Design and Application of Modern Heuristics

Mainz, September 2012

Franz Rothlauf

- These slides have been the basis for a class on the design and application of modern heuristics given at the university of Mainz in fall 2012
- The slides are based on the book "Design of Modern Heuristics" published at Springer
- You are free to use the material contained in these slides for your own classes or presentations. A reference to the book would be nice.
- Many thanks to Jörn Grahl who contributed to preparing the slides. For editing some of the latex displays, we used texpoint.
- Enjoy modern heuristics!

Agenda

Intro

- Round of introductions
- Purpose of class
- Organisational Issues

Grading and Participation

- First two days: Lecture

- Third day: Problem presentations and discussion of particular problems (see workshop literature)

- Fourth day: Discussion and Development of enhanced methods

Class Background Literature

- **Rothlauf, F. (2011):** *Design of Modern Heuristics: Principles and Application.* **Springer, Berlin**

- 2.1 Kallel, L.; Naudts, B.; Reeves, C. R. (2001): *Properties of fitness functions and search landscapes.* Theoretical aspects of Evolutionary Computing 2001, Springer, London, 175-206.
- 2.2 He, J.; Reeves, C.; Witt, C.; Yao, X. (2007): *A Note on Problem Difficulty Measures in Black-Box Optimization: Classification, Realizations and Predictability.* Evolutionary Computation 15(4): 435-443.
- 2.3
  - Wolpert, D. H.; Macready, W. G. (1997): *No Free Lunch Theorems for Optimization.* IEEE Transactions on Evolutionary Computation 1(1): 67-82.
  - Christensen, S.; Oppacher, F. (2001): *What can we learn from No Free Lunch? A first attempt to characterize the concept of a searchable function.* Proceedings of the Genetic and Evolutionary Computation Conference 2001, Morgan-Kaufman, San Francisco CA, 1219-1226.
  - Koehler, G. J. (2007): *Conditions that obviate the No Free Lunch Theorem.* INFORMS Journal on Computing 19(2): 273-279
- 5. Coello Coello, C. A. (2002): *Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art.* Computer methods in applied mechanics and engineering (191) 2002: 1245-1287.
- 7.
  - Blum, C.; Puchinger, J.; Raidl, G.; Roli, A. (2011): *Hybrid metaheuristics in combinatorial optimization: A survey.* Applied Soft Computing 11 (2011) 4135-4151.
  - Blum, C.; Roli, A. (1993): *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison.* ACM Computing Surveys 35(3): 268-308.

Selected Workshop Literature

- Debels, D.; Vanhoucke, M. (2007): A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem. In: *Operations Research* 55 (3), S. 457–469.

- Ropke, S.; Pisinger, D. (2006): An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. In: *Transportation Science* 40 (4), S. 455–472.

- Schittekat, P.; Sorensen, K. (2009): OR Practice--Supporting 3PL Decisions in the Automotive Industry by Generating Diverse Solutions to a Large-Scale Location-Routing Problem. In: *Operations Research* 57 (5), S. 1058–1067.

- Sung-Soon Choi; Byung-Ro Moon (2008): Normalization for Genetic Algorithms With Non-synonymously Redundant Encodings. In: *IEEE Transactions on Evolutionary Computation* 12 (5), S. 604–616.

Background

Performing Experimental Studies

- Bartz-Beielstein, T. (2006): *Experimental Research in Evolutionary Computation - The New Experimentalism.* Springer-Verlag, Berlin, Heidelberg, New York.

Publishing: what and how?

- Schrader, U.; Hennig-Thurau, T. (2009): *VHB-JOURQUAL2: Method, Results, and Implications of the German Academic Association for Business Research's Journal Ranking.* BuR - Business Research 2(2): 180-204.

Writing a Review:

- Lee, A. S. (1995): *Reviewing a manuscript for publication.* Journal of Operations Management 13 (1995) 87-92.

left intentionally blank

Modern Heuristics

1. Heuristics
   a. Construction Heuristics
   b. Improvement Heuristics
2. Approximation Algorithms
3. Modern Heuristics (Metaheuristics)

History

- Until 1970s: mostly exact optimization

- Many practical problems are NP-complete

$\rightarrow$ exact approaches have exponential running time

- Idea: relax optimality, increase efficiency

   $\rightarrow$ Heuristics

Construction heuristics

- A.k.a. single-pass heuristics
- Build solution from scratch
    - Several steps
    - Fix one part of solution per step
    - Often: fix one decision variable per step
- Terminate when solution is complete
- No improvement steps

Improvement heuristics

- **Start from a complete solution**

- **Improve solution**

    - Several steps

    - Possible changes define a „neighborhood"

    - No diversification: objective value increases

    - If no improvement is possible: terminate

Greedy search

- Construction & improvement heuristics
  are often greedy

- Choose alternative with highest objective value

- No looking ahead, myopic, fast, sub-optimal

Example: Travelling salesman problem

- **Connect n cities with minimal total distance**
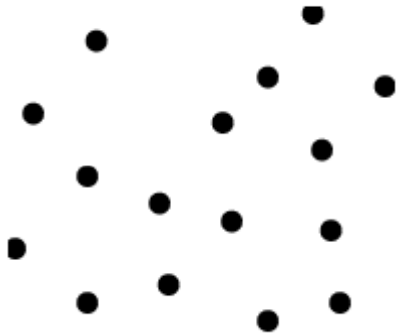
Data
•node location
•distance weights

Optimal solution
•path

Construction heuristics for TSP (1)

- **Nearest neighbor (Rosenkrantz et al, 1977)**
  - Start with random city
  - Connect nearest unconnected city
  - Terminate when all cities are connected
  - Although an upper bound $l(T)/l(T_{opt}) \leq (\log_2 n)/2$ exists on solution quality, it does not perform well in practice

Construction heuristics for TSP (3)

- **Nearest insertion**
  - Start with a random two-city tour
  - Select city with minimal distance to any connected city
  - Add city in a way that minimizes increase of tour length
  - Worst case performance: $l(T)/l(T_{opt}) \leq 2$

- **Cheapest insertion**
  - Like nearest insertion, but chooses city that increases tour length the least
  - Worst case performance: $l(T)/l(T_{opt}) \leq \log_2 n$

Construction heuristics for TSP (3)

- **Furthest insertion**
  - Start with longest two-city tour
  - Iteratively add city that increases tour length the most when inserted it in the best position on the current tour
  - Idea: start with cities that are far apart
  - Worst-case performance is $l(T)/l(T_{opt}) \leq \log_2 n$ (like cheapest insertion), but furthest insertion outperforms the other construction heuristics in practice.

