Multidimensional Data Visualization

Optimization-Based Visualization

Multidimensional scaling (MDS) – a technique for exploratory analysis of multidimensional data

- Pairwise dissimilarities between n objects are given by a matrix (δ_{ij}), $i, j = 1, \ldots, n$, it is supposed that $\delta_{ij} = \delta_{ji}$.
- The points representing objects in an m-dimensional embedding space $\mathbf{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, n$ should be found whose inter-point distances fit the given dissimilarities.
- The problem is reduced to minimization of a fitness criterion, e.g. so called *Stress* function

$$S(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \left(d\left(\mathbf{x}_i, \mathbf{x}_j\right) - \delta_{ij} \right)^2 ,$$

where $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$; $d(\mathbf{x}_i, \mathbf{x}_j)$ denotes the distance between the points $\mathbf{x}_i$ and $\mathbf{x}_j$; weights $w_{ij} > 0, i, j = 1, \ldots, n$.

MDS is a difficult global optimization problem

- Although *Stress* function is defined by an analytical formula which seems rather simple, it normally has many local minima.
- The problem is high dimensional: $\mathbf{x} \in \mathbb{R}^N$ and the number of variables is equal to $N = n \times m$.
- *Stress* function is invariant with respect to translation, rotation and mirroring.
- Smoothness of *Stress* function depends on distances $d(\mathbf{x}_i, \mathbf{x}_j)$, however, non-differentiability normally cannot be ignored. Minkowski distances

$$d_r(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{m} \left| x_{ik} - x_{jk} \right|^r \right)^{1/r}.$$

Global optimization

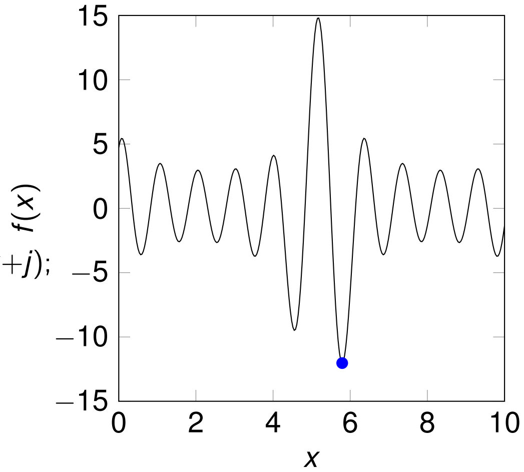
Find $f^* = \min_{\mathbf{x} \in \mathbb{A}} f(\mathbf{x})$ and $\mathbf{x}^* \in \mathbb{A},\ f(\mathbf{x}^*) = f^*$, where $\mathbb{A} \subseteq \mathbb{R}^n$.

Example:

- $n = 1$;
- $\mathbb{A} = [0, 10]$;
- Objective function

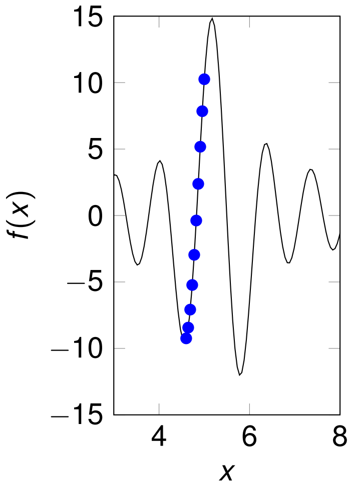
$$f(x) = -\sum_{j=1}^{5} j \sin((j+1)x + j);$$

- $f^* = -12.0312$;
- $x^* = 5.7918$.



Local optimization

- A point $\mathbf{x}^*$ is a local minimum point if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for $\mathbf{x} \in N$, where N is a neighborhood of $\mathbf{x}^*$.
- A local minimum point can be found stepping in the direction of steepest descent.
- Without additional information one cannot say if the local minimum is global.
- How do we know if we are in the deepest hole?



Algorithms for global optimization

- With guaranteed accuracy:
  - Lipschitz optimization,
  - Branch and bound algorithms.
- Randomized algorithms and heuristics:
  - Random search,
  - Multi-start,
  - Clustering methods,
  - Generalized descent methods,
  - Simulated annealing,
  - Evolutionary algorithms (genetic algorithms, evolution strategies),
  - Swarm-based optimization (particle swarm, ant colony).

