathcal{X}}$.
   E.g. $N(X) = \{X_1, X_2, X_3, X_4, X_5\}$.
2. Design a **neighbourhood search**:
   Algorithm that finds a feasible solution on the neighbourhood of a feasible solution $X$.
   There are two types of neghbourhood searches:
   - Exhaustive (chooses best profit among neighbour points)
   - Randomized (picks a random point among the neighbour points)

# Defining a neighbourhood function

$$N : \mathcal{X} \to 2^{\mathcal{X}}.$$

So, $N(X)$ is a subset of $\mathcal{X}$.

- $N(X)$ should contain elements that are similar or "close to" $X$.

# Defining a neighbourhood function

$$N : \mathcal{X} \rightarrow 2^{\mathcal{X}}.$$

So, $N(X)$ is a subset of $\mathcal{X}$.

- $N(X)$ should contain elements that are similar or "close to" $X$.
- $N(X)$ may contain infeasible elements of $\mathcal{X}$.

# Defining a neighbourhood function

$$N : \mathcal{X} \to 2^{\mathcal{X}}.$$

So, $N(X)$ is a subset of $\mathcal{X}$.

- $N(X)$ should contain elements that are similar or "close to" $X$.
- $N(X)$ may contain infeasible elements of $\mathcal{X}$.
- In order to be useful, we would like to be able to get to $X_{opt}$ from $X_0$ via a number of applications of $N(\cdot)$.

# Defining a neighbourhood function

$$N : \mathcal{X} \to 2^{\mathcal{X}}.$$

So, $N(X)$ is a subset of $\mathcal{X}$.

- $N(X)$ should contain elements that are similar or "close to" $X$.
- $N(X)$ may contain infeasible elements of $\mathcal{X}$.
- In order to be useful, we would like to be able to get to $X_{opt}$ from $X_0$ via a number of applications of $N(\cdot)$.
- I.E. the graph $G$ with $V(G) = \mathcal{X}$ and $E(G) = \{\{X, Y\} : Y \in N(X)\}$ should ideally be connected, or at least have one optimal solution in each of its connected components.

# Defining a neighbourhood function

$$N : \mathcal{X} \to 2^{\mathcal{X}}.$$

So, $N(X)$ is a subset of $\mathcal{X}$.

- $N(X)$ should contain elements that are similar or "close to" $X$.
- $N(X)$ may contain infeasible elements of $\mathcal{X}$.
- In order to be useful, we would like to be able to get to $X_{opt}$ from $X_0$ via a number of applications of $N(\cdot)$.
- I.E. the graph $G$ with $V(G) = \mathcal{X}$ and $E(G) = \{\{X, Y\} : Y \in N(X)\}$ should ideally be connected, or at least have one optimal solution in each of its connected components.
- Computing $N(X)$ should be fast, and in particular $|N(X)|$ shouldn't be too large.

# Examples of neighbourhood functions

First, define $dist(X,Y)$ for $X, Y \in \mathcal{X}$.
Let $d_0$ be a constant positive integer.
We can define a neighbourhood function as follows:

$$N_{d_0}(X) = \{Y \in \mathcal{X} : dist(X,Y) \leq d_0\}.$$

# Examples of neighbourhood functions based on distances

- $\mathcal{X} = \{0,1\}^n$, set of all binary $n$-tuples.
  Here $dist$ is the Hamming distance.
  $N_1([010]) = \{[000],[110],[011],[010]\}$.

$$|N_{d_0}(X)| = \sum_{i=0}^{d_0} \binom{n}{i}.$$

# Examples of neighbourhood functions based on distances

- $\mathcal{X} = \{0,1\}^n$, set of all binary $n$-tuples.
  Here $dist$ is the Hamming distance.
  $N_1([010]) = \{[000],[110],[011],[010]\}$.

$$|N_{d_0}(X)| = \sum_{i=0}^{d_0} \binom{n}{i}.$$

- $\mathcal{X} =$ set of all permutations of $\{1,2,\ldots,n\}$.
  Let $\alpha = [\alpha_1,\ldots,\alpha_n]$ and $\beta = [\beta_1,\ldots,\beta_n]$ be two permutations.
  Define distance as follows: $dist(\alpha,\beta) = |\{i : \alpha_i \neq \beta_i\}|$.
  Note that $N_1(X) = \{X\}$ is not very useful; we need $d_0 > 1$.

$$N_2([1,2,3,4]) = \{[1,2,3,4],[2,1,3,4],[3,2,1,4],$$
$$[4,2,3,1],[1,3,2,4],[1,4,3,2],[1,2,4,3]\}$$

  $|N_2(X)| = 1 + \binom{n}{2}$.

# Neighbourhoods and Local Search

Video: Coursera Discrete Optimization, Pascal Van Hentenryck, LS1: intuition n-queens
`https://www.coursera.org/lecture/discrete-optimization/ls-1-intuition-n-queens-ZRJeF`

# Designing a neighbourhood search algorithm

Input: $X$

Output: $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible, or "fail".

Possible Neighbourhood Search Strategies:

1. Find a feasible $Y \in N(X) \setminus \{X\}$ such that $P(Y)$ is maximized. Return "fail" if there is no feasible solution in $N(X) \setminus \{X\}$.
2. Find a feasible $Y \in N(X) \setminus \{X\}$ such that $P(Y)$ is maximized. if $P(Y) > P(X)$ then return $Y$; else return "fail". (steepest ascent method)
3. Find any feasible $Y \in N(X) \setminus \{X\}$. Return "fail" if there is no feasible solution in $N(X) \setminus \{X\}$.
4. Find any feasible $Y \in N(X) \setminus \{X\}$. if $P(Y) > P(X)$ then return $Y$; else return "fail".