Improvement heuristics for TSP (1)

- **Two-opt**
  - Remove any two possible edges and obtain two subtours. Insert two new edges such that resulting tour length is minimal.
  - If distances are Euclidean, no edges in resulting tour do cross (on-crossing tour)
  - Worst case performance $O(4\sqrt{n})$

- k**-opt (Lin, 1965)**
  - Generalization of 2-opt. Examine some or all $\binom{n}{k}$ k-subsets of edges in a tour
  - If exchange of k edges does not improve tour, tour is k-optimal
  - If triangle inequality holds, worst case performance of

    k-opt is $O\left(\frac{1}{4} n^{\frac{1}{2k}}\right)$ (Chandra et al, 1994)

Approximation Algorithms

- **Heuristics substitute optimality by tractability**

- **Approximation algorithms are heuristics with a quality bound**

- **Performance is measured by approximation ratio**

$$\rho(n) \geq \max\left(\frac{f(x^{approx})}{f(x^{\star})}, \frac{f(x^{\star})}{f(x^{approx})}\right)$$

  n **is problem size,** x^{approx} **is solution returned by algorithm, and** x^{*} **is optimal solution**

- **Definition holds for minimization and maximization**

Understanding approximation ratio

- An algorithm has an approximation ratio $\rho(n)$ if for any input size n the objective value of the returned solution is within a factor of $\rho(n)$ of the optimal objective value.

- If an algorithm always returns the optimum $\rho(n)=1$

- If algorithm returns solution that is never worse than $2f^*$, then $\rho(n)=2$

Trade-off between effort and quality

- Some approximation algorithms can achieve increasingly smaller approximation ratios $\rho(n) \to 1$ by using more and more computation time.

- We call them approximation schemes

- Required input : ϵ

- Approximation schemes return for any fixed $\epsilon > 0$ a solution with approximation ratio $1 + \epsilon$

Fully polynomial-time approx. scheme

- FPAS, FPTAS
- Returns a solution with approximation ratio $(1+\epsilon)$
- Running time is polynomial in both input size n and $1/\epsilon$
- Fast for small ϵ <u>and</u> large n
- Allows effective problem solving

Polynomial-time approx. scheme

- **PAS, PTAS**

- **Returns a solution with approximation ratio of $(1+\epsilon)$**

- **Running time polynomial in input size n**

- **However, running time can grow exponentially in $1/\epsilon$**

- **Fast for large n but <u>not</u> for small ϵ**

Constant-factor approximations

- APX

- Guarantee a constant-factor approximation ratio

- Approximation ratio is fixed, not a parameter

- Running time is polynomial in problem size n

Approximation and complexity

- **FPTAS are most effective, followed by PTAS and APX**

- **Introduce new complexity classes**

- **Problems in P can be solved in polynomial time**

- **Problems in NP require exponential time**

- $P \subseteq FPTAS \subseteq PTAS \subseteq APX \subseteq NP$

- **Problem is PTAS-hard if no FPTAS exists**

- **Problem is APX hard if no PTAS exists**

Modern heuristics

- Extended variants of improvement heuristics, „metaheuristics"

- Modern heuristics
  - Can be applied to a wide range of problems
  - Use intensification (exploitation) and diversification (exploration) steps

- Intensification steps shall improve quality

- Diversification explores new areas of search space, also accepting complete or partial solutions that are inferior to current solution

Principles (1)

- Start with one or more random solutions

- In iterative steps modify solution(s) to generate one or more new solution(s)

- New solutions are created by search operators (variation operators)

- Regularly perform intensification and exploration phases

  - During intensification, it uses objective function values and focuses variation on high-quality solutions

  - During diversification, usually objective function values are not considered. Modify solutions such that new areas of search space are explored.

Principles (2)

- **Modern heuristics perform a limited number of search steps**

- **To be applicable, two requirements must be met**

  – Representation: We must be able to represent complete solutions so that variation operators can be used.

  – Pair wise fitness comparisons must be possible, indicating which of two solutions is better.

Design elements of Modern Heuristics

1. Representation
2. Variation operators
3. Fitness function
4. Initial solution(s)
5. Search strategy

- Can be addressed to build a new heuristic and to categorize existing ones
- Central to this course

Example: Simulated Annealing (SA)

- Local search that accepts inferior solutions to escape from local optima

- Probability of accepting inferior solution depends on solution quality; it decreases during run.