Criteria of performance of global optimization algorithms

- Speed:
  - Time of optimization,
  - Number of objective function (and sometimes gradient, bounding, and other functions) evaluations,
  - Both criteria are equivalent when objective function is “expensive” – its evaluation takes much more time than auxiliary computations of an algorithm.
- Best function value found.
- Reliability – how often problems are solved with prescribed accuracy.

Parallel global optimization

- Global optimization problems are classified difficult in the sense of the algorithmic complexity theory. Global optimization algorithms are computationally intensive.
- When computing power of usual computers is not sufficient to solve a practical global optimization problem, high performance parallel computers and computational grids may be helpful.
- An algorithm is more applicable in case its parallel implementation is available, because larger practical problems may be solved by means of parallel computations.

Criteria of performance of parallel algorithms

- Efficiency of the parallelization can be evaluated using several criteria taken into account the optimization time and the number of processors.
- A commonly used criterion of parallel algorithms is the speedup:

  $$s_p = \frac{t_1}{t_p},$$

  where t_p is the time used by the algorithm implemented on p processors.
- The speedup divided by the number of processors is called the efficiency:

  $$e_p = \frac{s_p}{p}.$$

Covering methods

- Detect and discard non-promising sub-regions using
  - interval arithmetic $\left\{f\left(X\right)|X \in \underline{\overline{X}}, \underline{\overline{X}} \in [\mathbb{R},\mathbb{R}]^n\right\} \subseteq \underline{\overline{f}}\left(\underline{\overline{X}}\right)$,
  - Lipschitz condition $|f(x) - f(y)| \leq L||x - y||$,
  - convex envelopes,
  - statistical estimates,
  - heuristic estimates,
  - ad hoc algorithms.
- May be implemented using branch and bound algorithms.
- May guarantee accuracy for some classes of problems: $f^* \leq \min_{x \in D} f(x) + \epsilon$.

Branch and bound

- An iteration of a classical branch and bound processes a node in the search tree representing a not yet explored subspace.
- Iteration has three main components: selection of the node to process, branching and bound calculation.
- The bound for the objective function over a subset of feasible solutions should be evaluated and compared with the best objective function value found.
- If the evaluated bound is worse than the known function value, the subset cannot contain optimal solutions and the branch describing the subset can be pruned.
- The fundamental aspect of branch and bound is that the earlier the branch is pruned, the smaller the number of complete solutions that will need to be explicitly evaluated.

General algorithm for partition and branch-and-bound

```
Cover $D$ by $L = \{L_j | D \subseteq \bigcup L_j, j = \overline{1,m}\}$ using **covering rule.**
while $L \neq \emptyset$ do
    Choose $l \in L$ using **selection rule**, $L \leftarrow L \backslash \{l\}$.
    if bounding rule is satisfied for $l$ then
        Partition $l$ into $p$ subsets $l_j$ using **subdivision rule.**
        for $j = 1, \ldots, p$ do
            if bounding rule is satisfied for $l_j$ then
                $L \leftarrow L \bigcup \{l_j\}$.
            end if
        end for
    end if
end while
```

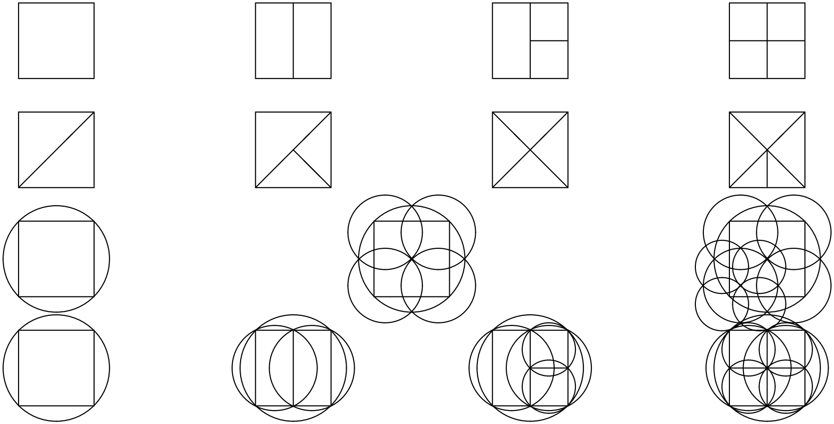
Rules of branch and bound algorithms for global optimization