Strategies 1 and 2 may be exhaustive.

Strateges 3 and 4 are usually randomized.

# A generic heuristic search algorithm

Given $N$, a neighbourhood function, the heuristic algorithm $h_N$ either:

- Perform one neighbourhood search (using one of the strategies)
- Perform a sequence of $j$ neighbourhood searches, where each one takes us from $X_i$ to $X_{i+1}$: $[X = X_0, X_1, \ldots, X_j = Y]$.

Algorithm $\textsc{GenericHeuristicSearch}(c_{max})$

Select a feasible solution $X \in \mathcal{X}$;
$X_{best} \leftarrow X$; (stores best so far); $c \leftarrow 0$;
while $(c \leq c_{max})$ do
  $Y \leftarrow h_N(X)$;
  if $(Y \neq$ "fail") then $X \leftarrow Y$;
    if $(P(X) > P(X_{best}))$ then $X_{best} \leftarrow X$;
  [else $c \leftarrow c_{max} + 1$; (add this if $h_N$ is not randomized)]
  $c \leftarrow c + 1$;
return $X_{best}$;

# More examples on neighbourhood search: Swap Neighbourhood

- Video: Coursera Discrete Optimization, Pascal Van Hentenryck, LS2: (see swap neightbouhood and then magic square start at minute 8:13) `https://www.coursera.org/lecture/discrete-optimization/ls-2-swap-neighborhood-car-sequencing-magic-square-2vaaM`
- Video: Coursera Discrete Optimization, Pascal Van Hentenryck, LS3: (local search for optimization: TSP/2-OPT see minute 5:37) `https://www.coursera.org/lecture/discrete-optimization/ls-3-optimization-warehouse-location-traveling-salesman-2`

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.
- Problem: it can get stuck in a local optimum.

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.
- Problem: it can get stuck in a local optimum.
- Improvement: run the algorithm many times from different random starting points $X$.

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.
- Problem: it can get stuck in a local optimum.
- Improvement: run the algorithm many times from different random starting points $X$.
- For Hill-Climbing, $h_N(X)$ returns:

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.
- Problem: it can get stuck in a local optimum.
- Improvement: run the algorithm many times from different random starting points $X$.
- For Hill-Climbing, $h_N(X)$ returns:
  - $Y \in N(X)$ such that $Y$ is feasible and $P(Y) > P(X)$,

# Hill-Climbing

- Idea: Go up the hill continuously, stop when stuck.
- Problem: it can get stuck in a local optimum.
- Improvement: run the algorithm many times from different random starting points $X$.
- For Hill-Climbing, $h_N(X)$ returns:
  - $Y \in N(X)$ such that $Y$ is feasible and $P(Y) > P(X)$,
  - or, otherwise, "fail".

# Hill Climbing Algorithm

Algorithm $\textsc{GenericHillClimbing}()$

  Select a feasible solution $X \in \mathcal{X}$.
  $X_{best} \leftarrow X$; $searching \leftarrow true$;
  while ($searching$) do
    $Y \leftarrow h_N(X)$;
    if ($Y \neq$ "fail") then
      $X \leftarrow Y$;
      if $(P(X) > P(X_{best}))$ then $X_{best} \leftarrow X$;
    else $searching \leftarrow false$;
  return $X_{best}$;

Hill-climbing can get trapped in a local optimum.
Other search strategies (**simulated annealing**,**tabu search**) try to escape from local optima.

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;
  - later $T \leftarrow \alpha T$ for a fixed $0 < \alpha < 1$.

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;
  - later $T \leftarrow \alpha T$ for a fixed $0 < \alpha < 1$.
- Going uphill is always accepted.

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;
  - later $T \leftarrow \alpha T$ for a fixed $0 < \alpha < 1$.
- Going uphill is always accepted.
- Going downhill is sometimes accepted with a probability based on how much downhill we go and on the current temperature.

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;
  - later $T \leftarrow \alpha T$ for a fixed $0 < \alpha < 1$.
- Going uphill is always accepted.
- Going downhill is sometimes accepted with a probability based on how much downhill we go and on the current temperature.
  - Given $Y = h_N(X)$ with $P(Y) \leq P(X)$,

# Simulated Annealing

- Analogy with a method of cooling metal: annealing.
  - Temperature $T$ decreases at each iteration, according to a **cooling schedule** $(T_0, \alpha)$:
  - Initally $T \leftarrow T_0$;
  - later $T \leftarrow \alpha T$ for a fixed $0 < \alpha < 1$.
- Going uphill is always accepted.
- Going downhill is sometimes accepted with a probability based on how much downhill we go and on the current temperature.
  - Given $Y = h_N(X)$ with $P(Y) \leq P(X)$,
  - accept $Y$ with probability

$$e^{(P(Y)-P(X))/T} = \frac{1}{e^{(P(X)-P(Y))/T}}$$

(We get pickier as we progress, since $T$ decreases)

# Simulated Annealing Algorithm

Algorithm $\textsc{GenericSimulatedAnnealing}(c_{max}, T_0, \alpha)$

$c \leftarrow 0$; $T \leftarrow T_0$;
Select a feasible solution $X \in \mathcal{X}$; $X_{best} \leftarrow X$;
while $(c \leq c_{max})$ do
  $Y \leftarrow h_N(X)$; // this is usually a randomized choice
  if $(Y \neq$ "fail"$)$ then
    if $(P(Y) > P(X))$ then
      $X \leftarrow Y$;
      if $(P(X) > P(X_{best}))$ then $X_{best} \leftarrow X$;
    else $r \leftarrow random(0, 1)$;
      if $(r < e^{\frac{P(Y)-P(X)}{T}})$ then $X \leftarrow Y$;
  $c \leftarrow c + 1$;
  $T \leftarrow \alpha T$;
return $X_{best}$;

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).
- What may be the risk?

