Note:
- Local minima depend on $g$ and neighborhood relation $\mathcal{N}$.
- Larger neighborhoods $\mathcal{N}$ induce
  - neighborhood graphs with smaller diameter;
  - fewer local minima.

Ideal case: exact neighborhood, i.e., neighborhood relation for which any local optimum is also guaranteed to be a global optimum.

- Typically, exact neighborhoods are too large to be searched effectively (exponential in size of problem instance).
- But: exceptions exist, e.g., polynomially searchable neighborhood in Simplex Algorithm for linear programming.
Trade-off:
- Using larger neighborhoods can improve performance of II (and other LS methods).
- **But**: time required for determining improving search steps increases with neighborhood size.

More general trade-off:
Effectiveness vs Efficiency (= time complexity of search steps).

In II, various mechanisms (pivoting rules) can be used for choosing improving neighbor in each step:

- **Best Improvement** (aka gradient descent, steepest descent, greedy hill-climbing): Choose maximally improving neighbor, i.e., randomly select from $I^*(s) := \{ s' \in N(s) \mid f(s') = g^* \}$, where $g^* := \min \{ f(s') \mid s' \in N(s) \}$.

  **Note**: Requires evaluation of all neighbors in each step.

- **First Improvement**: Evaluate neighbors in fixed order, choose first improving step encountered.

  **Note**: Can be much more efficient than Best Improvement; order of evaluation can have significant impact on performance.

---

**Iterative Improvement (2 OPT)**

```plaintext
procedure TSP-2opt-first(s)
input: an initial candidate tour $s \in S(\varepsilon)$
output: a local optimum $s \in S(\pi)$
$\Delta = 0$;
do
  Improvement := FALSE;
  for $i = 1$ to $n - 2$ do
    if $i = 1$ then $n' = n - 1$ elseif $n' = n$
      for $j = i + 2$ to $n'$ do
        $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
        if $\Delta_{ij} < 0$ then
          UpdateTour(s, i, j);
          Improvement := TRUE;
        end
      end
    end
  until Improvement == TRUE;
end TSP-2opt-first
```

**Example: Random-order first improvement for the TSP**

- **Given**: TSP instance $G$ with vertices $v_1, v_2, \ldots, v_n$.
- **search space**: Hamiltonian cycles in $G$; use standard 2-exchange neighborhood

  **Initialization**:
  - search position := fixed canonical path $(v_1, v_2, \ldots, v_n, v_1)$
  - $P :=$ random permutation of $\{1, 2, \ldots, n\}$

  **Search steps**: determined using first improvement w.r.t. $f(p) =$ weight of path $p$, evaluating neighbors in order of $P$ (does not change throughout search)

  **Termination**: when no improving search step possible (local minimum)
Speed-up Techniques: Neighborhood Pruning

- **Idea:** Reduce size of neighborhoods by excluding neighbors that are likely (or guaranteed) not to yield improvements in $g$.  
- **Note:** Crucial for large neighborhoods, but can be also very useful for small neighborhoods (e.g., linear in instance size).

Example: Heuristic candidate lists for the TSP

- **Intuition:** High-quality solutions likely include short edges.  
- **Candidate list** of vertex $v$: list of $v$’s nearest neighbors (limited number), sorted according to increasing edge weights.  
- Search steps (e.g., 2-exchange moves) always involve edges to elements of candidate lists.  
- Significant impact on performance of LS algorithms for the TSP.

