Section 2.3

Heuristic Local–Search Techniques

Why heuristic local search?

- In general, the local–search scheme from Slide no. 48 only guarantees termination at a *local* optimum.
- Clearly, a local optimum may be much worse than a global optimum.
- It is easy to construct examples where almost all local optima are nearly worst–case solutions.
- Therefore, the pure local–search scheme from Slide no. 48 is not a good optimization algorithm in general.

General ideas for improving the scheme:

- Occasionally accept forward steps from one feasible solution to the next one even if the new solution is worse.
- Perform several forward steps and select the best intermediate result as the starting point for the next iteration of the while–loop.
- Repeat local–search runs from many different start solutions.
- Run several local–search sequences simultaneously.

Note:

- In this lecture, many details of the various algorithmic approaches are left open.

  $\longrightarrow$ This means that these details are degrees of freedom.

  $\longrightarrow$ To be specified by designers of concrete algorithmic approaches for concrete optimization problems.

- The list of presented techniques is not at all exhaustive, but the presented taxonomy should cover all techniques.
- Each technique is only briefly touched upon.

  $\longrightarrow$ Systematology is preferred over details.

  $\longrightarrow$ Details can afterwards be looked up in the literature.

- The terminology and taxonomies used in the literature are not at all standardized.

  $\longrightarrow$ No guarantee that the presentation here perfectly conforms to all presentations in the literature.

Section 2.3.1: Simulated Annealing

What is Simulated Annealing?

- Also known as the *Metropolis algorithm*.
- *Main differences* to the pure local–search scheme:
  - In Step 2 of the local–search scheme on Slide no.48, choose the neighbor s of s^* at random.

    $\longrightarrow$ It may be $c(s) \geq c(s^*)$!

  - If $c(s) < c(s^*)$, s replaces s^* as the current feasible solution as before.
  - In case $c(s) \geq c(s^*)$, a probabilistic yes/no decision is made.
  - If the outcome is "yes," s replaces s^* (s is now called s^*).
  - Otherwise, the current feasible solution is not changed in this step.

Modified local–search scheme:

1. Start with an arbitrary feasible solution $s^* \in F_I$.
2. While a certain termination condition has not yet been reached:
   (a) Randomly select some $s \in F(I)$ such that $(s^*, s) \in A_i$.
   (b) If $c(s) < c(s^*)$, let s be the new s^*.
   (c) Otherwise:
      i. Make a probabilistic yes/no decision.
      ii. If the outcome is "yes," let s be the new s^*.
      iii. Otherwise, do nothing (leave s^* as is).
3. Deliver the best solution seen so far as the final solution.

Probabilistic yes/no decision:

- In principle, this means a "biased coin–flipping" experiment where the head and the tail of the coin may occur with different probabilities.
- On a computer, this amounts to applying a random number generator.
- *Random number generator*: A deterministic number generator that simulates a non–deterministic choice of numbers.
- In the simulated–annealing algorithm, the probability of "yes" is determined by the so–called *temperature* $T > 0$: For $c(s) \geq c(s^*)$, the probability of "yes" is
$$exp\left(\frac{c(s^*) - c(s)}{T}\right) .$$
- *Observation*: Since $c(s) \geq c(s^*)$ and $T > 0$, this is indeed a probabilistic decision, that is,
$$0 \leq exp\left(\frac{c(s^*) - c(s)}{T}\right) \leq 1 .$$

Cooling schedule:

- *Open questions*:
  - how to define T;
  - how to define the termination condition.

  $\longrightarrow$ Form the *cooling schedule* as defined below.

- *Cooling schedule*:
  - A finite sequence $T_1 > T_2 > T_3 > \cdots > T_k > 0$ of temperature values is defined.
  - For $i \in \{1, \ldots, k\}$, a positive integral *sequence length* n_i is defined in addition.

- *Application* of the cooling schedule:

  For $i = 1, 2, \ldots, k$, exactly n_i iterations of the while–loop on Slide no. 92 are run with T_i being the temperature value.

Remark:

- Often, further termination criteria are applied.
- *Examples*:
  - termination as soon as "nothing significant has changed" for a while;
  - termination after a fixed amount of CPU time.
- This allows to stop the procedure prematurely in case the cooling schedule was chosen too time-consuming.
- Of course, this will usually compromise quality of the result.

Assumptions on the neighborhood relation:

- *Recall*: A neighborhood relation for an instance I of an algorithmic problem may be viewed as a directed graph G_I
  - whose nodes are F_I (the feasible solutions to I) and
  - whose arcs determine which feasible solution is neighbored to which one.

- The following conditions will be essential:

  Strong connectivity: Any feasible solution in F_I can be reached from any other feasible solution in F_I through a finite number of moves from neighbor to neighbor.

Fundamental mathematical result:

- Consider an optimization problem with a strongly connected neighborhood G_I for each instance $I \in \mathcal{I}$.
- First consider the (unrealistic) variant of simulated annealing where
  - the temperature value $T > 0$ is constant and
  - (the unrealistic part:) the number of iterations of the loop is infinite.
- For this variant, the following fact can be shown for any feasible solution s and any $T > 0$:

  For $k \to \infty$, the probability $P_{Tk}(s)$ that s is the current feasible solution in step no. k of the search converges to

$$\lim_{k\to\infty} P_{Tk}(s) = \frac{\exp(-c(s)/T)}{\sum_{t\in F_I} \exp(-c(t)/T)}\,.$$

  $\longrightarrow$ Independent of the choice of the very first start solution!

Proof of this mathematical fact:

- Via Markov chains and ergodic theory.

  $\longrightarrow$ See lectures on probability theory or stochastic processes.

- Omitted here.

Consequence of this mathematical fact:

For a feasible solution s, for $T \searrow 0$ and for $k \to \infty$, the probability $P_{Tk}(s)$ converges to 0 in case s is not an optimal solution:

$$\lim_{\substack{T \searrow 0 \\ k \to \infty}} P_{Tk}(s) = 0\,.$$

$\longrightarrow$ Proof of the consequence on the next slide.