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).
- What may be the risk?
  - Cycling.

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).
- What may be the risk?
  - Cycling.
  - When going downhill from $X$ to $Y$ we may go back from $X$ to $Y$.

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).
- What may be the risk?
  - Cycling.
  - When going downhill from $X$ to $Y$ we may go back from $X$ to $Y$.
  - Cycling may also take several steps, such as $X \to Y \to Z \to X$.

# Tabu Search

- Neighbourhood search:
  Choose $Y \in N(X) \setminus \{X\}$ such that $Y$ is feasible and $P(Y)$ is maximum among all such elements (exhaustive neighbourhood search).
  It may happen that $P(Y) < P(X)$ (we escape from a local optimum).
- What may be the risk?
  - Cycling.
  - When going downhill from $X$ to $Y$ we may go back from $X$ to $Y$.
  - Cycling may also take several steps, such as $X \to Y \to Z \to X$.
- Tabu-search uses a strategy for avoiding cycling: a **tabu list**.
  After a move $X \to Y$,
  we forbit the application of $\textsc{change}(Y, X)$ for $L$ iterations
  ($L$ is the lifetime of the tabu list).

# Tabu List

- After a move $X \to Y$, we keep $\textsc{change}(Y, X)$ t the Tabu List for $L$ iterations.

# Tabu List

- After a move $X \to Y$, we keep $\textsc{change}(Y, X)$ t the Tabu List for $L$ iterations.
- Example:
  $\mathcal{X} = \{0, 1\}^n$, using $N_1(X) = \{Y \in \mathcal{X} : dist(X, Y) = 1\}$.
  $X = [0100]$ and $Y = [0101]$, we have that $\textsc{change}(Y, X) = 4 =$ index of coordinate that was swapped.
  Suppose $L = 2$.

| | | | | | |
|---|---|---|---|---|---|
| sequence of points: | $[0100]$ | $[0101]$ | $[1101]$ | $[1001]$ | $[1011]$ |
| tabu list: | | 4 | 4,1 | 1,2 | 2,3 |

# Tabu List

- After a move $X \to Y$, we keep $\text{CHANGE}(Y, X)$ t the Tabu List for $L$ iterations.
- Example:
  $\mathcal{X} = \{0,1\}^n$, using $N_1(X) = \{Y \in \mathcal{X} : dist(X,Y) = 1\}$.
  $X = [0100]$ and $Y = [0101]$, we have that $\text{CHANGE}(Y, X) = 4 =$ index of coordinate that was swapped.
  Suppose $L = 2$.

| | | | | | |
|---|---|---|---|---|---|
| sequence of points: | $[0100]$ | $[0101]$ | $[1101]$ | $[1001]$ | $[1011]$ |
| tabu list: | | 4 | 4,1 | 1,2 | 2,3 |

- So any sequence that cycles $X \to \ldots \to X$ has length at least $2L$. Choosing $L = 10$ is typical.

$\textsc{TabuList}$ is defined below to be a list where $\textsc{TabuList}[c] = \delta$, where $\delta$ is the designated forbidden (tabu) change at iteration $c$.

For tabu search, $h_N(X) = Y$, where

- $Y \in N(X)$, $Y$ is feasible;
- $\textsc{change}(X, Y) \notin \{ \textsc{TabuList}[d] : c - L \leq d \leq c - 1\}$;
- $P(Y)$ is maximum among all such feasible elements.

In absolute no circumstance implement $\textsc{TabuList}$ as an array indexed by the number of iterations! Instead, implement $\textsc{TabuList}$ as a queue of length $L$. Note that the algorithm may mislead you to think you are using such an array, given the notation defined above; careful!

# Tabu Search Algorithm: textbook version/typo correction

Algorithm $\textsc{GenericTabuSearch}(c_{max}, L)$

$c \leftarrow 1$;
Select a feasible solution $X \in \mathcal{X}$.
$X_{best} \leftarrow X$;
while $(c \leq c_{max})$ do
  $N \leftarrow N(X) \setminus \{F : \textsc{change}(X, F) \in \textsc{TabuList}[d], c - L \leq d \leq c - 1\}$;
  for each $(Y \in N)$ do if ($Y$ is infeasible) then $N \leftarrow N \setminus \{Y\}$;
  if $(N = \emptyset)$ then return $X_{best}$;
  Find $Y \in N$ such that $P(Y)$ is maximum; /* computes $Y = h_N(X)$ */
  $\textsc{TabuList}[c] \leftarrow \textsc{change}(Y, X)$;
  $X \leftarrow Y$;
  if $(P(X) > P(X_{best}))$ then $X_{best} \leftarrow X$;
  $c \leftarrow c + 1$;
return $X_{best}$;

# Tabu List Implementation

In absolute no circumstance implement $\textsc{TabuList}$ as an array indexed by the number of iterations!
In the real implementation, $\textsc{Tabulist}$ can be a queue of length $L$!!!
So, the operation
$\textsc{Tabulist}[c] \leftarrow \textsc{change}(Y, X)$;
must be implemented as:
$\textsc{Tabulist}$.insert($\textsc{change}(Y, X)$); (only keeps last $L$ elements)
and the line: $N \leftarrow N(X) \setminus \{F : \textsc{change}(X, F) \in \textsc{Tabulist}[d], c - L \leq d \leq c - 1\}$
should be understood as:
$N \leftarrow N(X) \setminus \{F : \textsc{change}(X, F)$ is in $\textsc{Tabulist}\}$;

