

DM811

Heuristics for Combinatorial Optimization

Lecture 14

Stochastic Local Search and Metaheuristics (2/2)

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

1. Combinatorial Optimization, Methods and Models
2. General overview
3. Solver System and Working Environment
4. Construction Heuristics
5. Local Search: Components, Basic Algorithms
6. Local Search: Neighborhoods and Search Landscape
7. Efficient Local Search: Incremental Updates and Neighborhood Pruning
8. Stochastic Local Search & Metaheuristics
9. Methods for the Analysis of Experimental Results
10. Configuration Tools: F-race
11. Very Large Scale Neighborhoods

Examples: GCP, CSP, TSP, SAT, MaxIndSet, SMTWP, Steiner Tree

Trajectory based:

- Stochastic Local Search
- Simulated Annealing
- Iterated Local Search
- Tabu Search
- Variable Neighborhood Search
- Guided Local Search

Population based:

- Evolutionary Algorithms
- (Ant Colony Optimization)
- (Particle Swarm Optimization)
- (Scatter Search and Path Relinking)
- (Cross Entropy Method / Estimation of Distribution Algorithms)

1. Population Based Metaheuristics
Evolutionary Algorithms

1. Population Based Metaheuristics
Evolutionary Algorithms

1. Population Based Metaheuristics
Evolutionary Algorithms

Key idea (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival

Iteratively apply **genetic operators** **mutation**, **recombination**, **selection** to a population of candidate solutions.

Key idea (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival

Iteratively apply **genetic operators** **mutation**, **recombination**, **selection** to a population of candidate solutions.

- **Mutation** introduces random variation in the genetic material of individuals (unary operator)

Evolutionary Algorithms

Key idea (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival

Iteratively apply **genetic operators** **mutation**, **recombination**, **selection** to a population of candidate solutions.

- **Mutation** introduces random variation in the genetic material of individuals (unary operator)
- **Recombination** of genetic material during reproduction produces **offspring** that combines features inherited from both **parents** (N-ary operator)

Key idea (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival

Iteratively apply **genetic operators** **mutation**, **recombination**, **selection** to a population of candidate solutions.

- **Mutation** introduces random variation in the genetic material of individuals (unary operator)
- **Recombination** of genetic material during reproduction produces **offspring** that combines features inherited from both **parents** (N-ary operator)
- Differences in **evolutionary fitness** lead **selection** of genetic traits ('survival of the fittest').

Evolutionary Algorithm (EA):
determine initial population sp

Evolutionary Algorithm (EA):

determine initial population *sp*

while termination criterion is not satisfied: **do**
 generate set *spr* of new candidate solutions
 by *recombination*

Evolutionary Algorithm (EA):

determine initial population sp

while termination criterion is not satisfied: **do**

 generate set spr of new candidate solutions
 by recombination

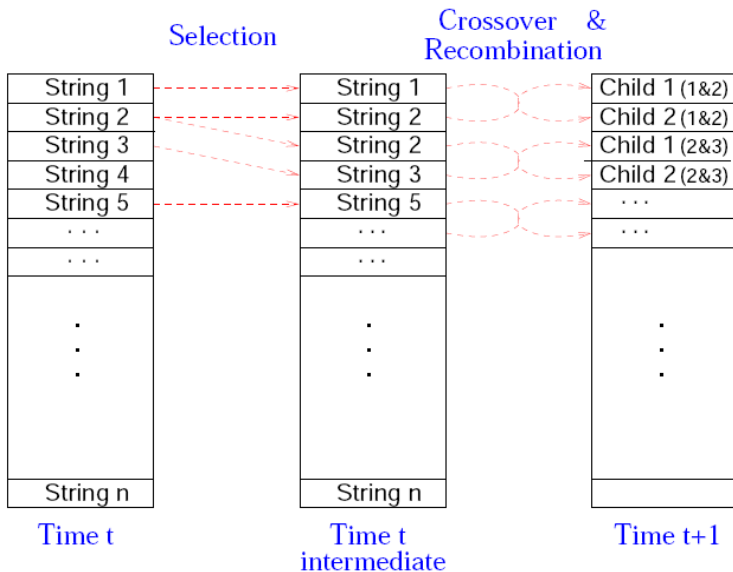
 generate set spm of new candidate solutions
 from spr and sp by mutation