Proof of the consequence:

- Consider two feasible solutions s_1 and s_2 such that $c(s_1) < c(s_2)$.
- Then we have

$$\frac{P_{Tk}(s_1)}{P_{Tk}(s_2)} \overset{k\to\infty}{\longrightarrow} \frac{\exp(-c(s_1)/T)}{\exp(-c(s_2)/T)}$$

$$= \exp\left(\frac{c(s_2) - c(s_1)}{T}\right) \overset{T\searrow 0}{\longrightarrow} \infty\,.$$

- The last limit is due to the assumption $c(s_2) - c(s_1) > 0$.
- Since $P_{Tk}(s_1)$ is a probability value, we have $P_{Tk}(s_1) \leq 1$.
- Thus, convergence to "∞" is only possible if

$$\lim_{T\searrow 0} \lim_{k\to\infty} P_{Tk}(s_2) = 0\,.$$

Interpretation of the consequence:

- If we were able to run simulated annealing
  - with an infinite number of temperature values $T_1 > T_2 > T_3 > \cdots \longrightarrow 0$ and
  - for each temperature value T_i with an infinite number n_i of steps,

  then we reach (and stay on) the optimal solutions at the very end ...
- ... whatever the meaning of "end" might be here.
- *Note*: From the formula for $P_{Tk}(s)$ on Slide no. 97,
  - one cannot only conclude that the *optimal* solutions become overwhelmingly likely when T is decreased (as we did),
  - but one can analogously conclude that, in general, a better solution is more likely than a worse solution.

How to choose the cooling schedule?

- Certain mathematical arguments suggest: a logarithmic scheme

$$T_k := \frac{C}{\log k},$$

  for some constant C makes a good cooling schedule.

- The *depth* of a local minimum α is the size of the deterioration that is sufficient for leaving α.

- The initial temperature should be at least as large as the maximum depth of a local minimum.

  $\longrightarrow$ Choice of the constant C above.

- Such a scheme allows one to prove upper bounds on the convergence rate.

- For example, applied to the TSP one can prove that one reaches an optimal state with probability of at least $\frac{3}{4}$ after an exponential number of iterations.

Discussion of Simulated Annealing

- Simulated Annealing is quite a popular method.
- Why:
  - A first prototype is easy to implement.
  - Only little mathematical background is required from the programmer.
  - Has the potential to provide feasible solutions of high quality.
  - The name is "cool"!?
- Problems:
  - Often, reasonable solutions can only be achieved at the cost of an enormous computational effort.
  - No quality guarantees at all.
  - Typically, a lot of experimental work is needed to adjust the parameters for a particular problem.

Background of Simulated Annealing

- *Annealing*: In chemistry and chemical engineering the process of cooling heated material.
- The annealing should not produce cracks and fissures in the material.
- In physical terms:
  - Cracks and fissures mean that the remaining potential energy inside the material is high.
  - The material always assumes a local minimum of potential energy when cooled down.
- The warmer the material,
  - the higher the chances that cracks and fissures occur, but also
  - the higher the chances that those cracks and fissures are closed again.

    $\longrightarrow$ That the material escapes from a bad local minimum.
- The formula used for the probabilistic decision has originally been invented to describe physical processes like cooling.

Section 2.3.2:

Feature-based Local Search

What is feature–based local search?

- In typical optimization problems, the feasible solutions to an instance are formed by *features*.
- More specifically:
  - For an instance, there is a finite ground set of *features*.
  - The feasible solutions to this instance are certain subsets of this ground set.

    $\longrightarrow$ Selections of features.
- *Concrete examples of features*

  $\longrightarrow$ On the next few slides.
- *Remark*:
  - Features $\equiv$ dimensions: if the feasible solutions are elements of some space $\{0,1\}^n$, the n dimensions may be interpreted as features.
  - Here we follow the terminology from the literature on local–search algorithms and speak of "features" rather than dimensions.

Concrete examples of features:

- *TSP*:
  - Recall the TSP from Slide no. 5.
  - There the feasible solutions to a TSP instance on n points were encoded as certain $(n \times n)$–matrices x with 0/1–entries.
  - *Semantics*: $x[i,j] = 1$ means that point no. j immediately follows point no. i cyclically on the round tour encoded by x.
  - Then the features are the pairs (i,j) for $i, j \in \{1,\ldots,n\}$.
- *Matching*:
  - Recall the matching problem from Slide no. 32.
  - Here the edges of the input graph are the features.

Concrete examples of features (cont'd):

- *Set covering*:
  - *Input*: a finite ground set $\mathcal{F}$ and a collection $\mathcal{S}$ of subsets of $\mathcal{F}$.
  - *Feasible output*: a subset $\mathcal{S}' \subseteq S$ of S such that $\mathcal{F}$ is covered by $\mathcal{S}'$:

$$\forall\, x \in \mathcal{F}\ \exists\, s \in \mathcal{S}'\ :\ x \in s\,.$$

  - *Objective*: minimizing $\#\mathcal{S}'$.
  - *Features*: the elements of $\mathcal{S}$.

Concrete examples of features (cont'd):

- *Max–cut*:
  - *Input*: an undirected graph $G = (V, E)$ and an edge weighting $c : E \longrightarrow \mathbb{R}$.
  - *Feasible output*: any subset $W \subseteq V$.
  - *Objective*: maximizing $\sum_{e \in E_W} c[\{v, w\}]$ where $E_W = \{\{v, w\} \in E \,|\, v \in W, w \in V \setminus W\}$.
  - *Features*: the nodes (v selected as a feature means $v \in W$).

Concrete examples of features (cont'd):

- Coloring:
  - *Input*: an undirected graph $G = (V, E)$.
  - *Output*: an assignment $C : V \longrightarrow \mathbb{N}$ of a positive integral number ("color") to each node.
  - *Feasible*: if $C[v] \neq C[v]$ for all $\{v, w\} \in E$.
  - *Objective*: minimizing $\max\{C[v] \,|\, v \in V\}$.
- *Features*: pairs (v, n) such that $v \in V$ and $n \in \mathbb{N}$.
- *Remark*:
  - In principle, the set of features is infinite in this example.
  - However, obviously, $\max\{C[v] \,|\, v \in V\} \leq \#V$ for any optimal solution.
  - Consequently, the assumption that $\{1, \ldots, \#V\}$ is the (finite) ground set of colors does not reduce generality.