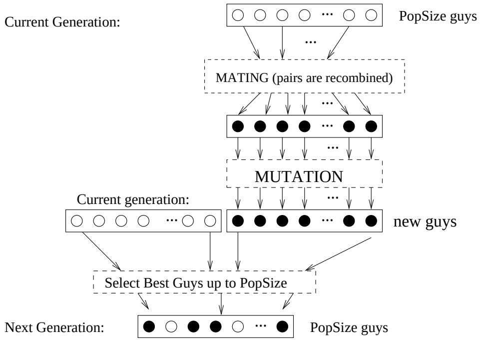
# Tabu Search Algorithm: with FIFO queue for TABULIST

Algorithm $\textsc{GenericTabuSearch}(c_{max}, L)$
  $c \leftarrow 1$;
  Select a feasible solution $X \in \mathcal{X}$.
  $X_{best} \leftarrow X$;
  while $(c \leq c_{max})$ do
    $N \leftarrow N(X) \setminus \{F : \textsc{change}(X, F)$ is in $\textsc{Tabulist}\}$
    for each $(Y \in N)$ do   if ($Y$ is infeasible) then $N \leftarrow N \setminus \{Y\}$;
    if $(N = \emptyset)$ then return $X_{best}$;
    Find $Y \in N$ such that $P(Y)$ is maximum; /* computes $Y = h_N(X)$ */
    $\textsc{Tabulist}$.insert($\textsc{change}(Y, X), L$); /* only keeps last $L$ entries */
    $X \leftarrow Y$;
    if $(P(X) > P(X_{best}))$ then $X_{best} \leftarrow X$;
    $c \leftarrow c + 1$;
  return $X_{best}$;

# Genetic Algorithms

Fix a number PopSize (population size).

One iteration works as follows:



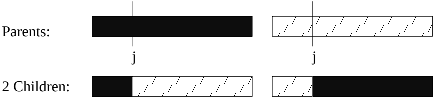

Iterate as many generations as you like.

# Mating Strategies (Recombination)

Producing children from parents.

**Method 1: Crossover.**

Let $j$ be a crossover point.




Example: $j = 3$

| | | |
|---|---|---|
| Parents: | $[110\|1101001]$ | $[100\|1000101]$ |
| Children: | $[110\|1000101]$ | $[100\|1101001]$ |

# Mating Strategies (Recombination), cont'd

**Method 2: Partially matched crossover (for permutations)**

Two crossover points: $1 \leq j < k \leq n$
Example: $j = 3$ and $k = 6$
$\alpha = [3, 1, \underline{4, 7, 6, 5}, 2, 8]$ $\beta = [8, 6, \underline{4, 3, 7, 1}, 2, 5]$

| swap | $\alpha$ | $\beta$ |
|---|---|---|
| $4 \leftrightarrow 4$ | $[3, 1, 4, 7, 6, 5, 2, 8]$ | $[8, 6, 4, 3, 7, 1, 2, 5]$ |
| $7 \leftrightarrow 3$ | $[7, 1, 4, 3, 6, 5, 2, 8]$ | $[8, 6, 4, 7, 3, 1, 2, 5]$ |
| $6 \leftrightarrow 7$ | $[6, 1, 4, 3, 7, 5, 2, 8]$ | $[8, 7, 4, 6, 3, 1, 2, 5]$ |
| $5 \leftrightarrow 1$ | $[6, 5, 4, 3, 7, 1, 2, 8]$ | $[8, 7, 4, 6, 3, 5, 2, 1]$ |

# Mating Schemes

Kids may be infeasible: incorporate constraints as penalties.

Various methods are possible for mating schemes:

1. Random monogamy with 2 kids per couple: randomly partition population into pairs, with two kids produced by each pair.
2. Make better parents having more kids:
   measure parent fitness by objective function; parents with higher fitness produce more kids.

Algorithm $\textsc{GenericGeneticAlgorithm}(PopSize, c_{max})$

```
Select an initial population 𝒫 with PopSize feasible solutions;
for each X ∈ 𝒫 do X ← h_N(X);    [mutation]
X_best ← element in 𝒫 with maximum profit; c ← 1;
while (c ≤ c_max) do
    𝒬 ← 𝒫; Construct a pairing of the elements in 𝒫;
    for each pair (W, X) in the pairing do
        (Y, Z) ← rec(W, X);    [recombination/mating]
        Y ← h_N(Y); Z ← h_N(Z);    [mutations]
        𝒬 ← 𝒬 ∪ {Y, Z};
    Set 𝒫 to be the best PopSize members of 𝒬;
    Let Y be the element in 𝒫 with maximum profit;
    if (P(Y) > P(X_best)) then X_best ← Y;
    c ← c + 1;
return X_best;
```

## Steepest Ascent for Uniform Graph Partition

PROBLEM: UNIFORM GRAPH PARTITION
INSTANCE: A COMPLETE GRAPH ON $2n$ VERTICES,
$cost : E \to Z^+ \cup \{0\}$ (COST FUNCTION)
FIND: THE MINIMUM VALUE OF
$C([X_0, X_1]) = \sum_{u \in X_0, v \in X_1} cost(u, v)$
SUBJECT TO $V = X_0 \cup X_1$, $|X_0| = |X_1| = n$.

Example: $n = 4$; $cost(1,2) = 1, cost(1,3) = 2, cost(1,4) = 5, cost(2,3) = 0, cost(2,4) = 5, cost(3,4) = 1$.
Only 3 feasible solutions (except for exchanging $X_0$ and $X_1$):

$$X_0 = \{1,2\}, \quad X_1 = \{3,4\}, \qquad C([X_0, X_1]) = 12$$
$$X_0 = \{1,3\}, \quad X_1 = \{2,4\}, \qquad C([X_0, X_1]) = 7 \quad (optimal)$$
$$X_0 = \{1,4\}, \quad X_1 = \{2,3\}, \qquad C([X_0, X_1]) = 9$$

# Uniform Graph Partition: Steepest Ascend Algorithm

Neighbourhood function: exchange $x \in X_0$ and $y \in X_1$.