Evolutionary Algorithm (EA):

determine initial population sp

while termination criterion is not satisfied: **do**

- generate set spr of new candidate solutions
by **recombination**
- generate set spm of new candidate solutions
from spr and sp by **mutation**
- select** new population sp from
candidate solutions in sp , spr , and spm



Original Streams

- **Evolutionary Programming** [Fogel et al. 1966]:
 - mainly used in continuous optimization
 - typically does not make use of **recombination** and uses **stochastic selection** based on **tournament mechanisms**.
 - often seeks to adapt the program to the problem rather than the solutions
- **Evolution Strategies** [Rechenberg, 1973; Schwefel, 1981]:
 - similar to Evolution Programming (developed independently)
 - originally developed for (continuous) numerical optimization problems;
 - operate on more natural representations of candidate solutions;
 - use **self-adaptation** of perturbation strength achieved by **mutation**;
 - typically use **elitist deterministic selection**.
- **Genetic Algorithms (GAs)** [Holland, 1975; Goldberg, 1989]:
 - mostly for discrete optimization;
 - often encode candidate solutions as bit strings of fixed length, (which is now known to be disadvantageous for combinatorial problems such as the TSP).

Problem: Pure evolutionary algorithms often lack capability of sufficient **search intensification**.

Problem: Pure evolutionary algorithms often lack capability of sufficient **search intensification**.

Solution: Apply subsidiary local search after initialization, mutation and recombination.

Memetic Algorithms [Dawkins, 1997, Moscato, 1989]

- transmission of **memes**, mimicking cultural evolution which is supposed to be direct and Lamarckian
- (aka **Genetic/Evolutionary Local Search**, or **Hybrid Evolutionary Algorithms** if more involved local search including other metaheuristics, eg, tabu search)

Memetic Algorithm (MA):

determine initial population sp

while termination criterion is not satisfied: **do**

- generate set spr of new candidate solutions
by **recombination**
- generate set spm of new candidate solutions
from spr and sp by **mutation**
- select** new population sp from
candidate solutions in sp , spr , and spm

Memetic Algorithm (MA):

determine initial population sp

perform subsidiary local search on sp

while termination criterion is not satisfied: **do**

 generate set spr of new candidate solutions
 by recombination

 perform subsidiary local search on spr

 generate set spm of new candidate solutions
 from spr and sp by mutation

 perform subsidiary local search on spm

select new population sp from

 candidate solutions in sp , spr , and spm

Individual	\longleftrightarrow	Solution to a problem
Genotype space	\longleftrightarrow	Set of all possible individuals determined by the solution encoding
Phenotype space	\longleftrightarrow	Set of all possible individuals determined by the genotypes (ie, the variable–value themselves)
Population	\longleftrightarrow	Set of candidate solutions
Chromosome	\longleftrightarrow	Representation for a solution in the population
Gene and Allele	\longleftrightarrow	Part and value of the representation of a solution (e.g., parameter or degree of freedom)
Fitness	\longleftrightarrow	Quality of a solution
Crossover Mutation	\longleftrightarrow	Search Operators
Natural Selection	\longleftrightarrow	Promoting the reuse of good solutions

Solution representation

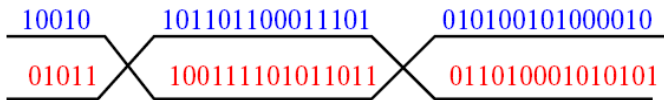
Separation between solution encode/representation (**genotype**) from actual solution (**phenotype**)

Let \mathcal{X} be the search space of a problem

- genotype set made of strings of length l whose elements are symbols from an alphabet $\mathcal{A} \rightsquigarrow$ set of all individuals is \mathcal{A}^l
 - the elements of strings are the **genes**
 - the values that each element can take are the **alleles**
- the search space is $\mathcal{S} \subseteq \mathcal{A}^l$ (set of feasible solutions)
- if the strings are member of a population they are called **chromosomes** and their recombination **crossover**
- an expression maps individual to solutions (phenotypes) $c : \mathcal{A}^l \rightarrow \mathcal{X}$ (example, unrelated parallel machine and Steiner tree)
- strings are evaluated by $f(c(s)) = g(s)$ which gives them a **fitness**