Choice of the feature set:

- *Note*: the features of the instances of a problem depend on the chosen problem formulation.
- *Example*: if the feasible solutions to a general TSP instance on n objects $o_1, \ldots, o_n$ are regarded as permutations of the objects,
  - then the canonical features are again pairs (i, j) with $i, j \in \{1, \ldots, n\}$,
  - but now (i, j) stands for o_i being at position j.
- In typical (yet not all) optimization problems,
  - each feature is assigned a *feature cost* and
  - the cost of a solution is the sum of the costs of the features that make up the solution.

Counter–example:

Question:

Are *all* neighborhood relations feature–based?

Answer:

Definitely not.

Abstract counter–example:

All algorithmic problems in which the number of feasible solutions to an instance may be infinite.

$\longrightarrow$ Not representable as subsets of a finite set.

Now feature–based local search:

- *Idea of feature–based local search*:
  - penalize certain features or
  - forbid changes of certain features

  to let the local search escape from a local optimum.

- In the following, we will briefly touch upon the most prominent examples of both options:
  - *Guided local search*:

    $\longrightarrow$ Penalizes certain features to drive them out of the current solution.
  - *Taboo search*:

    $\longrightarrow$ Forbids inserting/removing certain features in/from the current solution.

Useful terminology:

- Consider an instance I of an optimization problem.
- Again, let $\mathcal{F}_I$ denote the set of all features of the instance I.
- For a feature $x \in \mathcal{F}_I$ let $C[x]$ denote the feature cost.
- A feasible solution S to I can be identified with the set $F(S) \subseteq \mathcal{F}_I$ of the features that make up S
- In optimization problems in which the cost of a solution is the sum of the costs of the selected features, the cost of solution S may then be rewritten as

$$C[S] = \sum_{x \in F(S)} C[x] \,.$$

Guided local search:

- In principle, this is the general local–search scheme from Slide no. 48.
- *Crucial difference*: something different happens whenever the search runs into a local optimum (not just termination).
- Handling a local optimum:
  - The algorithm examines all features that make up the local optimum.
  - For each of these features a "utility of penalization" is determined.
  - One or more features with the highest "utility of penalization" are penalized.
  - The penalty is so large that the current solution is not a local optimum anymore.
  - Then the local search continues as usual.
- What does "penalized" and "utility of penalization" mean?

  $\longrightarrow$ On the very next slide.

"Penalized" and "utility of penalization":

- "*Penalized*":

  The feature cost of a feature is increased by some value (the *penalty*).

- "*Utility of penalization*":

  The "utility of penalization" of a feature is an estimation how promising it would be to penalize this feature.

- *Ideas* for such an estimation:
  - If the original cost value of a feature is high, it might be promising to drive it out of the solution by penalizing it.
  - On the other hand, if a feature has often been penalized and is again in the current solution, it might not be too promising to penalize it yet another time.
- How resolve this conflict?

  $\longrightarrow$ On the next slide.

Resolving the conflict:

- The approach on this slide is but an example.
- It is taken from the literature and often used in practice.
- For each feature $x \in \mathcal{F}_I$, the number of search steps in which x has been penalized so far is counted in an additional variable p_x.
- For the j–th penalization of feature x, a penalty value λ_{xj} is defined.
- The cost of a feasible solution S to I (incl. all penalties) is then

$$\tilde{C}[S] := \sum_{x \in F(S)} \left(C[x] + \sum_{j=1}^{p_x} \lambda_{xj} \right).$$

- The utility of penalization of feature $x \in F(S)$ is then defined as something like

$$\frac{\tilde{C}[x]}{1 + p_x}.$$

  $\longrightarrow$ Conflict resolved.

Taboo search

- This is another variant of the general local–search scheme from Slide no. 48.
- *Fundamental difference*:
  - Taboo search always moves on to the neighbor of minimal cost.
  - Unlike local search, it does so even in case the current solution is a local optimum.

    $\longrightarrow$ In such a case, the move step causes a deterioration.
  - In order to terminate the algorithm, an additional, external stopping criterion must be incorporated (e.g. termination after a certain number of steps).
- *Problem*: After escaping from a local minimum by a neighborhood step, the algorithm is very likely to return to this local minimum very soon.
- *Potential consequence* whenever this problem occurs: an infinite loop on a few feasible solutions.

The "taboo" in taboo search:

- To avoid such an infinite loop, the most recent changes of features are "kept in mind" by the search for a number of search steps.
- Change of a feature $x \in \mathcal{F}_I$ on the move from a feasible solution s_k to the neighbored s_{k+1} means:

  $$x \in F(s_k) \setminus F(s_{k+1}) \text{ or } x \in F(s_{k+1}) \setminus F(s_k).$$

- As long as a change is "kept in mind," the search must not undo this change.

  $\longrightarrow$ Must not drop a feature recently inserted nor re–insert a feature recently dropped.

- Typically (yet not exclusively),
  - a number k of steps is globally pre–defined and
  - each change is removed from the taboo list after k steps.

  $\longrightarrow$ The taboo "expires" after k steps.

Examples:

- *TSP*:
  If a pair (i, j) was inserted in (resp. removed from) the tour, it must not be removed (inserted) again until the taboo expires.

- *Coloring*:
  If a node v was de–assigned a color c, v must not be re–assigned c again until the taboo expires.

Aspiration:

- Practical experience with taboo search has shown that it is good
  - to occasionally drop all taboos, that is,
  - to continue the taboo search with a cleaned list of taboos (no more taboos "in mind").
- Clearly, the search immediately resumes recording new taboos after such a refresh.
- A condition that forces the taboo search to drop all taboos is called an *aspiration criterion* in the literature.
- *Typical example*: if the current solution is the best solution seen so far.
- Aspiration is but one (however, the most fundamental) improvement technique for taboo search.

Section 2.3.3:

Approaches of the Kernighan-Lin Type

What does that mean?

- Until now, we exclusively focused on approaches that realize the first idea on Slide no. 88:

  > Occasionally accept forward steps ... even if the new solution is worse.

- Next we turn our attention to the second idea:

  > Perform several forward steps and select the best ... result ...