Algorithm UGP($C_{max}$)
    $X = [X_0, X_1] \leftarrow$ SelectRandomPartition
    $c \leftarrow 1$
    while ($c \leq C_{max}$) do
        $[Y_0, Y_1] \leftarrow$ Ascend($X$)
        if not fail then
                $\{X_0 \leftarrow Y_0;\ \ X_1 \leftarrow Y_1;\}$
        else return
        $c \leftarrow c + 1$

# Ascend Algorithm

```
Algorithm Ascend([X_0, X_1])
    g ← 0
    for each i ∈ X_0 do
        for each j ∈ X_1 do
            t ← G_[X_0,X_1](i, j)   (gain obtained in exchange)
            if (t > g) then {x ← i;  y ← j;  g ← t}
    if (g > 0) then
        Y_0 ← (X_0 ∪ {y}) \ {x}
        Y_0 ← (X_1 ∪ {x}) \ {y}
        fail ← false
        return [Y_0, Y_1]
    else {fail ← true; return [X_0, X_1]}
```

# SelectRandomPartition

Two possible algorithms:

1. Picking $X_0$ as a random $n$-subset $r$ of a $2n$-set:
   Get a random integer $r \in [0, \binom{2n}{n} - 1]$ and apply `kSubsetLexUnrank`$(r, n, 2n)$.
2. Randomly shufling elements in $[0, 2n-1]$:
   Create array $A[0, 2n-1]$ with randomly chosen numbers as elements.
   Create array $B[0, 2n-1]$ initially with $B[i] = i$.
   Sort $A$, doing same swaps on $B$.
   Take $X_0$ as the first half of $B$, and $X_1$ as the second half.

# Hill-climbing for Steiner triple systems

Textbook, Section 5.4.

## Definition

A *Steiner triple system* of order $v$, denoted $STS(v)$, is a pair $(V, \mathcal{B})$ where:
$V = \{1, 2, \ldots v\}$ is a set of points,
$\mathcal{B} = \{B_1, B_2, \ldots, B_b\}$ is a set of 3-sets, called blocks, such that any pair of points in $V$ is in a unique block $B_i \in \mathcal{B}$.

**Example:** $STS(9)$:

$$\begin{aligned} V = & \quad \{1,2,3,4,5,6,7,8,9\} \\ \mathcal{B} = & \{ \quad \{1,2,3\},\{1,4,7\},\{1,5,9\},\{1,6,8\},\{4,5,6\},\{2,5,8\}, \\ & \quad\;\; \{2,6,7\},\{2,4,9\},\{7,8,9\},\{3,6,9\},\{3,4,8\},\{3,5,7\}\} \end{aligned}$$

# Replication number and number of blocks

**Lemma**

*Let $(V, \mathcal{B})$ be an $STS(v)$. Then, every point in $V$ occurs in exactly $r = \frac{v-1}{2}$ blocks and $|\mathcal{B}| = \frac{v(v-1)}{6}$.*

PROOF:

1. Any point $x$ must appear in some block with each of all other $(v-1)$ points. Point $x$ occurs with $2$ other points in each of the $r_x$ blocks it appears. Therefore, $r_x = \frac{v-1}{2}$.
2. We count $T$, the number of points with their replications appearing on $\mathcal{B}$, in two ways: $T = 3 \times b$ and $T = v \times r$. Thus, $3 \times b = v \times r$, which implies $b = \frac{v(v-1)}{6}$.

# Necessary and sufficient conditions for existence of $STS(v)$

Since $r = \frac{v-1}{2}$ (point replication number) and $b = \frac{v(v-1)}{6}$ (number of blocks) must be integer numbers, we need $v \equiv 1, 3 \pmod 6$.
These necessary conditions have been proven to be sufficient:

**Theorem**

$$\exists STS(v) \iff v \equiv 1, 3 \pmod 6$$

So, there exists an $STS(v)$ for
$v = 1, 3, 7, 9, 13, 15, 19, 21, 25, 17, 31, 33, \ldots$

A partial Steiner triple system consists of a set of triples $\mathcal{B}$ with each pair of points appearing in at most one $B_i \in \mathcal{B}$. Then, we can formulate the search problem as follows.

# Searching for Steiner Triple Systems

PROBLEM: CONSTRUCT A STEINER TRIPLE SYSTEM
INSTANCE: $v$ SUCH THAT $v \equiv 1, 3 \pmod 6$
FIND: MAXIMIZE $|\mathcal{B}|$
SUBJECT TO: $([1, v], \mathcal{B})$ IS A PARTIAL STEINER TRIPLE SYSTEM

The **universe** $\mathcal{X}$ is the set of all sets of blocks $\mathcal{B}$, such that $([1, v], \mathcal{B})$ is a partial Steiner triple system.

An **optimal solution** is any feasible solution with $|\mathcal{B}| = \frac{v(v-1)}{6}$.