- The **rules** of **covering** and **subdivision** depend on type of partitions used: hyper-rectangular, simplicial, etc.
- **Selection rules**:
  - Best first, statistical – the best candidate: priority queue.
  - Depth first – the youngest candidate: stack or without storing.
  - Breadth first – the oldest candidate: queue.
- **Bounding rule** describes how the bounds for minimum are found. For the upper bound the best currently found function value may be accepted. The lower bound may be estimated using
  - Convex envelopes of the objective function.
  - Lipschitz condition $|f(x) - f(y)| \leq L||x - y||$.
  - Interval arithmetic $\left\{f(X) \,|\, X \in \underline{\overline{X}}, \underline{\overline{X}} \in [\mathbb{R}, \mathbb{R}]^n\right\} \subseteq \underline{\overline{f}}\left(\underline{\overline{X}}\right)$.

Generalized branch and bound template

- Standard parts of branch and bound algorithms are implemented in the package, only specific parts should be implemented by the user.
- Suitable for implementation of branch and bound algorithms for combinatorial optimization and for covering methods of continuous global optimization.
- Parallelization tools include master-slave and distributed versions of parallel branch and bound.

Rules of covering rectangular feasible region and branching



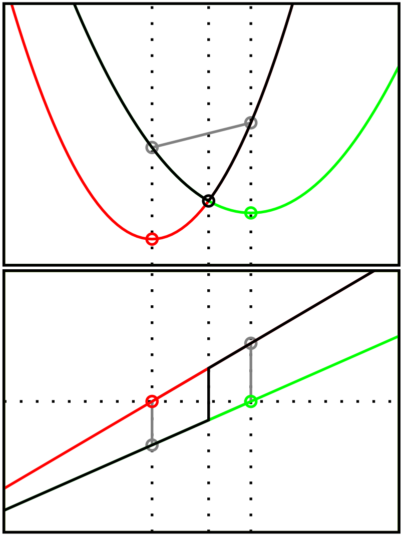
MDS with city-block distances

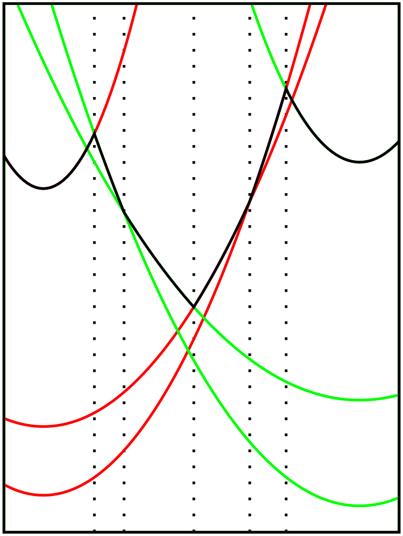
- If city-block distances $d_1(\mathbf{x}_i, \mathbf{x}_j)$ are used, *Stress* can be redefined as

$$S(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \left(\sum_{k=1}^{m} \left| x_{ik} - x_{jk} \right| - \delta_{ij} \right)^2 .$$

- In the case of city-block distances and $m \geq 2$ *Stress* can be non-differentiable even at a minimum point. With this respect the case of city-block distances is different from the other cases of Minkowski distances when positiveness of distances $d(\mathbf{x}_i^*, \mathbf{x}_j^*),\ i, j = 1, \ldots, n$ at a local minimum point $\mathbf{x}^*$ implies differentiability of *Stress*.
- However *Stress* with city-block distances is piecewise quadratic, and such a structure can be exploited for tailoring of ad hoc global optimization algorithms.

Piecewise quadratic function non-differentiable at a minimum point





Decomposition of the original optimization problem into a set of quadratic programming problems

- Let $A(\mathbf{P})$ denote a set such that

  $$A(\mathbf{P}) = \left\{\mathbf{x} | x_{ik} \le x_{jk} \text{ for } p_{ki} < p_{kj},\ i,j = 1,\dots,n,\ k = 1,\dots,m\right\},$$

  where $\mathbf{P} = (p_{11}, p_{12}, \dots, p_{mn})$, $\mathbf{p}_k = (p_{k1}, p_{k2}, \dots, x_{kn})$ are m permutations of $1, \dots, n$.
- For $\mathbf{x} \in A(\mathbf{P})$,

  $$S(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \left(\sum_{k=1}^{m} \left(x_{ik} - x_{jk}\right) z_{kij} - \delta_{ij} \right)^2,$$

  where

  $$z_{kij} = \begin{cases} 1, & p_{ki} > p_{kj}, \\ -1, & p_{ki} < p_{kj}. \end{cases}$$