Example

1001010 1101100 0111010 1010010 1000010
 0101110 0111101 0110110 1101000 1010101

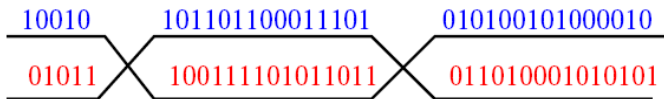


Which Produces the Offspring

0101110101101100011101011010001010101
 10010100111101011011010100101000010

Example

1001010 1101100 0111010 1010010 1000010
 0101110 0111101 0110110 1101000 1010101



Which Produces the Offspring

01011101101100011101011010001010101
 10010100111101011011010100101000010

Note: binary representation is appealing but not always good (in constrained problems binary crossovers might not be good)

Conjectures on the goodness of EA

schema: subset of \mathcal{A}^l where strings have a set of variables fixed.

Ex.: $S = 1 * * 1$

1. exploit intrinsic parallelism of schemata (but epistasis)
2. Schema Theorem:

$$E[N(S, t + 1)] \geq \frac{F(S, t)}{\bar{F}(t)} N(s, t) [1 - \epsilon(S, t)]$$

$\bar{F}(t)$ av. fitness of population, $F(S, t)$ fitness schema, $\epsilon(S, t)$ destroy effect of operators

- a method for solving all problems \Rightarrow disproved by **No Free Lunch Theorems**: no metaheuristic is better than random search; success comes from adapting the method to the problem at hand
- building block hypothesis

Initial Population

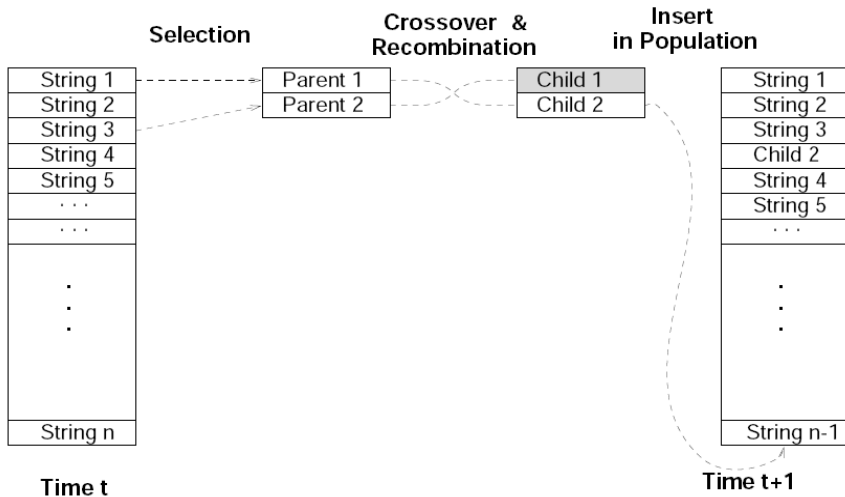
- **Generation:** often, independent, uninformed random picking from given search space.
- Which size? Trade-off
- Minimum size: connectivity by recombination is achieved if at least one instance of every allele is guaranteed to be present at each gene.
Eg: binary repr. and uniform sampling with replacement:

$$\Pr\{\text{presence of allele in } M \text{ strings of length } l\} = (1 - (0.5)^{M-1})^l$$

for $l = 50$, it is sufficient $M = 17$ to guarantee $P_2^* > 99.9\%$.

- Attempt to cover at best the search space, eg, Latin hypercube, Quasi-random (low-discrepancy) methods (Quasi-Monte Carlo method).
- **But:** can also use multiple runs of randomized construction heuristic.

Selection



Main idea: selection should be related to fitness

- Fitness proportionate selection (**roulette-wheel** method)

$$p_i = \frac{f_i}{\sum_j f_j}$$

- **Tournament selection**: a set of chromosomes is chosen and compared and the best chromosomes chosen.
- **Rank based** and selection pressure
- **Fitness sharing** (aka niching): probability of selection proportional to the number of other individuals in the same region of the search space.