# Stinson's hill-climbing algorithm for STSs

Algorithm Stinson's Algorithm($v$)

  Numblocks $\leftarrow 0$
  $V \leftarrow \{1, 2, \ldots v\}$
  $\mathcal{B} \leftarrow \emptyset$
  While (Numblocks $< \frac{v(v-1)}{2}$) do { $\textsc{Switch}$}
  output $(V, \mathcal{B})$

To present $\textsc{Switch}$, we need:

## Definition

A point $x$ is said to be a **live point** in $([1, v], \mathcal{B})$ if $r_x < \frac{v-1}{2}$.
A pair $\{x, y\}$ is said to be a **live pair** in $([1, v], \mathcal{B})$ if there exists no $B \in \mathcal{B}$ with $\{x, y\} \subseteq B$

# Stinson's hill-climbing for STSs: Switch Algorithm

```
Algorithm SWITCH
    Choose a random live point x.
    Choose random y, z such that
      {x, y} and {x, z} are live pairs.
    If ({y, z} is a live pair) then
       B ← B ∪ {{x, y, z}}
       Numblocks ← Numblocks +1
    else
        Let {w, y, z} ∈ B be the block containing {y, z}
        B ← B ∪ {{x, y, z}} \ {{w, y, z}}
```

See implementation details in the textbook.
Using appropriatte data structures, SWITCH is implemented in constant time.

# Two heuristics for the Knapsack Problem

Knapsack (Optimization) Problem

Instance: Profits $p_0, p_1, \ldots, p_{n-1}$
Weights $w_0, w_1, \ldots, w_{n-1}$
Knapsack capacity $M$

Universe: $\mathcal{X} = \{0,1\}^n$ (set of all $n$-tuples)
an $n$-tuple $[x_0, x_1, \ldots, x_{n-1}]$ is feasible if $\sum_{i=0}^{n-1} w_i x_i \leq M$.

Objective: maximize $P(X) = \sum_{i=0}^{n-1} p_i x_i$.

Algorithm $\text{KNAPSACKSIMULATEDANNEALING}(c_{max}, T_0, \alpha)$

$c \leftarrow 0$; $T \leftarrow T_0$; $X \leftarrow [x_0, x_1, \ldots, x_{n-1}] = [0, 0, \ldots, 0]$;
$CurW \leftarrow 0$; $X_{best} \leftarrow X$;
while $(c \leq c_{max})$ do
  $j \leftarrow$ randomInt$(0, n-1)$; $Y \leftarrow X$; $y_j \leftarrow 1 - x_j$; (using $N_1(X)$)
  if $(y_j = 1)$ and $(curW + w_j > M)$ then $Y \leftarrow fail$;
  if $(Y \neq fail)$ then if $(y_j = 1)$ then
      $X \leftarrow Y$;
      $curW \leftarrow curW + w_j$;
      if $P(X) > P(X_{best})$ then $X_{best} \leftarrow X$;
    else $r \leftarrow$ random$(0, 1)$;
      if $(r < e^{-p_j/T})$ then
        $X \leftarrow Y$; $curW \leftarrow curW - w_j$;
  $c \leftarrow c + 1$; $T \leftarrow \alpha T$;
return $(X_{best})$;

# Knapsack Simulated Annealing Results

**TABLE 5.3**
**Summary data for the knapsack simulated annealing algorithm.**

| $\alpha$ | $c_{max}$ | profits found minimum | profits found maximum | profits found average |
|---|---|---|---|---|
| 0.999 | 1000 | 1441 | 1454 | 1446.8 |
| 0.999 | 5000 | 1448 | 1456 | 1452.1 |
| 0.999 | 20000 | 1448 | 1456 | 1450.9 |
| 0.9995 | 1000 | 1445 | 1455 | 1448.4 |
| 0.9995 | 5000 | 1450 | 1458 | 1454.6 |
| 0.9995 | 20000 | 1452 | 1458 | 1453.9 |
| 0.9999 | 1000 | 1445 | 1455 | 1449.6 |
| 0.9999 | 5000 | 1450 | 1458 | 1454.3 |
| 0.9999 | 20000 | 1453 | 1458 | 1456.1 |

# Tabu Search for Knapsack

We will use the same neighbourhood $N_1(.)$.

Do exhaustive search on the neighbourhood in order to find the best way to update the current solution.

Instead of Profit improvement only, we look for improvements based on the ratio $p_i/w_i$:

1. Chose $i$ with maximum $p_i/w_i$ among the indexes $j$ where $x_j = 0$, $j$ is not on $\textsc{TabuList}$, and changing $x_j$ to 1 does not exceed $M$.
2. If there is no $j$ as above, then choose $i$ with minimum $p_i/w_i$ among the indexes $j$ where $x_j = 1$ and $j$ is not on $\textsc{TabuList}$.

This can be expressed by saying that we want to maximize

$$(-1)^{x_j}\frac{p_j}{w_j}.$$

Algorithm $\textsc{KnapsackTabuSearch}(c_{max}, L)$

Select a random feasible $X = [x_0, x_1, \ldots, x_{n-1}] \in \{0,1\}^n$;
$curW \leftarrow \sum_{i=0}^{n-1} x_i w_i$; $X_{best} \leftarrow X$;
for $(c \leftarrow 1; c \leq c_{max}; c \leftarrow c+1)$ do
  $N \leftarrow \{0, 1, \ldots, n-1\} \setminus \{j : j$ is in $\textsc{TabuList}\}$;
  for each $(i \in N)$ do
    if $(x_i = 0)$ and $(curW + w_i > M)$ then $N \leftarrow N \setminus \{i\}$;
  if $(N = \emptyset)$ then break for-loop;
  Find $i \in N$ such that $(-1)^{x_i} p_i / w_i$ is maximum;
  $\textsc{TabuList.insert}(i, L)$; (removing oldest, if has $L+1$ items)
  $x_i \leftarrow 1 - x_i$;  (swap $i$ coordinate)
  if $(x_i = 1)$ then $curW \leftarrow curW + w_i$;
    else $curW \leftarrow curW - w_i$;
  if $P(X) > P(X_{best})$ then $X_{best} \leftarrow X$;
return $X_{best}$;