- Analogy from cooling metals or liquids
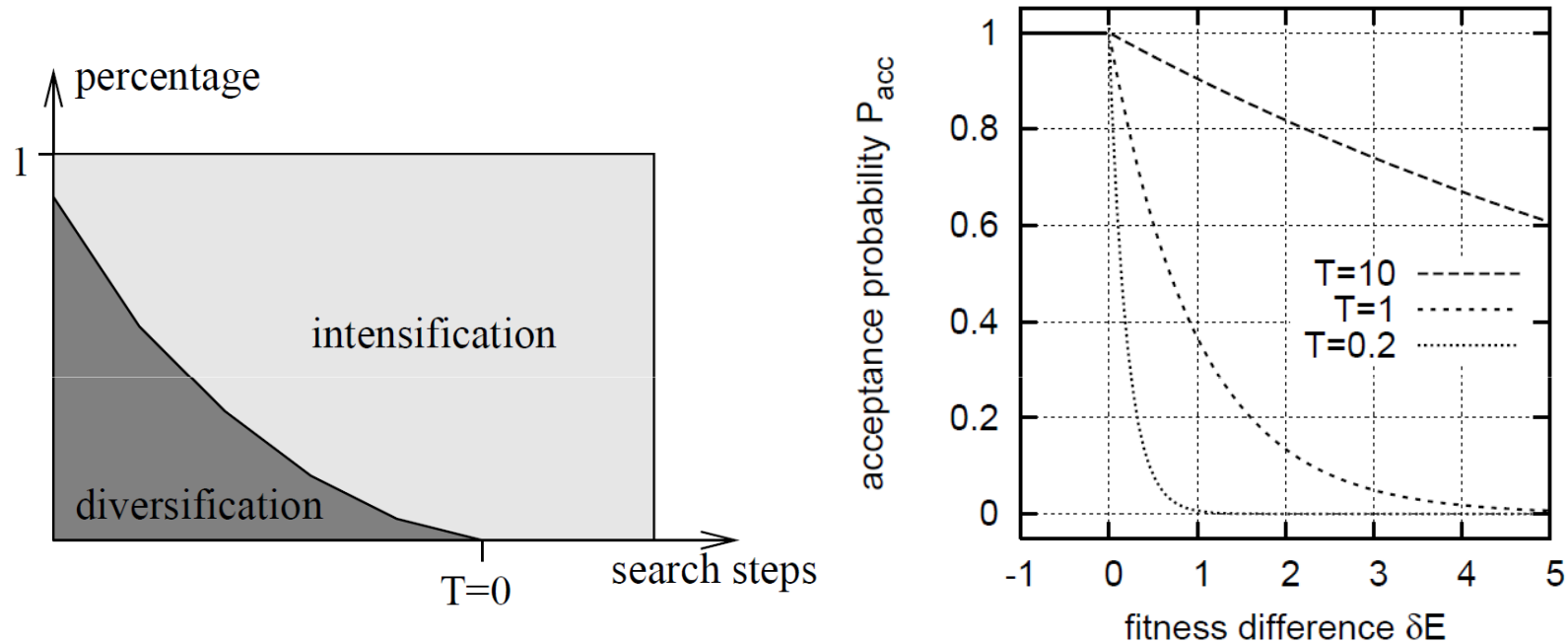
Simulated Annealing

- Uses iterative steps
- In each step: apply variation operator(s) to current solution x^o , obtain new solution x^n
- Accept x^n with probability

$$P_{acc}(T) = \begin{cases} 1 & \text{if } f(x^n) \leq f(x^o) \\ \exp(\frac{-\delta E}{T}) & \text{if } f(x^n) > f(x^o) \end{cases}$$

- Temperature *T* is a strategy parameter
- δE is the fitness difference

Diversification and intensification

- For $T \rightarrow 0$, SA becomes local search
- Probability of accepting inferior solution decreases with fitness difference

Getting the cooling schedule right

- If *T* is reduced very slowly, SA returns optimal solution; however resulting runtime is prohibitive

- If *T* is reduced too fast, SA converges to local optimum.

- Often a fixed schedule is used where $T_{i+1}=cT_i$ (0<c<1) and $c\in[0.9,\ 0.999]$

- $T_0 \approx \sigma$(f(x)) ... 2σ(f(x)), where σ(f(x)) is the standard deviation of objective function values of randomly generated solutions.

Example: SA for the TSP (1)
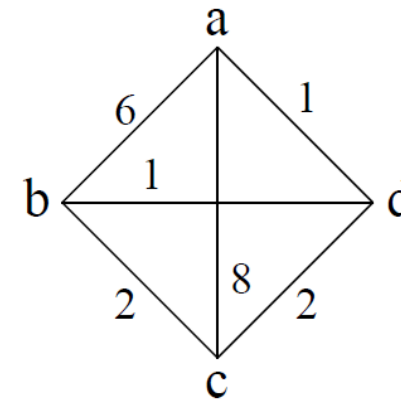
- **Representation: sequence of cities**
- a **is starting city**
- **Two solutions** x,y **are neighbors if Hamming distance** $d(x,y)=2$**.**
- **Three solutions: adbca (12), abdca (17), adcba (11)**

  $$\sigma = \sqrt{62}/3 \approx 2.62 \rightarrow T_0 = 3$$

- **Linear cooling schedule** $T_{i+1} = 0.9T_i$

Modern Heuristics – Modern Heuristics

Example: SA for the TSP (2)

- **Start with initial solution x^o=abcda, $\mathrm{f(x^0)}{=}11$**
- **Variation operator randomly exchanges the position of two cities in tour**
- **New solution: x^1 = abdca, $\mathrm{f(x^1) = 17}$**
- **Replace x^0 with x^1 with probability $\mathrm{P{=}exp(\text{-}6/3)}{\approx}\,0.14$**
- **Generate uniform random number $\mathrm{rnd}{=}[0,1)$ and if rnd<0.14 replace old solution, otherwise continue with x^0.**
- **Then, reduce the temperature: $\mathrm{T}_1 = 2.7$**
- **Continue until a time limit reached or no improvement for some number of steps.**

36

2. Optimization Problems

1. Prerequisites
   1. Search Spaces
   2. Fitness Landscapes
2. Problem Complexity
3. No-Free Lunch Theorem
4. Locality
   1. Fitness distance correlation
   2. Ruggedness
5. Decomposability

Search spaces

- **For formulating optimization models, we need a set of (feasible) solutions**

- **This set defines a search space X**

- **The search spaces "contains"**

  - possible solutions of a problem and

  - relations between the different solutions

(Topological space)