Selection pressure:

$p_k = \alpha + \beta k$ probability for individual k

$$\begin{cases} \sum_{k=1}^M (\alpha + \beta k) = 1 \\ \phi = \frac{\text{Pr}[\text{selecting the best}]}{\text{Pr}[\text{selecting the median}]} \end{cases} \quad \text{selection pressure}$$

$\text{Pr}[\text{selecting the best}] = \alpha + \beta M$; $\text{Pr}[\text{selecting the median}] = \alpha + \beta(\frac{M+1}{2})$

Solving the system of equations

$$\alpha = \frac{2M - \phi(M+1)}{M(M-1)} \quad \beta = \frac{2(\phi-1)}{M(M-1)} \quad 1 \leq \phi \leq 2$$

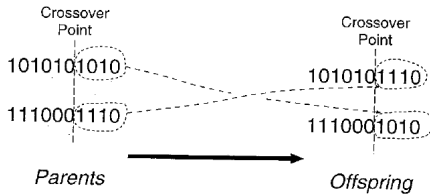
Then for a pseudo-random number the selected individual k from the cumulative probability is found in $O(1)$ solving the quadratic equation:

$$\sum_{i=1}^k \alpha + \sum_{i=1}^k \beta i = \alpha k + \beta \frac{(k+1)k}{2} = r$$

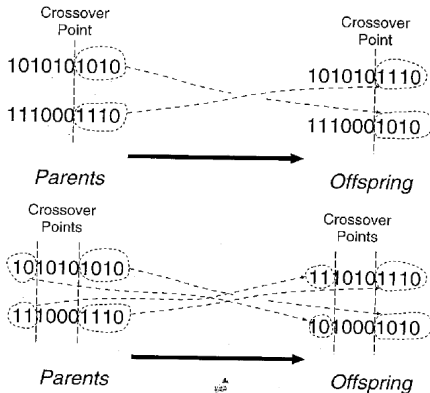
Recombination operator (Crossover)

- Binary or assignment representations
 - one-point, two-point, m-point (preference to positional bias w.r.t. distributional bias)
 - uniform cross over (through a mask controlled by a Bernoulli parameter p)
- Permutations
 - Partially mapped crossover (PMX)
 - Mask based crossover
 - Order crossover (OX)
 - Cycle crossover (CX)
- Sets
 - greedy partition crossover (GPX)
- Real vectors
 - arithmetic crossovers
 - k-point crossover

Example: crossovers for binary representations



Example: crossovers for binary representations



Uniform (mask):

s1: 1010101010

s2: 1110001110

mask: 1101011110

o1: 1010001010

o2: 1110101110

Permutations: relations of interest: adjacency, relative position, absolute position

Partially mapped crossover: defines interchanges

s1: 16 345 2

s2: 43 126 5

o1: -- 126 _

o2: -- 345 _

o1: 35 126 4

o2: 21 345 6

Order crossovers:

One point crossover: $q \in \{1..n\}$ at random

$$\begin{array}{ll} \pi_{\lambda}^{o1} := \pi_{\lambda}^{s2} & \lambda = 1..q \\ \pi_{\lambda}^{o2} := \pi_k^{s1} & k \text{ smallest with } \pi_k^{s1} \notin \{\pi_1^{o1} .. \pi_{\lambda}^{o1}\} \end{array}$$

s1: 7132 58469

s2: 1426 39875

o1: 1426 73589

o2: 7132 46985

preserves relative positions

Order crossovers:

Two point crossover: $q_1, q_2 \in \{1..n\}$, $q_1 < q_2$ at random

$$\begin{aligned} \pi_{\lambda}^{o1} &:= \pi_{\lambda}^{s2} & \lambda = 1..q \\ \pi_{\lambda}^{o2} &:= \pi_k^{s1} & k \text{ smallest with } \pi_k^{s1} \notin \{\pi_1^{o1} .. \pi_{\lambda}^{o1}\} \end{aligned}$$

s1: 71 3258 469

s2: 14 2639 875

o1: 71 ____ 469

o2: 14 ____ 875

o1: 71 2385 469

o2: 14 ____ 875

preserves relative positions

Order crossovers:

Uniform crossover: $\lambda = \{1..n\}$, $\xi_\lambda \in \{0, 1\}$

$$\begin{array}{ll}
 \text{if } \xi_\lambda = 1 & \pi_\lambda^{O_1} := \pi_k^{S_2} \quad k \text{ smallest with } \pi_k^{S_2} \notin \{\pi_1^{O_1} \dots \pi_\lambda^{O_1}\} \\
 \text{if } \xi_\lambda = 0 & \pi_\lambda^{O_1} := \pi_k^{S_1} \quad k \text{ smallest with } \pi_k^{S_1} \notin \{\pi_1^{O_1} \dots \pi_\lambda^{O_1}\}
 \end{array}$$

Cycle crossover:

- divide elements into cycles
- select randomly cycles from parents

```

Positions:   1 2 3 4 5 6 7 8 9 10 11 12
Parent 1:    A B C D E F G H I J  K  L
Parent 2:    h k c e f d b l a i  g  j
Cycle label: 1 2 3 4 4 4 2 1 1 1  2  1

Offspring:   A k C e f d b H I J  g  L

```

Partitions:

Greedy partitioning crossover

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

choose the largest set left alternating parent selection

Partitions:

Greedy partitioning crossover

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

choose the largest set left alternating parent selection

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

$o1 = \{\{1, 2, 3, 4\}\}$

Partitions:

Greedy partitioning crossover

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

choose the largest set left alternating parent selection

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

$o1 = \{\{1, 2, 3, 4\}, \{6, 7, 8\}\}$

Partitions:

Greedy partitioning crossover

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

choose the largest set left alternating parent selection

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

$o1 = \{\{1, 2, 3, 4\}, \{6, 7, 8\}, \{9, 10\}\}$

Partitions:

Greedy partitioning crossover

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

choose the largest set left alternating parent selection

$s1 = \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\}$

$s2 = \{\{4, 6, 7, 8\}, \{1, 2, 10\}, \{3, 5, 9\}\}$

$o1 = \{\{1, 2, 3, 4\}, \{6, 7, 8\}, \{9, 10\}\}$

reassign randomly left elements

$o1 = \{\{1, 2, 3, 4\}, \{6, 7, 8, 5\}, \{9, 10\}\}$

- Crossovers appear to be a crucial feature of success
- Therefore, more commonly: ad hoc crossovers
- Two off-springs are generally generated
- **Crossover rate** controls the application of the crossover. May be adaptive: high at the start and low when convergence

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution
- **Mutation rate** controls the application of bit-wise mutations. It may be adaptive: low at the start and high when convergence

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution
- **Mutation rate** controls the application of bit-wise mutations. It may be adaptive: low at the start and high when convergence
- Possible implementation through Poisson variable which determines the m genes which are likely to change allele.

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution
- **Mutation rate** controls the application of bit-wise mutations. It may be adaptive: low at the start and high when convergence
- Possible implementation through Poisson variable which determines the m genes which are likely to change allele.
- Can also use **subsidiary selection function** to determine subset of candidate solutions to which mutation is applied.

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution
- **Mutation rate** controls the application of bit-wise mutations. It may be adaptive: low at the start and high when convergence
- Possible implementation through Poisson variable which determines the m genes which are likely to change allele.
- Can also use **subsidiary selection function** to determine subset of candidate solutions to which mutation is applied.
- With real vector representation: Gaussian mutation

Subsidiary local search

- Often useful and necessary for obtaining high-quality candidate solutions.
- Typically consists of selecting some or all individuals in the given population and applying an **iterative improvement procedure** to each element of this set independently.

New Population

- Determines population for next cycle (**generation**) of the algorithm by selecting individual candidate solutions from
 - current population +
 - new candidate solutions from recombination, mutation (and subsidiary local search).
- **Generational Replacement** (λ, μ): $\lambda \leftarrow \mu$
- **Elitist strategy** ($\lambda + \mu$) the best candidates are always selected
- **Steady state** (most common) only a small number of least fit individuals is replaced
- **Goal**: Obtain population of **high-quality** solutions while maintaining **population diversity**.
Survival of the fittest and maintenance of diversity (duplicates avoided)

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
 - 1) copy common edges (param. p_e)
 - 2) add new short edges (param. p_n)
 - 3) copy edges from parents ordered by increasing length (param. p_c)
 - 4) complete using randomized greedy.