## Two heuristics for the Knapsack Problem

**TABLE 5.6**
**Summary data for the knapsack tabu search algorithm (24 items).**

| $L$ | profits found (in 25 runs) | | | # optimal solutions found |
|---|---|---|---|---|
| | minimum | maximum | average | |
| 1 | 13079298 | 13466838 | 13388643.5 | 0 |
| 2 | 13084476 | 13500943 | 13415747.5 | 0 |
| 3 | 13245597 | 13500943 | 13456205.2 | 0 |
| 4 | 13264009 | 13500943 | 13446933.8 | 0 |
| 5 | 13358351 | 13500943 | 13458145.8 | 0 |
| 6 | 13148978 | 13549094 | 13427333.6 | 1 |
| 7 | 13116665 | 13549094 | 13462902.4 | 4 |
| 8 | 13346220 | 13549094 | 13497932.2 | 7 |

# A Genetic Algorithm for the TSP

Traveling Salesman Problem (TSP)

Instance: a complete graph $K_n$
a cost function $c : V \times V \to R$

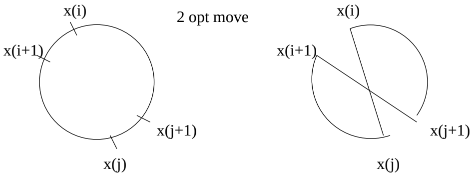
Find: a Hamiltonian circuit $[x_0, x_1, \ldots, x_{n-1}]$ that minimizes
$C(X) = c(x_0, x_1) + c(x_1, x_2) + \ldots + c(x_{n-1}, x_0)$

Note that $2n$ permutations represent the same cycle.
Universe: $\mathcal{X}$ = set of all $n!$ permutations.
Steps:

- Selection of initial population.
- Mutation: steepest ascent $2$-opt.
- Recombination using two methods: partially matched crossover and another method.

# Mutation

Steepest ascent algorithm based on the $2$-opt heuristic:




Gain in applying a $2$-opt move:

$$\begin{aligned} G(X,i,j) &= C(X) - C(X_{ij}) \\ &= c(x_i, x_{i+1}) + c(x_j, x_{j+1}) - c(x_{i+1}, x_{j+1}) - c(x_i, x_j) \end{aligned}$$

$N(X) =$ all $Y \in \mathcal{X}$ that can be obtained from $X$ by a 2-opt move.

Algorithm $\text{SteepestAscentTwoOpt}(X)$

$done \leftarrow false$;
while $(not\ done)$ do
  $done \leftarrow true$; $g_0 \leftarrow 0$;
  for $i \leftarrow 0$ to $n-1$ do
    for $j \leftarrow i+2$ to $n-1$ do
      $g \leftarrow G(X,i,j)$;
      if $(g > g_0)$ then
        $g_0 \leftarrow g$; $i_0 \leftarrow i$; $j_0 \leftarrow j$;
  if $(g_0 > 0)$ then
    $X \leftarrow X_{i_0,j_0}$;
    $done \leftarrow false$;

# Selecting the initial population

Randomly pick one and then mutate it:

Algorithm $\textsc{Select}(popsize)$
for $i \leftarrow 0$ to $popsize - 1$ do
  $r \leftarrow \textsc{RandomInteger}(0, n! - 1)$;
  $P_i \leftarrow \textsc{PermLexUnrank}(n, r)$;
  $\textsc{SteepestAscentTwoOpt}(P_i)$;
return $[P_0, P_1, \ldots, P_{popsize-1}]$;

# Recombination algorithm 1: Partially Matched Crossover

Algorithm $\textsc{PMRec}(A, B)$
$h \leftarrow \textsc{RandomInteger}(10, n/2)$; (length of the substring)
$j \leftarrow \textsc{RandomInteger}(0, n-1)$; (start of the substring)
$(C, D) \leftarrow \textsc{PartiallyMatchedCrossover}(A, B, j, (h+j) mod\ n)$
$\textsc{SteepestAscentTwoOpt}(C)$;
$\textsc{SteepestAscentTwoOpt}(D)$;
return $(C, D)$;

# Recombination Algorithm 2

Algorithm $\text{MGKR}\textsc{ec}(A, B)$

$h \leftarrow \textsc{RandomInteger}(10, n/2)$; (length of the substring)
$j \leftarrow \textsc{RandomInteger}(0, n-1)$; (start of the substring)
$T \leftarrow \emptyset$;
(pick subcycle of length $h$ starting from pos $j$:)
for $i \leftarrow 0$ to $h-1$ do
  $D[i] \leftarrow B[(i+j) \mod n]$;
  $T \leftarrow T \cup \{D[i]\}$;
Complete cycle with permutation in $A$ using guys not already in $D$ in the order prescribed by $A$:
for $j \leftarrow 0$ to $n-1$ do
  if $A[j] \notin T$ then $\{D[i] \leftarrow A[j]; i \leftarrow i+1; \}$
$\textsc{SteepestAscentTwoOpt}(D)$;
(Similarly build $C$ swapping $A$ and $B$ roles:)...