- A first variant of the latter technique has been developed in the early seventies by two researchers, Kernighan and Lin.

General concept:

- Again, see Slide no. 48 for the terminology used on this slide.
- In each iteration of the while–loop in the local–search scheme, the search constructs a certain path in the neighborhood graph.
- This path starts with the current solution s^*.
- The best solution $s \neq s^*$ on this path is selected as the new current solution s^*.
- *Options* in case s is worse than the old s^*:
  - s is nonetheless accepted or
  - the algorithm terminates or
  - the algorithm tries to find another path.
- In the literature, the first option is typically applied (without any discussion of alternatives, in fact).

Construction of the path:

- Of course, the path is constructed incrementally, arc by arc, starting at s^*.
- Clearly, there are many good and bad strategies
  - to choose the next arc in each step of this incremental construction and
  - to terminate the construction of the path.

Standard procedure from the literature:

- The strategy (for both aspects simultaneously) in the literature is based on ideas very similar to taboo search.
- Therefore, it applies to feature–based optimization problems in the first place.

Details of the standard procedure:

- A taboo list is maintained (and permanently expanded) throughout the construction of the path.
- At the beginning of the path construction, the set of taboos is empty.
- During the path construction, taboos are never removed from the set.
- An arc in the neighborhood graph whose insertion would violate the taboos is regarded as temporarily infeasible.
- In each step of the path construction, the best feasible arc is chosen.
- The construction of the path terminates as soon as it arrives at a node of the neighborhood graph all of whose out–going arcs have become infeasible.

Examples:

- *TSP*:
  - Consider the neighborhood structure visualized on Slide no. 45.
  - Whenever an arc is removed from the round tour, its re–insertion is becoming a taboo.
- *Max–cut*:
  - From Slide no. 107 recall the max–cut problem.
  - Various neighborhood structures could be defined based on inserting nodes into W and removing nodes from W.
  - In any such case, we could taboo the re–insertion of a removed node and the removal of an inserted node.

Kernighan-Lin: what's in a name?

- Actually,
  - these two guys never described the approach in full abstract generality,
  - but only presented concrete instances of this technique for two concrete problems: TSP and max–cut.
- These two instances are commonly called the *Kernighan-Lin algorithm* and the *Lin-Kernighan algorithm* in the literature.
- The term, "approaches of the Kernighan–Lin type," is not common in the literature.
- It is chosen in this lecture in honor of these two pioneers of heuristic algorithms.

Iterated/Chained Local Search

- This variant is based on another, anonymous algorithm ("black box").
- The black box gets a feasible solution as an input and delivers a feasible solution as its output.
- In general, the black box is a heuristic: the output
  - should be better than the input,
  - but it may be even worse,
  - or it may even be identical with the input.

  $\longrightarrow$ Any variant of the local–search scheme may serve as a black box.
- In each iteration of the loop (cf. Slide no. 48),
  - first an arbitrary neighbor s of s^* is chosen,
  - then the black box is applied to s.
- The result $\tilde{s}$ of the black box is the new current solution s^*.

Options

in case $\tilde{s}$ is worse than the old s^*:

- $\tilde{s}$ is nonetheless accepted or
- the algorithm terminates or
- the algorithm applies the black box to another neighbor of s^*.

$\longrightarrow$ Perfectly analogous to Kernighan–Lin type approaches (cf. Slide no. 123).

Rationale of Section 2.3.3

- Again, the problem of local optima is addressed by these techniques.
- The path may escape a "pit" around a local optimum.
- Even if the best solution on the path is worse than the starting point.

Section 2.3.4:

Population-Based Approaches

Multi-start Local Search

- All variants of local search considered so far only apply one run of the search.
- *Problem*:
  - Chances are high that the search will never leave a small subspace of the solution space.
  - The really good feasible solutions may be somewhere else in the search space.
- *Simplest imaginable idea*:
  - Generate a set of feasible solutions (e.g. randomly).
  - Start a local search (or simulated annealing, taboo search, whatever) from each of them.
  - Deliver the best feasible solution seen by any of these searches.

Multi-start and population-based

- Multi-start is the simplest population-based strategy: a population of independent individuals, no competition, no cooperation.

- In the following, we will consider several models:
  - Competition: survival of the fittest (evolution strategies).
  - Recombination (genetic algorithms).
  - "Social" collaboration (ant colony).

Survival of the fittest

- Generate a set of feasible solutions much like on the last slide.
- Like in multi-start, start a local search from each of them.
- *Difference*:
  - All of these searches are performed in *rounds*'' : simultaneously in a pseudo–parallel fashion.
  - Always the better searches survive, and the best ones produce ‘‘off-spring'' .
- Details of rounds: next slide.

One round:

- Each of the searches performs one move step in the neighborhood graph.
- Then the current solutions (the *population*) of all of these searches are compared with each other.
- The searches with the worst current solutions are dropped.
- Each of the best current solutions S is "forked" into at least two searches.

  $\longrightarrow$ Start solution(s) of the new search(es) (a.k.a. *offspring*): feasible solution(s) neighbored to S.

Evolutionary algorithms

- Also known as *evolution strategies*.
- This is the most prominent variant of "survival of the fittest".
- It aims at simulating asexual biological evolution:
  - "Fit" members of the population are able to reproduce by duplication.

    $\longrightarrow$ The "fitter" the member, the higher its probability of duplicating.
  - The offspring are randomly chosen from the "parents" neighborhood.
  - Members of the population are randomly killed by the circumstances.

    $\longrightarrow$ The "fitter" the member, the smaller its probability of being killed.

Evolutionary algorithms (cont'd):

- Like "survival of the fittest" (but unlike biological evolution), the process is organized in rounds.
- *One round*:
  - A certain number of members of the population are selected randomly with a probability that is monotonously *in*creasing with their cost values.
  - These members of the population are dropped.
  - Another number of members of the population are selected randomly with a probability that is monotonously *de*creasing with their cost values.
  - These members of the population produce offspring.
  - Each member of the new population
    - is mutated randomly like in "survival of the fittest,"
    - however, not at all odds,
    - but only with a certain (typically *very* small) probability.