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
 - 1) copy common edges (param. p_e)
 - 2) add new short edges (param. p_n)
 - 3) copy edges from parents ordered by increasing length (param. p_c)
 - 4) complete using randomized greedy.
- **Subsidiary local search:** LK variant.

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
 - 1) copy common edges (param. p_e)
 - 2) add new short edges (param. p_n)
 - 3) copy edges from parents ordered by increasing length (param. p_c)
 - 4) complete using randomized greedy.
- **Subsidiary local search:** LK variant.
- **Mutation:** apply double-bridge to tours chosen uniformly at random.

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
 - 1) copy common edges (param. p_e)
 - 2) add new short edges (param. p_n)
 - 3) copy edges from parents ordered by increasing length (param. p_c)
 - 4) complete using randomized greedy.
- **Subsidiary local search:** LK variant.
- **Mutation:** apply double-bridge to tours chosen uniformly at random.
- **Selection:** Selects the μ best tours from current population of $\mu + \lambda$ tours (=simple elitist selection mechanism).

Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
 - 1) copy common edges (param. p_e)
 - 2) add new short edges (param. p_n)
 - 3) copy edges from parents ordered by increasing length (param. p_c)
 - 4) complete using randomized greedy.
- **Subsidiary local search:** LK variant.
- **Mutation:** apply double-bridge to tours chosen uniformly at random.
- **Selection:** Selects the μ best tours from current population of $\mu + \lambda$ tours (=simple elitist selection mechanism).
- **Restart operator:** whenever average bond distance in the population falls below 10.

- Through Markov chains modelling some versions of evolutionary algorithms can be made to converge with probability 1 to the best possible solutions in the limit [Fogel, 1992; Rudolph, 1994].
- Convergence rates on mathematically tractable functions or with local approximations [Bäck and Hoffmeister, 2004; Beyer, 2001].
- "No Free Lunch Theorem" [Wolpert and Macready, 1997]. On average, within some assumptions, blind random search is as good at finding the minimum of all functions as is hill climbing.

- Through Markov chains modelling some versions of evolutionary algorithms can be made to converge with probability 1 to the best possible solutions in the limit [Fogel, 1992; Rudolph, 1994].
- Convergence rates on mathematically tractable functions or with local approximations [Bäck and Hoffmeister, 2004; Beyer, 2001].
- "No Free Lunch Theorem" [Wolpert and Macready, 1997]. On average, within some assumptions, blind random search is as good at finding the minimum of all functions as is hill climbing.

However:

- These theoretical findings are not very practical.
- EAs are made to produce useful solutions rather than perfect solutions.

NFL: No Free Lunch

All search algorithms are equivalent when compared over all possible discrete functions. Wolpert, Macready (1995)

Consider any algorithm A_i applied to function f_j .

$On(A_i, f_j)$ outputs the order in which A_i visits the elements in the codomain of f_j . Resampling is ignored. For every pair of algorithms A_k and A_l and for any function f_j , there exist a function f_l such that

$$On(A_i, f_j) \equiv On(A_k, f_l)$$

Consider a “BestFirst” versus a “WorstFirst” local search with restarts. For every j there exists an l such that

$$On(BestFirst, f_j) \equiv On(WorstFirst, f_l)$$

Research Goals

- Analyzing classes of optimization problems and determining **experimentally** the best components for evolutionary algorithms.
- Applying evolutionary algorithms to problems that are dynamically changing.
- Gaining **theoretical** insights for the choice of components.
- Prove bounds on the runtime that such algorithms have in order to obtain optimal or nearly optimal solutions.
(Bio-inspired algorithms are
 - general-purpose algorithms
 - randomized algorithms = stochastic search algorithmscomputational complexity analysis is achieved by bounding the expected runtime to achieve good solutions for a certain problem)

- Moraglio A. and Poli R. (2011). **Topological crossover for the permutation representation**. *Intelligenza Artificiale*, 5(1), pp. 49–70.
- Neumann F., Witt C., Neumann F., and Witt C. (2010). **Bioinspired Computation in Combinatorial Optimization**. Natural Computing Series. Springer Berlin Heidelberg.
- Reeves C. (2002). **Genetic algorithms**. In *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, vol. 57 of **International Series in Operations Research & Management Science**, pp. 55–82. Kluwer Academic Publishers, Norwell, MA, USA.