Since function $S(\mathbf{x})$ is quadratic over polyhedron $\mathbf{x} \in A(\mathbf{P})$ the minimization problem

$$\min_{\mathbf{x} \in A(\mathbf{P})} S(\mathbf{x})$$

is a quadratic programming problem. It is equivalent to

$$\min \left[-\sum_{k=1}^{m} \sum_{i=1}^{n} x_{ik} \sum_{j=1}^{n} w_{ij} \delta_{ij} z_{kij} + \right.$$

$$\left. \frac{1}{2} \sum_{k=1}^{m} \sum_{l=1}^{m} \sum_{i=1}^{n} \left(x_{ik} x_{il} \sum_{t=1, t \neq i}^{n} w_{it} z_{kit} z_{lit} - \sum_{j=1, j \neq i}^{n} x_{ik} x_{jl} w_{ij} z_{kij} z_{lij} \right) \right]$$

$$\text{s.t.} \ \sum_{i=1}^{n} x_{ik} = 0, \ k = 1, \ldots, m,$$

$$x_{\{j | p_{kj} = i+1\}, k} - x_{\{j | p_{kj} = i\}, k} \geq 0, \ k = 1, \ldots, m, \ i = 1, \ldots, n-1.$$

Quadratic programming problem

- The definition of quadratic programming problem in matrix form:

$$\min \left(-\mathbf{d}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{D} \mathbf{x} \right)$$

$$\begin{aligned} \text{s.t. } \mathbf{A}^0 \mathbf{x} &= 0, \\ \mathbf{A}^k \mathbf{x} &\geq 0, \; k = 1, \ldots, m, \end{aligned}$$

- where
  - $\mathbf{d}$ is a vector of dimensionality $(nm \times 1)$,
  - $\mathbf{D}$ is an $(nm \times nm)$ matrix,
  - $\mathbf{A}^0$ is an $(m \times nm)$ matrix,
  - $\mathbf{A}^k$, $k = 1, \ldots, m$, are $((n-1) \times nm)$ matrices.

Definition of quadratic programming problem

$$
\begin{array}{rcl}
d_{kn-n+i} & = & \left. \displaystyle\sum_{j=1, j\neq i}^{n} w_{ij}\delta_{ij} z_{kij} \right| \begin{array}{c} k = 1, \dots, m, \\ i = 1, \dots, n. \end{array} \\
D_{kn-n+i, ln-n+j} & = & \left\{ \begin{array}{ll} \sum_{t=1, t\neq i}^{n} w_{it} z_{kit} z_{lit}, & i = j, \\ -w_{ij} z_{kij} z_{lij}, & i \neq j, \end{array} \right| \begin{array}{c} k, l = 1, \dots, m, \\ i, j = 1, \dots, n. \end{array} \\
A^0_{kj} & = & \left\{ \begin{array}{ll} 1, & j = kn - n + 1, \dots, kn, \\ 0, & \text{otherwise}, \end{array} \right| \begin{array}{c} k = 1, \dots, m, \\ j = 1, \dots, mn. \end{array} \\
A^k_{ij} & = & \left\{ \begin{array}{cc} 1, & p_{k, j-kn+n} = i + 1, \\ -1, & p_{k, j-kn+n} = i, \\ 0, & \text{otherwise}, \end{array} \right| \begin{array}{c} k = 1, \dots, m, \\ i = 1, \dots, n - 1, \\ j = 1, \dots, mn. \end{array}
\end{array}
$$

Two level MDS method with city-block distances

- Taking into account the structure of the minimization problem a two level minimization method can be applied: to solve a combinatorial optimization problem at the upper level, and to solve a quadratic programming problem at the lower level:

$$\min_{\mathbf{P}} S(\mathbf{P}), \text{ s.t. } S(\mathbf{P}) = \min_{\mathbf{x} \in A(\mathbf{P})} S(\mathbf{x}) \sim$$

$$\sim \min \left(-\mathbf{c_P}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{Q_P} \mathbf{x} \right) \text{ s.t. } \mathbf{Ex} = \mathbf{0},\ \mathbf{A_P x} \geq \mathbf{0},$$

$$\mathbf{c_P} \in \mathbb{R}^{nm},\ \mathbf{Q_P} \in \mathbb{R}^{nm \times nm},\ \mathbf{E} \in \mathbb{R}^{m \times nm},\ \mathbf{A_P} \in \mathbb{R}^{(n-1) \times nm}.$$

- For the lower level problem a quadratic programming method can be applied.