Brief excursion: Search on representations

- So far, we have essentially considered the case that local–search techniques are based on
  - the solution space of the input instance and
  - a neighborhood on this space.
- However, it is also possible to base this sort of techniques on
  - an *abstract representation* of the feasible solutions and
  - a neighborhood structure on these abstract representatives.
- *Prerequisites*:
  - Each abstract representative must be associated with exactly one feasible solution.
  - The feasible solution associated with a given abstract representation must be efficiently computable.

Examples from this lecture:

- Semi–definite programming (Slides nos. 61...):
  - There the local search was based on subsets of the constraint set instead of feasible solutions.
  - These abstract representatives are associated with unique feasible (or even infeasible!) solutions.
- *Later on*: the simplex algorithm for linear programming.

Note:

- The association between abstract representatives and (in)feasible solutions is not necessarily one–to–one.
- It is not required that each (in)feasible solution be associated with an abstract representative.

Abstract representatives in biological evolution:

- Each individual living being is (more or less!) a product of its *genes*.

  $\longrightarrow$ In some sense, the genes of an individual are its abstract representation ("translated" into a living individual in a fuzzy, randomized fashion).

- From a *very* abstract point of view, the genes of a living individual may be viewed as one long string over an alphabet of four letters.

- Biological evolution may be viewed as proceeding solely on the genes:

  $\longrightarrow$ The genes are the true entities that mutate, replicate, and (in sexual reproduction) combine.

- For two living individuals that may produce offspring together sexually, these strings are (nearly) equally long.

"Evaluating" the genes:

- Each individual is a trial at its genes:

  $\longrightarrow$ How well does an individual that carries these genes?

- In other words: Letting the individual struggle for life is much like
  - evaluating an objective function for its abstract representation (genes) and
  - deciding upon its "survival" through a random decision with a probability that is monotonously increasing in the value of the objective function (cf. Slide no. 137).

- The space of biologically possible individuals is certainly much larger than the number of possible genes.

  $\longrightarrow$ Each abstract representation corresponds to a feasible solution (but not necessarily vice versa).

Genetic algorithms:

- Conceptually, genetic algorithms are very similar to evolution strategies:
  - A search proceeding in rounds.
  - A population (of genes!) is maintained.
  - In each round, some members of the population are killed with a probability that is monotonously *de*creasing in the fitness of the individual.

  $\longrightarrow$ Cf. Slide no. 136.

- *Main difference*:
  - In evolution strategies, a new generation ("child generation") is generated from selected members of the previous generation ("parent generation") by means of (asexual) mutation.
  - In genetic algorithms, each member of the child generation is generated from two members of the parent generation by means of (sexual) *recombination*.

Genes in genetic algorithms:

- As said above, the search does not take place on the feasible solutions themselves but on abstract representatives.
- Like in biology, the abstract representative of a feasible solution is called its *gene*.
- For each instance, all abstract representatives will be strings of the same length over the same set.
- The set may be discrete or real-valued, whatever.
- Frequent example: $\{0, 1\}$.

Note:

The usage of biological terms such as *gene* and *chromosome* is not perfectly standardized in the literature on genetic algorithms.

Example I of genes: feature–based problem definitions

- Recall feature–based problem definitions from Slides nos. 105 ff.
- Consider a fixed instance of some feature–based problem and let n denote the number of features for this instances.
- Encode each feasible solution s to this instance as the 0/1–vector x_s of length n such that for $i \in \{1, \ldots, n\}$:

$$x_s[i] = 1 \iff i \text{ is selected in } s.$$

- This encoding of an instance is an example of genes.

Example II of genes: TSP

- From Slide no. 106 recall that the TSP can be modeled in a feature–based fashion.
- However, we have seen that the feasible solutions to a TSP instance on n cities may also be encoded as the permutations of $(1, \ldots, n)$.

  $\longrightarrow$ Such a representation may be viewed as a string of length n over the alphabet $\{1, \ldots, n\}$.
- Permutations can alternately be encoded in a binary fashion using *permutation matrices:*
  - For n cities, this is an $(n \times n)$–matrix X of 0/1–entries.
  - For $i, j \in \{1, \ldots, n\}$, $X[i, j] = 1$ means that i is the j–th city to be visited.

  $\longrightarrow$ Arises from the identity matrix by permuting the rows or columns.

Example III of genes: coloring

- From Slide no. 109 recall the definition of the coloring problem.
- Also recall that, without loss of generality, the number of colors may be restricted to the number n of nodes of the graph.
- Therefore, a feasible (or infeasible) solution may be encoded as a string of length n over the alphabet $\{1, \ldots, n\}$.
- Alternatively, a feasible (or infeasible) solution may be encoded in a binary fashion through an $(n \times n)$–matrix X:

$$X[i,j] = 1 \iff \text{node no. } i \text{ is assigned color no. } j.$$

$\longrightarrow$ Much like in the example of TSP from the last slide.

Recombination:

Typically, recombination is realized as follows in genetic algorithms:

- two offspring are simultaneously produced from a pair of parents,
- both offspring are produced by the same (deterministic or randomized) procedure, but
- the roles of the two parents are exchanged in the production of the two offspring.

$\longrightarrow$ In total contrast to the biological rules for producing offspring.

Terminology:

Recombination is often called *crossover* or *crossing–over* both in molecular biology and in genetic algorithms.

Quality of a crossover strategy:

- Since the selection of the parents is based on their degrees of fitness, the offspring of a pair of parents should preferably resemble their parents.

  $\longrightarrow$ If the parents are fit, chances are high that the offspring are fit, too.

- For example, in feature–based problems:

  – If a feature is selected in both parents, it should also be selected in the offspring.

  – If a feature is selected in neither parent, it should not be selected in the offspring, either.

- Moreover, an offspring should not essentially inherit from one parent alone but from both parents.

  $\longrightarrow$ Should not be very similar to one parent and, simultaneously, very different to the other parent.

Notation:

- Consider an instance and let n denote the length of the genes.
- Let P_1 and P_2 denote two members of the parent generation.
- From them, two offspring O_1 and O_2 are generated.