Perturbative Search on the Traveling Salesman Problem

- $k$-exchange heuristics
  - 2-opt
  - 2.5-opt
  - Or-opt
  - 3-opt
- complex neighborhoods
  - Lin-Kernighan
  - Helsgaun’s Lin-Kernighan
  - Dynasearch
  - ejection chains approach

Implementations exploit speed-up techniques

1. neighborhood pruning: fixed radius nearest neighborhood search
2. neighborhood lists: restrict exchanges to most interesting candidates
3. don’t look bits: focus perturbative search to “interesting” part
4. sophisticated data structures

TSP data structures

Tour representation:

- determine pos of $v$ in $\pi$
- determine succ and prec
- check whether $u_k$ is visited between $u_i$ and $u_j$
- execute a $k$-exchange (reversal)

Possible choices:

- $|V| < 1.000$ array for $\pi$ and $\pi^{-1}$
- $|V| < 1.000.000$ two level tree
- $|V| > 1.000.000$ splay tree

Moreover static data structure:

- priority lists
- k-d trees

Discuss implementation of local search for TSP by Stützle: http://www.sls-book.net/implementations.html

```
two_opt_b(tour);
two_opt_f(tour);
two_opt_best(tour);
two_opt_first(tour);
three_opt_first(tour);
```
Solution Representations and Neighborhoods

Three different types of solution representations:

- **Permutation**
  - *linear permutation*: Single Machine Total Weighted Tardiness Problem
  - *circular permutation*: Traveling Salesman Problem
- **Assignment**: Graph Coloring Problem, SAT, CSP
- **Set, Partition**: Max Independent Set

A neighborhood function $N : S \to S \times S$ is also defined through an operator. An operator $\Delta$ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \iff \exists \delta \in \Delta \delta(s) = s'$$

Permutations

$\Pi(n)$ indicates the set of all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2, \ldots, n)$ is the identity permutation $\iota$.

If $\pi \in \Pi(n)$ and $1 \leq i \leq n$ then:

- $\pi_i$ is the element at position $i$
- $pos_\pi(i)$ is the position of element $i$

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

the permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')_i = \pi'(\pi(i))$

For each $\pi$ there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi(n)$$

Neighborhood Operators for Linear Permutations

**Swap operator**

$$\Delta_S = \{\delta^i_S | 1 \leq i \leq n\}$$

$$\delta^i_S(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

**Interchange operator**

$$\Delta_X = \{\delta^i_x | 1 \leq i < j \leq n\}$$

$$\delta^i_x(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

**Insert operator**

$$\Delta_I = \{\delta^i_I | 1 \leq i \leq n, 1 \leq j \leq n, j \neq i\}$$

$$\delta^i_I(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_{j-1} \pi_j \pi_{i+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_i \pi_{j+1} \ldots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Circular Permutations

**Reversal (2-edge-exchange)**

$$\Delta_R = \{\delta^ij_R | 1 \leq i < j \leq n\}$$

$$\delta^ij_R(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_i \pi_{j+1} \ldots \pi_{i+1} \pi_j \pi_{j+1} \ldots \pi_n)$$

**Block moves (3-edge-exchange)**

$$\Delta_B = \{\delta^ijk_B | 1 \leq i < j < k \leq n\}$$

$$\delta^ijk_B(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_k \pi_i \ldots \pi_{j-1} \pi_{k+1} \ldots \pi_n)$$

**Short block move (Or-edge-exchange)**

$$\Delta_{SB} = \{\delta^ij_{SB} | 1 \leq i < j \leq n\}$$

$$\delta^ij_{SB}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \ldots \pi_{j-1} \pi_{j+3} \ldots \pi_n)$$
Neighborhood Operators for Assignments

An assignment can be represented as a mapping 
\( \sigma: \{X_1 \ldots X_n\} \rightarrow \{v: v \in D, |D| = k\} \):
\[ \sigma = \{X_i = v_i, X_j = v_j, \ldots\} \]

One exchange operator
\[ \Delta_{1E} = \{\delta_{1E}^{|i|}: 1 \leq i \leq n, 1 \leq l \leq k\} \]
\[ \delta_{1E}^{|i|}(\sigma) = \{\sigma: \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \forall j \neq i\} \]