- **Very general, a search space can be defined as a topological space**
- **A topological space is defined as a set X of decision alternatives together with a collection of subsets of X called open sets such that**
  - the empty set $\emptyset$ and whole space X are open sets,
  - the intersection of a finite number of open sets is also an open set, and
  - the union of an arbitrary number of open sets is an open set.
  - A set Y is a subset of a set X (denoted as $Y \subset X$) if every element $\in Y$ is also in X ($x \in Y \rightarrow x \in X$)
- **To define a topological space, we need no definition of similarity between elements in a search space.**

Metric search spaces

- Common topological space where similarity between elements can be measured using some kind of metric
- We have a set X of solutions and a real-valued distance function (also called a metric)

$$d : X \times X \to R$$

assigning a real-valued distance to any pair $x, y \in X$. It is required, that for any $x, y, z \in X$:

$$d(x, y) \geq 0,$$
$$d(x, x) = 0,$$
$$d(x, y) = d(y, x),$$
$$d(x, z) \leq d(x, y) + d(y, z)$$

City-block metric (Manhattan metric)

- For $x, y \in R$, define distance $d(x,y) = |x\text{-}y|$

- Extending to two dimensions: city-block metric (also known as Manhattan distance)

$$d(x,y) = |x_1 - y_1| + |x_2 - y_2|$$

where $x = (x_1, x_2)$, $y = (y_1, y_2)$.

- In n dimensions, the metric becomes

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

41

Euclidean metric

- Solutions are vectors of continuous variables

  $x = (x_1, x_2, ..., x_n), x_i \in R$

- Euclidean distance between x and y is

$$d(x, y) := \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

- For n=1: city-block metric, n=2: standard straight line distance between two points on a 2d-plane

Hamming metric

- Often used for binary search spaces; counts number of items that are not identical

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|,$$

where $d(x,y) \in \{0,...,n\}$.

- Binary Hamming metric can be extended to continuous and discrete decision variables:

$$d(x,y) = \sum_{i=1}^{n} z_i,$$

where

$$z_i = \begin{cases} 1, & \text{for } x_i = y_i, \\ 0, & \text{for } x_i \neq y_i. \end{cases}$$
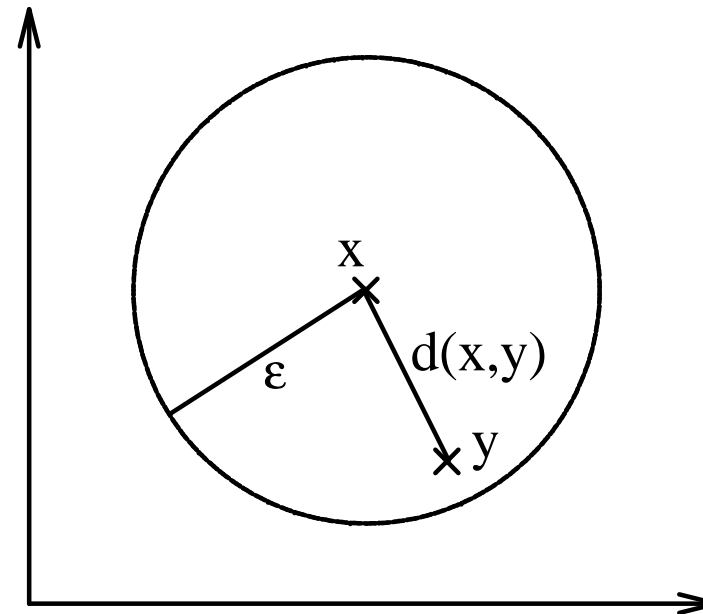
Neighborhoods

- On metric search spaces, we can define similarities between solutions based on the used distance d.

- Neighborhoods determine which solutions are similar to each other with respect to some metric.

- A neighborhood is a mapping $N(x)$: $X \rightarrow 2^x$

  where X is the search space, 2^x is the set of all possible subsets of X and N is a mapping that assigns to each element $x \in X$ a set of elements $y \in X$.

- Usually a neighborhood definition assigns to each solution $x \in X$ a set of solutions y that are similar to x in some sense.

Euclidean Neighborhoods

- We define a neighborhood for a 2-dimensional continuous search space and Euclidean distances.

$$d(x,y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

- All solutions y, where $d(x,y)$ <ϵ are neighboring solutions to x

- All neighboring solutions can be found inside a circle around x with radius ϵ.

City-Block Neighborhoods

- We define a neighborhood for a 2-dimensional continuous search space and city-block distances.

$$d(x, y) := |x_1 - y_1| + |x_2 - y_2|$$

- All solutions inside a rhombus with the vertices $(x_1\text{-}\epsilon, y_1)$, $(x_1, y_1\text{+}\epsilon)$, $(x_1\text{+}\epsilon, y_1)$, $(x_1, y_1\text{-}\epsilon)$ are neighboring solutions.