Fundamental examples of crossover strategies:

- *One–point crossover*:
  - A position $p \in \{1, \ldots, n-1\}$ is chosen according to an arbitrary selection rule.
  - $O_1[i] := P_1[i]$ and $O_2[i] := P_2[i]$ for $i \in \{1, \ldots, p\}$.
  - $O_1[i] := P_2[i]$ and $O_2[i] := P_1[i]$ for $i \in \{p+1, \ldots, n\}$.
- *Two–point crossover*:
  - Two positions $\ell, r \in \{1, \ldots, n\}$, $\ell < r$, are chosen according to an arbitrary selection rule.
  - $O_1[i] := P_1[i]$ and $O_2[i] := P_2[i]$ for $i \in \{1, \ldots, \ell-1, r+1, \ldots, n\}$.
  - $O_1[i] := P_2[i]$ and $O_2[i] := P_1[i]$ for $i \in \{\ell, \ldots, r\}$.
- *Uniform crossover*: For each $i \in \{1, \ldots, n\}$,
  - $O_1[i] := P_1[i]$ and $O_2[i] := P_2[i]$ with some probability,
  - $O_1[i] := P_2[i]$ and $O_2[i] := P_1[i]$ otherwise.

General problem:

- The result may be the abstract representative of an **in**feasible solution.
- One possible strategy to overcome this problem:
  - Make all solutions feasible by dropping all side constraints.
  - As a surrogate, penalize the (degree of) deviation from feasibility.
- *Alternative ideas*:
  - Re–define the genetic representation of solutions such that reasonable crossover strategies will produce feasible offspring from feasible parents.
  - Apply the crossover and repair the result afterwards.

$\longrightarrow$ Examples on the next few slides.

Example of "Re-define ... feasible offspring from feasible parents"

- The core of typical scheduling problems is as follows:
  - *Input*: a finite set of item, $a_1, \ldots, a_n$, a duration d_i for each item a_i, and a selection S pairs (a_i, a_j) such that the graph with node set $\{a_1, \ldots, a_n\}$ and these selected pairs as directed arcs is acyclic.
  - *Output*: an assignment of each item a_i to a time $t_i \geq 0$.
  - *Constraint*: $t_i + d_i \leq t_j$ for each $(a_i, a_j) \in S$.
  - *Objective*: minimize $\max_i t_i$.
- *Interpretation*:
  - A large process or project is broken down into indivisible tasks a_i.
  - Certain tasks must be delayed until certain other tasks are finished.
  - The total duration of the project or process is to be minimized.

Example (cont'd)

- For each item a_i, the genetic representation contains a real number x_i.
- *Semantics*: $x_i = \min\{t_i - (t_j + d_j) \,|\, (a_j, a_i) \in S\}$.
- Obviously, the feasible solutions correspond in a one-to-one fashion to the *nonnegative* genetic representations.
- For this space of genetic representation, any crossover strategy is appropriate that transforms nonnegative representation into nonnegative ones.
- *Examples*: the offspring are constructed by
  - exchanging elements of the parents,
  - taking the element-wise minimum/maximum of each element of the parents,
  - taking (weighted) average values of the parents' elements.

Example of crossover/repair: PMX for the TSP

- This is a particular crossover–and–repair strategy tailored to the TSP.
- What's in a name: PMX = partially mapped crossover.
- The crossover strategy used is two–point crossover.
- Again, let $[\ell, \ldots, r]$ denote the positions of the exchanged substring.
- What needs to be repaired in O_1 after the crossover:
  Cities may appear twice in O_1,
  - once in $\{\ell, \ldots, r\}$ and
  - once in $\{1, \ldots, \ell - 1, r + 1, \ldots, n\}$.

  $\longrightarrow$ Unavoidably, some other cities do not appear at all in this case.

PMX (cont'd):

- On the next few slides, we will only consider the procedure for O_1.

  $\longrightarrow$ The procedure for O_2 is mirror–symmetric, with the roles of P_1 and P_2 exchanged.

- *Repair loop:* While some city appears twice, the following procedure is applied for repair:

  - Let $i \in \{\ell, \ldots, r\}$ and $j \in \{1, \ldots, \ell - 1, r + 1, \ldots, n\}$ such that $O_1[i] = O_1[j]$.

  - Set $O_1[j] := P_1[i]$.

Number example:

			ℓ					r			
P_1:	1	2	3	4	10	5	6	11	7	8	9
P_2:	5	4	6	9	11	2	1	10	7	8	3

$\Downarrow$ Two–point crossover

O_1:	1	2	6	9	11	2	1	10	7	8	9
O_2:	5	4	3	4	10	5	6	11	7	8	3

$\Downarrow O_1[1] := P_1[7]$

O_1:	6	2	6	9	11	2	1	10	7	8	9

$\Downarrow O_1[2] := P_1[6]$

O_1:	6	5	6	9	11	2	1	10	7	8	9

$\Downarrow O_1[1] := P_1[3]$

O_1:	3	5	6	9	11	2	1	10	7	8	9

$\Downarrow O_1[11] := P_1[4]$

O_1:	3	5	6	9	11	2	1	10	7	8	4

Analysis of PMX:

- As this example shows, it may be necessary to repair a position more than once.
- Clearly, the repair loop only terminates when O_1 becomes feasible.
- Thus, it suffices to prove that the repair loop indeed terminates.
- For an easier exposition, we will consider an auxiliary directed graph $G = (V, A)$ with $V = \{1, \ldots, n\}$ and for all $i, j \in V$: $(i, j) \in A$ if, and only if, there is $h \in \{\ell, \ldots, r\}$ such that $P_2[h] = i$ and $P_1[h] = j$.
- In the example from the last slide:

			ℓ					r			
P_1:	1	2	3	4	10	5	6	11	7	8	9
P_2:	5	4	6	9	11	2	1	10	7	8	3

$\longrightarrow (6,3), (9,4), (11,10), (2,5), (1,6), (10,11) \in A$.