Upper level combinatorial problem

- The upper level objective function is defined over the set of m-tuple of permutations of $1, ..., n$.
- The number of feasible solutions is $(n!)^m$, avoiding mirrored solutions it can be reduced to approximately $(n!)^m/(2^m m!)$.
- Solution of combinatorial problem:
    - explicit enumeration of all feasible solutions;
    - branch and bound;
    - pure random search;
    - multistart;
    - evolutionary algorithm.

Local search based on quadratic programming

- A minimum point of a quadratic programming problem is not necessary a local minimizer of the initial problem of minimization of *Stress*, if it is on the boundary of polyhedron.
- Therefore local search may be continued:
  - Go to the neighbor polyhedron on the opposite side of the active inequality constrains.
  - If $i, \ldots, j$ inequality constraints $\mathbf{A_P x} \geq \mathbf{0}$ are active, $i \leq p_{kt} \leq j + 1$ should be updated to $i + j + 1 - p_{kt}$.
  - Perform quadratic programming.
  - Repeat while better values are found and some inequality constrains are active.

Parallel explicit enumeration

- Each processor runs the same algorithm generating feasible solutions which should be enumerated explicitly, but only each p-th is explicitly enumerated on a processor where p is the number of processors.
- The first processor explicitly enumerates the first, $(p+1)$ and so on generated solutions. The second processor enumerates 2-nd, $(p+2)$, ... The p-th processor enumerates p-th, $2p$-th, ...
- It is assumed that generation of the solutions to be explicitly enumerated requires much less computational time than the explicit enumeration which requires solution of the lower level quadratic programming problem.
- The results of different processors are collected when the generation of solutions and explicit enumeration are finished.
- The standardized message-passing communication protocol MPI is used for communication between parallel processors.

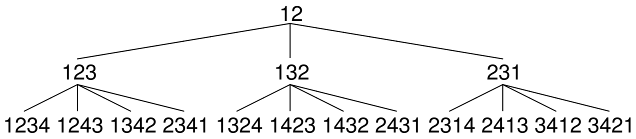
Branch and bound for MDS with city-block distances

- A subset of feasible solutions is represented by a partial solution defined by m-tuple $\overline{\mathbf{P}}$ of permutations of $1, \ldots, \overline{n}$ where only $\overline{n}$ of n objects are considered.
- Depth first selection strategy is used to avoid storing of candidate nodes.
- The lower bound is a partial *Stress* evaluated at the solution of the lower level quadratic programming problem for $\overline{n}$ objects over a polyhedron $A(\overline{\mathbf{P}})$:

$$\min_{\overline{\mathbf{x}} \in A(\overline{\mathbf{P}})} \sum_{i=1}^{\overline{n}} \sum_{j=1}^{\overline{n}} w_{ij} \left(\sum_{k=1}^{m} \left| x_{ik} - x_{jk} \right| - \delta_{ij} \right)^2,$$

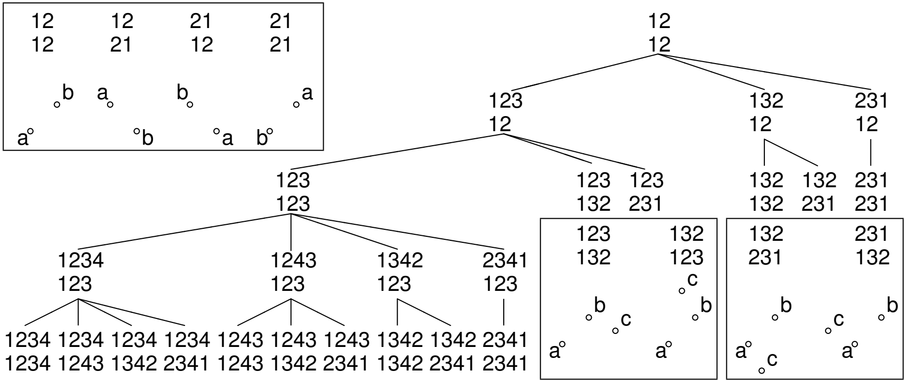
where $\overline{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_{\overline{n}})$.