$(x_1, y_1 + \varepsilon)$

$(x_1 + \varepsilon, y_1)$

ε x

$d(x,y)$ y

$(x_1 - \varepsilon, y_1)$

$(x_1, y_1 -)\varepsilon$
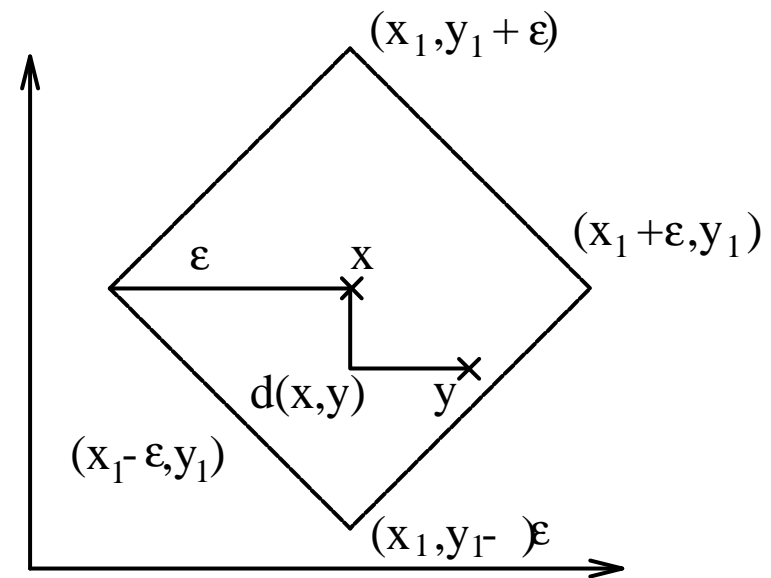
Neighborhoods

- Defining a proper neighborhoods is difficult
- Example:
    - A user can choose from four fruits. These four decision alternatives can be modeled using a metric search space X={0,1}^{2}. Each solution ((0,0), (0,1),(1,0), (1,1)) represents one type of fruit.
    - Although no similarities are defined for the different fruits, the use of a binary search space induces that the solution (0,0) is more similar to (0,1) than to (1,1) (using Hamming distance).
    - Therefore, this problem space is inappropriate for the problem definition as it defines similarities where no similarities exist.
    - A more appropriate model would be $x \in$ {0,...3} and using Hamming distance. Then, all distances between the different solutions are equal and all solutions are neighboring solutions (for ϵ=1).

Neighborhoods

- Definition and use of decision variables naturally leads to a metric.

- If metric induced by decision variables does not fit to the metric of the problem description, the model is inappropriate.

- If a metric used in problem definition does not fit to the metric used in the model, similarities between different decision alternatives do not match the similarities between different solutions described by the model.

Neighborhoods: Example

- Nine different decision alternatives {a, b, c, d, e, f, g, h, i}. An objective value is assigned to each decision alternative. We assume that the decision alternatives form a metric space using the city-block metric, where the distances between all elements are equal. Therefore, all decision alternatives are neighbors (for ϵ =1).

- **Model 1**: We use a metric space X={0,1,2}^{2} and the city-block metric. Therefore, each decision alternative is represented by x=(x_1,x_2), where $x_i \in${0,1,2}. Two solutions are similar to each other if the decision variables have the same values (e.g. solution (1,1) is more similar to (1,2) than to (2,2)). For ϵ =1, each solution has either three or four neighbors. Consequently, our neighborhood differs from the original problem.

- **Model 2**: We use binary variables x_{ij} and the search space is defined as X=x_{ij}, where $x_{ij} \in${0,1}. We have an additional restriction, $\sum_j x_{ij}$=1, where $i \in${1,2} and $j \in${1,2,3}. Again, Hamming distance can be used. For ϵ=1, no neighboring solutions exist. For ϵ=2, each solution has only two neighbors.

- We see that different models for the same problem result in different neighborhoods which do not coincide with the neighborhoods of the original problem.

Neighborhoods: Example

Different search spaces result into different problems

decision alternatives	model 1 (x_1, x_2)	model 2 $\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$
$\{a, b, c, d, e, f, g, h, i\}$	$\{(0,0), (0,1), (0,2),$ $(1.0), (1,1), (1,2),$ $(2,1), (2,2), (2,3)\}$	$\{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$ $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$ $\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}\}$

Fitness Landscapes (1)

- For combinatorial search spaces where a metric is defined, we can introduce the concept of fitness landscape.

- A fitness landscape (X,f,d) of a problem instance consists of

    - a set of solutions $x \in X$,

    - a distance measure d,

   - and an objective function f that measures the quality of each solution.
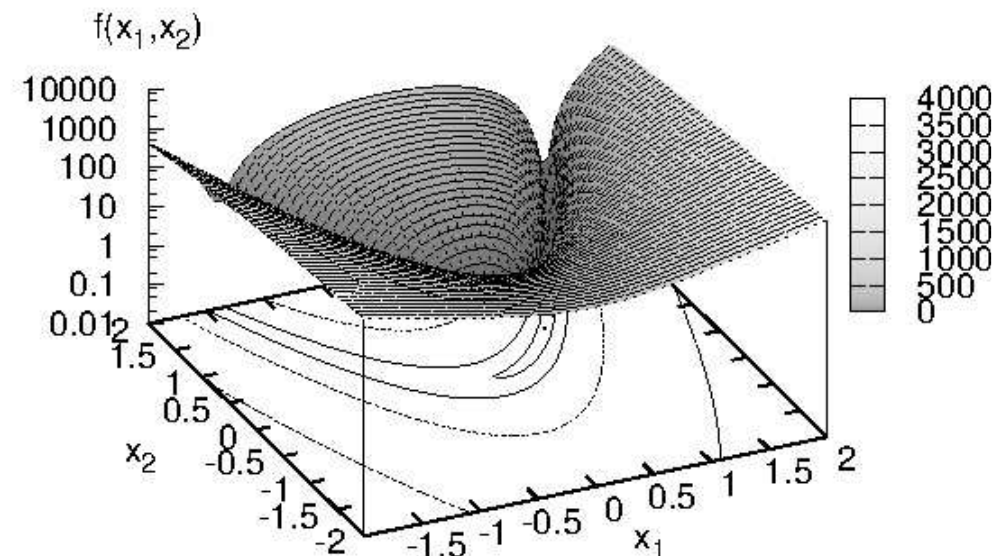
Fitness Landscapes (2)

- d_{min}=min$_{x,y\in X}$(d(x,y)) is the minimum distance between two elements x and y of a search space.

- Two solutions x and y are denoted as neighbors if $d(x,y)$=d_{min}.

- Often, d_{min}=1.

- The fitness landscape can be described as a graph G_L with a vertex set V=X and an edge set E={$(x,y)\in S\times S$ | $d(x,y)$=d_{min}}.

- The distance between two solutions $x,y\in X$ is proportional to the number of nodes that are on the path of minimal length between x and y in the graph G_L.

Fitness Landscapes and Optimal Solutions

- We have a one-dimensional minimization problem. Independently of the used neighborhood, u is the global optimum.

- If we use the 1-dimensional Euclidean distance as metric, we can define a neighborhood around x as $N(x) = \{y \mid y \in X, d(x,y) \leq \epsilon\}$. The solution v is a local optimum if $\epsilon < d_1$.