Analysis of PMX (cont'd):

- Each iteration of the repair loop replaces i by j for some $(i, j) \in A$.
- Repairing a position more than once means proceeding along some path of G.
- Clearly, this path starts with one of the nodes $P_1[1], \ldots, P_1[\ell - 1]$, $P_1[r + 1], \ldots, P_1[n]$.
- Therefore, the repair loop terminates unless it proceeds along a cycle of G.
- However, for each node $i \in V$ in a cycle of G, it is $i \in \{P_1[\ell], \ldots, P_1[r]\}$ and $i \in \{P_2[\ell], \ldots, P_2[r]\}$.
- Obviously, each node is entered by at most one arc.
- In summary, a node i on a cycle of G cannot be reached from any of the nodes $P_1[1], \ldots, P_1[\ell - 1]$, $P_1[r + 1], \ldots, P_1[n]$.

		ℓ					r			
1	2	3	4	10	5	6	11	7	8	9
5	4	6	9	11	2	1	10	7	8	3

$\longrightarrow (6, 3), (9, 4), \underline{(11, 10)}, (2, 5), (1, 6), \underline{(10, 11)} \in A.$

Problem–specific crossover strategies

- In PMX,
  - the crossover strategy was one of the general ones (viz. two–point crossover),
  - but the repair strategy was specific for problems in which the feasible solutions are permutations.
- In the following, we will consider an example of strategies that are problem–specific from the very beginning.
- Again, we will consider a strategy for the TSP as an example.
- This exemplary strategy is called *order crossover* (OX).
- This strategy may also be viewed as a variant of two–point crossover.
- As before, let P_1 and P_2 denote the two parents.
- Again, we will only consider the construction of one offspring, O_1.

Concrete strategy:

- Like in two–point crossover, positions ℓ, $r \in \{1, \ldots, n\}$, $\ell < r$, are chosen according to some selection rule.
- Then $O_1[i] := P_1[i]$ for all $i \in \{\ell, \ldots, r\}$.
- For all $i \in \{1, \ldots, \ell-1, r+1, \ldots, n\}$, the values $O_1[i]$ are defined such that the result O_1 is indeed a permutation of $\{1, \ldots, n\}$.
- *Contribution of* P_2:

  The values $O_1[1], \ldots, O_1[\ell-1], O_1[r+1], \ldots, O_1[n]$ are defined such that their relative order is identical to their relative order in P_2.
- *More specifically*:
  - Let $i_1, i_2 \in \{1, \ldots, \ell-1, r+1, \ldots, n\}$.
  - Let $j_1, j_2 \in \{1, \ldots, n\}$ such that $O_1[i_1] = P_2[j_1]$ and $O_1[i_2] = P_2[j_2]$.
  - *Identical relative order*: If $i_1 < i_2$, then $j_1 < j_2$.

Mutation in genetic algorithms:

- Each round of a genetic algorithm comprises the following steps:
  - A certain number of pairs of parents is selected for reproduction.
  - The pairs are combined to form offspring.
  - With a small probability, such an offspring is then mutated like in evolution strategies.
- *Rationale* of the additional mutation step:
  - Simulates the biological procedure more precisely.
  - Often seems to have a positive effect on the outcome of the genetic algorithm.

Concluding remarks on genetic algorithms:

- There is a body of mathematical theory for genetic algorithms.
- This theory is mainly based on statistics and on mathematical biology.
- In particular, there are some general, non–quantitative results about the convergence towards the "good" feasible solutions.
- However, to the best of the lecturer's knowledge,
  - the model assumptions are highly unrealistic,
  - and thus the results cannot give more than some evidence for practice.

  $\longrightarrow$ Much like in the case of simulated annealing (Slides nos. 97 ff.).

"Cultural algorithms"

- In evolution strategies, the only "social activity" among the members of the population is competition.
- In genetic programming, the social activity is much more collaborative.
- This point of view inspired population–based approaches that simulate collaborative strategies of social actors.
  $\longrightarrow$ So–called *cultural algorithms*.
- This means:
  - The individual members of the population react on each other in a positive or negative fashion.
  - The more promising a member's way to go, the "more positive" the reactions of other members.
- Since social collaboration is very complex and not at all understood, simple models of social life are taken as patterns.
- *Example*: The collaboration of ants in an ant colony.
  $\longrightarrow$ See the next few slides.

Ant–Colony Optimization

- Ants quickly converge to the shortest and most convenient paths from sources of food and material to the nest.
- Apparently, no steering intelligence is behind this "automatic procedure".
- Instead, the steering mechanisms are quite primitive:
  - Each ant lays out a trail of a certain chemical substance (a certain *pheromone*).
  - Each ant tries to follow the most intensive trail.
  - The shorter and more convenient a path, the higher the probability that a "randomly strolling" ant has taken it before.

  $\longrightarrow$ A tendency towards the short, convenient paths.
- Since ants make mistakes (even ants are "only human"), the intensity of a trail only gives the *probability* that an ant follows this trail.

  $\longrightarrow$ Important to avoid local optima!

Simplified model scenario:

- Let there be only two paths from a nest A to a source S of food.
- For simplicity, let these two paths not meet between A and S (internally disjoint).
- Let there be no other paths in the scenario.
- Suppose one of the paths is twice as long as the other path.
- Suppose $2n$ ants leave the nest in search for food.

  $\longrightarrow$ On average, n ants take either path.

- The ants that take the shorter path are earlier at S.

  $\longrightarrow$ At that moment, only on the shorter path are there pheromone trails.

  $\longrightarrow$ Due to the trails, those ants will return on the shorter path with high probability.

  $\longrightarrow$ When the ants from the longer path are going to return, the pheromone trail on the shorter path is much more intensive than on the longer path.

Positive and negative feedback:

- *Positive feedback*: Each ant stimulates other ants to follow it.
- *Negative feedback*:
  - The amount of pheromone on a trail decreases ("evaporates") permanently.
  - Clearly, the next ant on this trail will refresh it.
  - Thus, if a trail is not used very often, its probability of being used decreases even more.

Translation into an algorithm:

- The problem is to be re–formulated such that
  - each instance corresponds to a directed graph,
  - each solution to an instance corresponds to a path in the graph,
  - side constraints of the problem occur as side constraints on paths, and
  - the cost function can be (at least approximately) expressed as the length of the corresponding path.
- *Idea*:
  - The ants move around in this graph (obeying the above–mentioned side constraints).
  - The mechanism of pheromones is simulated.
  - Then, hopefully(!), the ants will tend to move along short paths.