(Algorithm continued)

(Similarly build $C$ swapping $A$ and $B$ roles:)...
$j \leftarrow \textsc{RandomInteger}(0, n-1)$; (start of the substring)
$T \leftarrow \emptyset$;
for $i \leftarrow 0$ to $h-1$ do
  $C[i] \leftarrow A[(i+j) \mod n]$;
  $T \leftarrow T \cup \{C[i]\}$;
for $j \leftarrow 0$ to $n-1$ do
  if $B[j] \notin T$ then $\{C[i] \leftarrow B[j];\ i \leftarrow i+1;\ \}$
$\textsc{SteepestAscentTwoOpt}(C)$;
return $(C, D)$;

# Genetic Algorithm for TSP

Algorithm $\textsc{GeneticTSP}(popsize, c_{max})$

$[P_0, P_1, \ldots, P_{popsize-1}] \leftarrow \textsc{Select}(popsize)$;
Sort $P_0, P_1, \ldots, P_{popsize-1}$ in increasing order of cost.
$X_{best} \leftarrow P_0$; $BestCost \leftarrow C(P_0)$;
for $(c \leftarrow 1; c \leq c_{max}; c \leftarrow c+1)$ do
  for $i \leftarrow 0$ to $popsize/2 - 1$ do
    $(P_{popsize+2i}, P_{popsize+2i+1}) \leftarrow \textsc{Rec}\ (P_{2i}, P_{2i+1})$;
  Sort $P_0, P_1, \ldots, P_{2popsize-1}$ in increasing order of cost.
  $curCost \leftarrow C(P_0)$;
  if $(curCost < BestCost)$ then
    $X_{best} \leftarrow P_0$;
    $BestCost \leftarrow curCost$;
return $X_{best}$;

**TABLE 5.7**
**GENETICTSP data with recombination operation PMREC.**

| M | n | Opt. Cost | *popsize* | $c_{max}$ | cost found min | cost found max | cost found avg | No. Opt. found |
|---|---|---|---|---|---|---|---|---|
| M50a | 50 | 185 | 8 | 50 | 192 | 214 | 200.50 | 0 |
| | | | | 100 | 191 | 219 | 200.00 | 0 |
| | | | | 200 | 190 | 203 | 196.60 | 0 |
| | | | 16 | 50 | 187 | 207 | 193.20 | 0 |
| | | | | 100 | 187 | 206 | 193.20 | 0 |
| | | | | 200 | 187 | 200 | 193.70 | 0 |
| | | | 32 | 50 | 189 | 205 | 194.70 | 0 |
| | | | | 100 | 186 | 199 | 190.70 | 0 |
| | | | | 200 | 188 | 200 | 192.40 | 0 |
| M50b | 50 | 158 | 8 | 50 | 163 | 184 | 175.40 | 0 |
| | | | | 100 | 163 | 195 | 173.70 | 0 |
| | | | | 200 | 160 | 191 | 177.30 | 0 |
| | | | 16 | 50 | 159 | 176 | 167.40 | 0 |
| | | | | 100 | 163 | 184 | 171.50 | 0 |
| | | | | 200 | 161 | 189 | 172.10 | 0 |
| | | | 32 | 50 | 161 | 173 | 167.60 | 0 |
| | | | | 100 | 163 | 178 | 169.40 | 0 |
| | | | | 200 | 159 | 178 | 166.70 | 0 |
| M50c | 50 | 155 | 8 | 50 | 162 | 181 | 169.40 | 0 |
| | | | | 100 | 159 | 186 | 169.50 | 0 |
| | | | | 200 | 159 | 187 | 169.30 | 0 |
| | | | 16 | 50 | 155 | 171 | 161.30 | 1 |
| | | | | 100 | 155 | 182 | 166.10 | 1 |
| | | | | 200 | 157 | 182 | 167.70 | 0 |
| | | | 32 | 50 | 155 | 170 | 161.60 | 1 |
| | | | | 100 | 158 | 167 | 161.40 | 0 |
| | | | | 200 | 157 | 180 | 162.50 | 0 |

A Genetic Algorithm for TSP

**TABLE 5.8**
**GENETICTSP with recombination operation MGKREC.**

| M | n | Opt. Cost | *popsize* | $c_{max}$ | cost found min | max | avg | No. Opt. found |
|---|---|---|---|---|---|---|---|---|
| M50a | 50 | 185 | 8 | 50 | 186 | 196 | 191.70 | 0 |
| | | | | 100 | 186 | 199 | 190.30 | 0 |
| | | | | 200 | 186 | 194 | 189.20 | 0 |
| | | | 16 | 50 | 186 | 192 | 189.20 | 0 |
| | | | | 100 | 185 | 192 | 187.00 | 3 |
| | | | | 200 | 185 | 192 | 187.60 | 1 |
| | | | 32 | 50 | 186 | 192 | 188.10 | 0 |
| | | | | 100 | 185 | 190 | 187.30 | 1 |
| | | | | 200 | 185 | 190 | 187.30 | 1 |
| M50b | 50 | 158 | 8 | 50 | 160 | 171 | 165.30 | 0 |
| | | | | 100 | 159 | 166 | 161.60 | 0 |
| | | | | 200 | 159 | 170 | 162.00 | 0 |
| | | | 16 | 50 | 158 | 164 | 161.20 | 1 |
| | | | | 100 | 158 | 162 | 159.80 | 1 |
| | | | | 200 | 159 | 163 | 160.70 | 0 |
| | | | 32 | 50 | 158 | 165 | 160.70 | 1 |
| | | | | 100 | 159 | 163 | 160.30 | 0 |
| | | | | 200 | 158 | 160 | 158.90 | 2 |
| M50c | 50 | 155 | 8 | 50 | 156 | 168 | 160.50 | 0 |
| | | | | 100 | 155 | 167 | 160.70 | 2 |
| | | | | 200 | 155 | 162 | 157.30 | 5 |
| | | | 16 | 50 | 155 | 162 | 157.50 | 2 |
| | | | | 100 | 155 | 159 | 156.30 | 5 |
| | | | | 200 | 155 | 159 | 155.70 | 8 |
| | | | 32 | 50 | 155 | 159 | 156.10 | 6 |
| | | | | 100 | 155 | 158 | 155.40 | 8 |
| | | | | 200 | 155 | 156 | 155.10 | 9 |