- Analogously, w is a local optimum for all neighborhoods with $\epsilon < d_2$.

- For $\epsilon \geq d_2$, the only locally optimal solution is the global optimal solution u.

$f(x)$

d_2

w

d_1

v

u

x

0 1
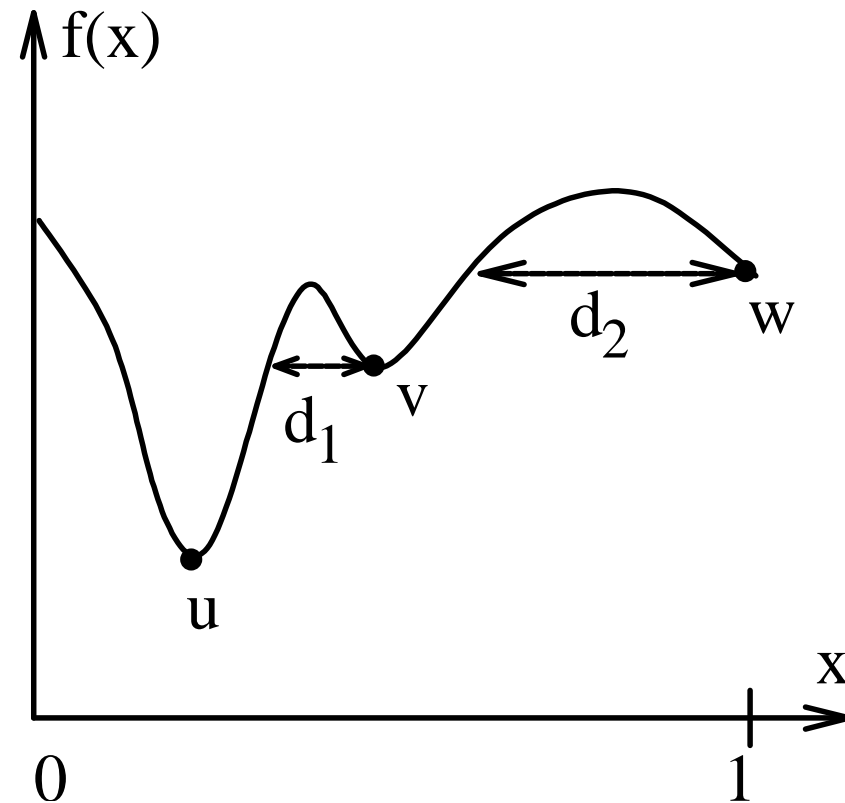
Intro: What are difficult problems?

- **Two different questions:**
  - **How difficult is a problem?**
    - Equivalent to "What is the complexity of the best-performing algorithm that can solve this problem?
    - Complexity classes
  - **How well can a problem be solved using optimization method xyz?**
    - Is optimization methods xyz the right one?
    - No-free lunch theorem

Example: Random Search

- **Functionality**
  - New solutions are chosen randomly and no prior information about the structure of the problem or previous search steps is used.
  - All possible optimization problems have the same difficulty
- **There are no easy or difficult problems for random search.**
- **Number of fitness evaluations for finding the optimum is independent from optimization problem (if optimum is unique).**

Difficult Problems (Problems closed under permutation)

- **Problems, where no meaningful metric can be defined/exists**
  - Examples:
    - Finding largest value in unordered sequence
    - Finding largest value in white noise
  - We have a set of solutions $x \in X$ with objective values $f(x)$.
  - No metric is defined: search algorithms do not "know" how to guide the search through the search space
  - Can only be solved by to iteratively examining all elements of search space, returning the best found solution
  - All optimization methods that can be applied to such problems behave like random search
  - The difficulty of such problems (O(|X|)) is independent of used optimization algorithm.

Complexity of Problems and Algorithms

- The complexity of a problem is the effort that is necessary to solve the problem.

- It is possible to define upper and lower bounds on problem difficulty.

- Lower bounds tell us that a problem has at least this problem difficulty whereas upper bound limit problem difficulty from above.

- Complexity of problems is closely related to the complexity of algorithms.

Complexity of Problems and Algorithms

Upper bounds on problem difficulty:

- We can find upper bounds on problem difficulty

- Based on the complexity of algorithms

- If an algorithm can solve a problem, an upper bound on the difficulty of the problem is the complexity of the algorithm.

Example:

- We study the problem of finding a friend's telephone number in the telephone book. The most straightforward approach is to search through the whole book starting from 'A'. Effort $O(n)$. Therefore, we have an upper bound on problem complexity (linear) as we know a linear algorithm that can solve the problem. A more effective way to solve this problem is bisection which iteratively splits the entries of the book in halves. With n entries, we only need $\log(n)$ search steps to find an address. So, we have a new, improved, upper bound on problem difficulty.

Complexity of Problems and Algorithms

Lower bounds on problem complexity

- Finding lower bounds on problem difficulty is more difficult

- We have to show that no algorithm exists that needs less effort to solve the problem.

- Optimization problems where no metric is defined can only be solved when examining all available solutions.

- Therefore, we have a lower bound on problem difficulty as the effort of algorithms to solve such problems increases at least linearly with the size of the search space.

- The lower bound must hold for all possible algorithms that can be used to solve the problem.

- A problem is denoted to be closed if the upper and lower bound on its problem difficulty are identical.

Formulating Complexity: Landau notation

- Landau notation can be used to compare asymptotic growth of functions

- Helpful when measuring the complexity of problems or algorithms.

- Allows to formulate asymptotic upper and lower bounds on function values.

Landau notation

With $f, g : \mathbb{N} \to \mathbb{R}$, we can define:

- **asymptotic upper bound** ("big O notation"):
  $f \in O(g) \Leftrightarrow \exists\, c > 0, \exists\, n_0 > 0 \;\forall\, n \geq n_0 :$
  $|f(n)| \leq c|g(n)|$: f is dominated by g.