Example I: TSP

- Consider a general TSP instance with n points.
- At the beginning, a number of ants is placed in an arbitrary (random or sophisticated) distribution on the n points.
- Like in the previous algorithms (but unlike animal behavior), the algorithm proceeds in rounds.
- More specifically,
  - the algorithm comprises an arbitrary number of *major* rounds,
  - and each major round consists of n *minor* rounds.
- In each round, each ant moves on to a point that it has not yet visited during the current major round.

  $\longrightarrow$ In each major round, each ant completes a round tour on the n points.
- The best tour completed by any ant is the overall output of the algorithm.

Ant–colony algorithm for the TSP (cont'd):

- Whenever an ant moves on from one point i to another point j, it leaves some amount of pheromone on the arc (i, j).
- To simulate negative feedback, the amount of pheromone decreases ("evaporates") after each minor round by a constant factor.
- The point to be visited by an ant in the next minor round is chosen randomly (from the points not yet visited in this major round).
- The probability of each candidate point j depends
  - on the distance $d(i, j)$ from the currently visited object i to j and
  - on the amount of pheromone currently on the arc (i, j).

  $\longrightarrow$ The smaller the distance and the larger the amount of pheromone, the higher the probability of taking this arc.

Example II: set covering

- From Slide no. 107 recall the set–cover problem.
- We will here use the terminology from there.
- *Re–formulation* as a path problem in a directed graph:
  - The nodes of the graphs are the elements of $\mathcal{S}$.
  - The graph is complete, that is, for any $s_1, s_2 \in \mathcal{S}$, the arc (s_1, s_2) exists.
  - (Ordered) subsets of $\mathcal{S}$ can then be viewed as paths in this graph.
  - The weight of an arc (s_1, s_2) shall express the "benefit" of adding s_2 to the solution given that s_1 is part of the solution.

    $\longrightarrow$ For example, $\#(s_2 \setminus s_1)$ could express this benefit.
- An ant must not stop walking through the graph until the elements of $\mathcal{S}$ on this path together cover $\mathcal{F}$.

Section 2.3.5:

Local Search

under

Complex Side Constraints

Problem

- All variants of the general local–search scheme depend on an appropriate neighborhood structure in some way or other.
- Examples:
  - The fundamental local–search algorithm requires an easily enumerable neighborhood structure.
  - Simulated annealing requires a neighborhood structure in which random selection is easy.
- Unfortunately, the side constraints often make the natural neighborhood definitions inappropriate.

  $\longrightarrow$ Example on the next slides.

Example: TSP

- The neighborhood sketched on Slide no. 45 is appropriate: any pair of arcs of the current tour induces a neighbored feasible solution.
- However, the TSP does not seem to occur very often in its purist form in reality.
- In fact, real–world variants of the TSP typically come with additional side constraints.
- Typical example:
  - A list of pairs (i, j) is given as an additional input.
  - For each such pair (i, j), object no. i must be visited before object no. j (*precedence constraints*).
- *Consequence*:
  - Removing two arbitrary arcs from a tour and re–connecting the tour like on Slide no. 45 may result in an infeasible solution.
  - Unless the list of pairs is very short, the probability of infeasibility is very high.

Problem summary:

- The neighborhood relation may become very sparse (maybe even disconnected).
- *Consequences*:
  - Due to the loose neighborhood connections, the search is likely to stay in a small subset of the search space (around the start solution).
  - Chances are high that the search quickly traps into a local optimum and cannot leave it anymore.
- Additional problem for algorithms such as simulated annealing:

  Such an algorithm may require many trials of neighbored, but infeasible, "solutions" until a *feasible* neighbored solution is found.

Typical approach:

- Some of the side constraints are relaxed, that is:
  - The selected side constraints are dropped.
  - Additional penalty terms in the cost function penalize violations of the dropped side constraints.
- If the side constraints to be dropped are well selected, the natural neighborhood relations will again be appropriate.
- Natural approach to the example (TSP with precedence constraints):
  - The precedence constraints are dropped.
  - The number of input pairs (i, j) such that no. j is visited before no. i is added to the cost function (possibly multiplied by some weighting factor).

Variable weight of the penalties:

- The rough penalty terms is to be multiplied by a (nonnegative) *penalty factor*, which expresses the relative priority of the penalty compared to the original objective.
- Clearly, this factor need not be constant throughout an application of any kind of local–search technique (simulated annealing, evolution strategies, ...).
- *Natural scheme*:
  - In the beginning, the factor is very small or even zero.

    $\longrightarrow$ The side constraints are dropped, (more or less) without a substitute.
  - Throughout the procedure, the factor is increased from time to time.
  - At the end, the factor is very large.

    $\longrightarrow$ The problem becomes (approximately) a pure feasibility problem.

Heuristic idea behind variable weights:

- Chances are high that the objective function is rather "smooth" on the set of all feasible and infeasible solutions.

  $\longrightarrow$ Need not be equally smooth on the feasible solutions alone!

- In this case, the good *feasible* solutions are probably found in the areas where the objective function is generally good on feasible and infeasible solutions.

- Therefore, it might be a good idea,

  – first to approach these areas quickly (disregarding feasibility) and

  – enforcing feasibility only later on.

- It might even be a better idea to increase the force on feasibility gradually throughout the search.

  $\longrightarrow$ Exactly the scheme formulated on the last slide.

Natural translation into an algorithmic goal:

- In the first couple of rounds of the procedure, the search shall converge to the areas of the ground set where the objective function is quite good.

  $\longrightarrow$ Optimization is more important than feasibility.

- In the last couple of rounds, the search shall converge to feasibility.

  $\longrightarrow$ Feasibility is more important than optimization.

General problem with relaxation:

- The feasible solutions to the original problem are among the very (very!) good solutions with respect to the new objective function, that is,
  - the original objective function
  - plus some penalty terms.
- Experience seems to suggest that the various local–search algorithms presented here
  - indeed converge to the very good solutions,
  - however, often at a miserable convergence rate.
- *Heuristic consequence*:
  - Chances are high that such a search procedure requires a lot of time until the very first feasible solution is seen.
  - However, if the algorithm indeed finds a feasible solution, this is probably quite good.