Two exchange operator
\[ \Delta_{2E} = \{\delta_{2E}^{|i,j|}: 1 \leq i < j \leq n\} \]
\[ \delta_{2E}^{|i,j|}(\sigma) = \{\sigma: \sigma'(X_i) = \sigma(X_i), \sigma'(X_j) = \sigma(X_j) \text{ and } \sigma'(X_l) = \sigma(X_l) \forall l \neq i, j\} \]

Neighborhood Operators for Partitions or Sets

An assignment can be represented as a partition of objects selected and not selected 
\( s: \{X\} \rightarrow \{C, \overline{C}\} \)
(\(\overline{C}\) can also be represented by a bit string)

One addition operator
\[ \Delta_{1E} = \{\delta_{1E}^{|v|}: v \in \overline{C}\} \]
\[ \delta_{1E}^{|v|}(s) = \{s: C' = C \cup v \text{ and } \overline{C}' = \overline{C} \setminus v\} \]

One deletion operator
\[ \Delta_{1E} = \{\delta_{1E}^{|v|}: v \in C\} \]
\[ \delta_{1E}^{|v|}(s) = \{s: C' = C \setminus v \text{ and } \overline{C}' = \overline{C} \cup v\} \]

Swap operator
\[ \Delta_{1E} = \{\delta_{1E}^{|u|, v|}: v \in C, u \in \overline{C}\} \]
\[ \delta_{1E}^{|u|, v|}(s) = \{s: C' = C \setminus u \setminus v \text{ and } \overline{C}' = \overline{C} \cup v \cup u\} \]

Other Examples

Delta evaluations and neighborhood examinations in:
- Permutations
- SMTWTP
- Assignments
- SAT
- Sets
  - Max Independent Set

SMTWTP
- Interchange: size \( \binom{n}{2} \) and \( O(|i - j|) \) evaluation each
  - first-improvement: \( \pi_j, \pi_k \)
    - \( p_{\pi_j} \leq p_{\pi_k} \) for improvements, \( w_j T_j + w_k T_k \) must decrease because jobs in \( \pi_j \)
    - \( p_{\pi_j} \geq p_{\pi_k} \) possible use of auxiliary data structure to speed up the computation
  - first-improvement: \( \pi_j, \pi_k \)
    - \( p_{\pi_j} \leq p_{\pi_k} \) for improvements, \( w_j T_j + w_k T_k \) must decrease at least as the best
    - \( p_{\pi_j} \geq p_{\pi_k} \) possible use of auxiliary data structure to speed up the computation
- Swap: size \( n - 1 \) and \( O(1) \) evaluation each
- Insert: size \( (n - 1)^2 \) and \( O(|i - j|) \) evaluation each
  But possible to speed up with systematic examination by means of swaps:
  an interchange is equivalent to \( |i - j| \) swaps hence overall \( O(n^2) \)
Example: Iterative Improvement for $k$-col

- **search space** $S$: set of all $k$-colorings of $G$
- **solution set** $S'$: set of all proper $k$-coloring of $F$
- **neighborhood relation** $N$: 1-exchange neighborhood (as in Uninformed Random Walk)
- **memory**: not used, i.e., $M := \{0\}$
- **initialization**: uniform random choice from $S$, i.e., $\text{init}\{\emptyset, \varphi'\} := 1/|S|$ for all colorings $\varphi'$
- **step function**:
  - **evaluation function**: $g(\varphi) :=$ number of edges in $G$ whose ending vertices are assigned the same color under assignment $\varphi$ (Note: $g(\varphi) = 0$ iff $\varphi$ is a proper coloring of $G$.)
  - **move mechanism**: uniform random choice from improving neighbors, i.e., $\text{step}\{\varphi, \varphi'\} := 1/|I(\varphi)|$ if $s' \in I(\varphi)$, and 0 otherwise, where $I(\varphi) := \{\varphi' \mid N(\varphi, \varphi') \land g(\varphi') < g(\varphi)\}$
- **termination**: when no improving neighbor is available i.e., $\text{terminate}\{\varphi, \top\} := 1$ if $I(\varphi) = \emptyset$, and 0 otherwise.