- **asymptotically negligible** ("little o notation"):
  $f \in o(g) \Leftrightarrow \forall\, c > 0, \exists\, n_0 > 0 \;\forall\, n \geq n_0 :$
  $|f(n)| < c|g(n)|$: f is growing slower than g.

Landau notation

- **asymptotic lower bound**:
  $f \in \Omega(g) \Leftrightarrow g \in O(f)$: f grows at least as fast as g.

- **asymptotically dominant**:
  $f \in \omega(g) \Leftrightarrow g \in o(f)$: f grows faster than g.

- **asymptotically tight bound**:
  $f \in \Theta(g) \Leftrightarrow g \in O(f) \wedge f \in O(g)$: g and f grow the same.

Examples

1. We want to find the smallest number in an unordered list of n numbers. The complexity of this problem is $\Theta(n)$ when using linear search and examining all possible elements in the list. As it is not possible to solve this problem faster than linear, there is no gap between the lower bound $\Omega(n)$ and upper bound $O(n)$.

2. We want to find an element in an ordered list with $O(n)$ items (for example finding a telephone number in the telephone book). Binary search iteratively splits the list in two halves and can find any item in $\log(n)$ search steps. Therefore, the upper bound on the complexity of this problem is $O(\log(n))$. As the lower bound is equal to the upper bound (see literature), the complexity of the problem is $\Theta(\log(n))$.

3. We want to sort an array of *n* arbitrary elements. By using standard sorting algorithms like merge sort it can be solved in $O(n \log(n))$. As the lower bound is $\Omega(n \log(n))$, the difficulty of this problem is $\Theta(n \log(n))$.

Complexity Classes

- Computational complexity theory categorizes decision problems in different groups based on their difficulty.

- Difficulty is defined with respect to the amount of computational resources that are at least necessary.

- Effort (amount of computational resources) necessary to solve an optimization problem depends on time and space complexity.

  – Time complexity: how many iterations/number of search steps are necessary to solve a problem.

  – Space complexity: amount of space (memory) necessary to solve a problem.

- Both depend on the size n of the problem.

Complexity Classes

- A set of problems where the amount of computational resources necessary to solve the problem have the same asymptotic behavior.

- For all problems in one complexity class, we can give bounds on the computational complexity (in general, time and space complexity).

- Usually, bounds depend on the size n of the problem.

Complexity Class P

- The complexity class P (polynomial) is the set of decision problems that can be solved by an algorithm with worst-case polynomial time complexity.

- Time necessary to solve a problem in P is asymptotically bounded (for $n > n_o$) by a polynomial function $O(n^k)$.

- For all problems in P, an algorithm exists that can solve any instance of the problem in $O(n^k)$.

- All problems in P can be solved effectively.

Complexity Class NP

1. Set of decision problems where a "yes" solution of a problem can be verified in polynomial time.

   – Formal representation of x and time to check its validity are polynomial or polynomially-bounded.

2. Set of all decision problems that can be solved by a non-deterministic algorithm in worst-case polynomial time

   – A non-deterministic algorithm always selects the value (possibility) that leads to a "yes" answer, if a "yes" answer exists.

- Both definitions of NP are equivalent to each other (Consider that non-deterministic algorithms can not be carried out by conventional computers and there is no idea how to construct a non-deterministic algorithm).

Complexity Class NP: Informally

- The class NP consists of all "reasonable" problems of practical importance where a "yes" solution can be verified in polynomial time

- This means the objective value of the optimal solution can be calculated fast.

- For problems not in NP, even verifying that a solution is valid (is a "yes" answer) can be extremely difficult (needs exponential time).

Tractable and Intractable Problems

- Problems that can be solved using a polynomial-time algorithm (upper bound $O(n^k)$ on the running time of the algorithm, k constant) are tractable.

- Tractable problems are easy to solve

- Running time increases relatively slowly with larger problem size n.

- Example: Finding the lowest element in an unordered list of size n is tractable. There are algorithms with $O(n)$ time complexity. Spending twice as much effort allows us to solve problems twice as large.

Tractable and Intractable Problems

- Problems are intractable if they cannot be solved by a polynomial-time algorithm and there is a lower bound on the running time which is $\Omega\left(k^n\right)$.

- Example:
  - Finding the correct number for a decimal door lock with n digits is intractable. The time necessary for finding the correct key is $\Omega(10^n)$. Using a lock with one more digit increases number of search steps by a factor of 10.
  - We have n binary decision variables and assign a random variable to each solution. Resulting problem is closed under permutation. Finding optimal solution is $\Theta(\log(n))$.

Polynomial and Exponential functions

constant	$O(1)$
logarithmic	$O(\log n)$
linear	$O(n)$
quasilinear	$O(n \log n)$
quadratic	$O(n^2)$
polynomial (of order c)	$O(n^c)$, $c > 1$
exponential	$O(k^n)$
factorial	$O(n!)$
super-exponential	$O(n^n)$

NP-hard

- All decision problems in P are tractable

- If we assume that P$\neq$NP, then some problems are in NP but not in P.

- They are difficult: no polynomial-time algorithms exist.

- Among decision problems in NP, there are problems where no polynomial algorithm is available and which can be transformed to each other in polynomial time.

- Consequently, a problem is denoted to be <u>NP-hard</u> if an algorithm for solving this problem is polynomial-time reducible to an algorithm that is able to solve any problem in NP.

NP-hard

- A problem A is polynomial-time reducible to a different problem B if and only if there is a transformation that transforms any solution of A into a solution of B in polynomial time such that if and only if a solution is a "yes" instance for A it is also a "yes" instance for B.

- Informally, a problem A is reducible to some other problem B if problem B has same difficulty or is easier than problem A.