A search tree for $m = 1$



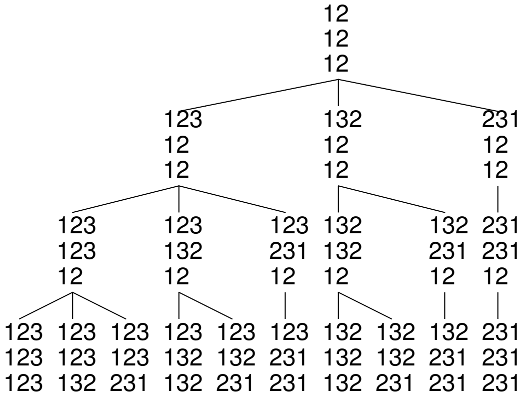
- Every numeral represents a value of p_{1i}, $i = 1, \ldots, \overline{n}$.
- To refuse mirrored solutions the search tree starts with a root node representing partial solution “12”. Although the sequence numbers of the first two objects may be changed (13 and 23) after assignment of the third object, their sequence is not changed ($1 < 2$, $1 < 3$ and $2 < 3$).
- The number of feasible solutions is $n!/2$.
- The number of nodes $\sum_{i=2}^{n} \frac{i!}{2}$.

A search tree for $m = 2$



- Every numeral represents p_{ki}, $k = 1, \ldots, m$, $i = 1, \ldots, \bar{n}$.
- The number of feasible solutions is $n!^2/8 + n!/4$.
- The lower bound for *STRESS* is evaluated only over the partial solutions defined by permutations of equal sizes.

A search tree for $m = 3$



- The number of feasible solutions is $n!^3/48 + n!^2/8 + n!/6$.

Parallel branch and bound algorithm

- Each process runs the same algorithm generating partial solutions of up to some level l: $\overline{n} \leq l$.
- Each p-th partial solution at the level $\overline{n} = l$ is evaluated and branched if needed, where p is the number of processes.
- The first process evaluates and branches the first, $(p + 1)$ and so on generated partial solutions. The second process evaluates 2-nd, $(p + 2)$, ... The p-th processor evaluates p-th, $2p$-th, ...
- The results of different processes are collected at the end of computation.
- The standardized message-passing communication protocol MPI is used for communication between parallel processes.

Evolutionary algorithm (n_p, t_c, N_{init}, p_{mut})

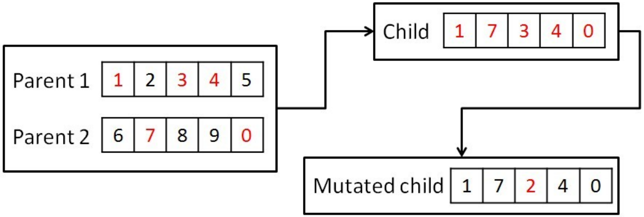
```
Generate $N_{init}$ random feasible solutions.
Perform local search from $n_p$ best solutions to form initial population.
**while** $t_c$ time has not passed
    With probability $p_{mut}$ mutate randomly chosen individual.
    Uniformly randomly generate two indices of parents $i$ and $j$.
    Uniformly randomly generate two integers $k$ and $l$ from $1, \dots, n$
    Compose permutations of descendant:
        elements $1, \dots, k-1, l+1, \dots, n$ are elements of $\mathbf{p}_i$,
        elements $k, \dots, l$ are the missing numbers ordered in
            the same way as they are ordered in $\mathbf{p}_j$.
    Perform local search based on quadratic programming.
    **If** the offspring is more fitted than the worst individual,
        **then** the offspring replaces the latter.
```

Genetic operations

(1) Two parents are selected form population

(2) and combined to generate a child (Crossover).

(3) A small random change of each gene is made with probability $\frac{1}{d}$



Parallel version of evolutionary algorithm

- Multiple populations.
- Each processor runs the same evolutionary algorithm with different sequences of random numbers.
- The results of different processors are collected when search is finished after predefined time.
- Communications between processors are kept to minimum.