- Therefore, NP-hard problems are at least as hard as any other problem in NP, although they might be harder. Therefore, NP-hard problems are not necessarily in NP.
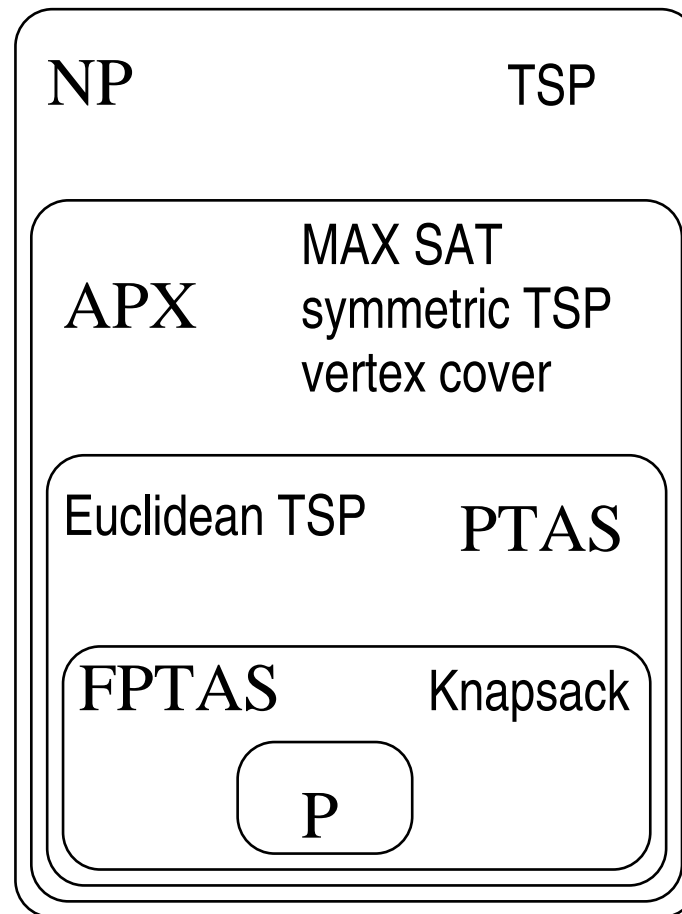
NP-complete

- Cook introduced the set of NP-complete problems as a subset of NP.
- A decision problem A is denoted to be NP-complete if
  - A is in NP and
  - A is NP-hard.
- No other problem in NP is more than a polynomial factor harder than any NP-complete problem.
- NP-complete problems are the most difficult problems in NP.

NP-complete

- All NP-complete problems form one set: NP-complete problems have the same complexity.

- However, it is unclear if NP-complete problems are tractable, or not.

- If we are able to find a polynomial-time algorithm for any one of the NP-complete problems, then every NP-complete problem can be solved in polynomial time.

- Then, also all other problems in NP can be solved in polynomial time (are tractable) and thus P=NP.

- On the other hand, if it can be shown that one NP-complete problem is intractable, then all NP-complete problems are intractable and P$\neq$ NP (one million Dollar question!).

Different classes of NP optimization problems

NP   TSP

APX   MAX SAT
           symmetric TSP
           vertex cover

Euclidean TSP   PTAS

FPTAS   Knapsack

P

No-Free-Lunch Theorem and Black-box optimization

- There is a trade-off between effectiveness and application range of optimization methods

- Black-box optimization methods are algorithms that need no additional information about the structure of a problem but are able to reliably and efficiently return high-quality solutions for a large variety of different optimization problems

- NFL theorem says that Black-box optimization is not possible

- An algorithm's performance can only be high if (correct) problem-specific assumptions are made about the structure of the optimization problem and the algorithm is able to exploit these problem-specific properties.

No-Free-Lunch Theorem

… for both static and time dependent optimization problems, the average performance of any pair of algorithms across all possible problems is exactly identical. This means in particular that if some algorithm A_1's performance is superior to that of another algorithm A_2 over some set of optimization problems, then the reverse must be true over the set of all other optimization problems (Wolpert and Macready, 1997)
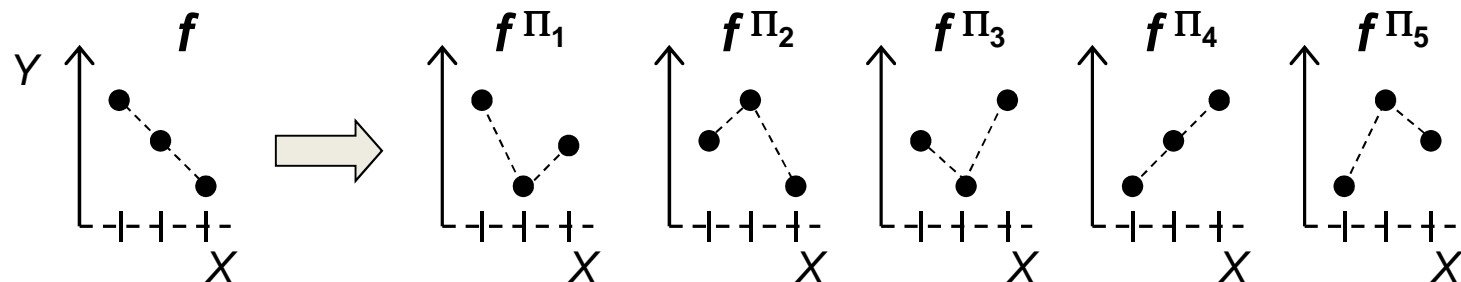
No-Free-Lunch Theorem

- **Optimization Problem** $f; X \rightarrow Y$
  - X is finite
  - Y is finite and ordered
- **Heuristics search method** H
- **Sequence of solutions generated by H:**
  $$H(f, m) = ((x_1, f(x_1)), (x_2, f(x_2)), ..., (x_m, f(x_m)))$$
- H **generates** x_{m+1} **dependent on** $H(f, m)$
  - „Black-Box Algorithm", solutions are only sampled once
- **We observe a sequence of fitness values** $H^y(f, m) = (f(x_1), f(x_2), ..., f(x_m))$
- **We measure performance of search** g:
  - $g(H^y(f, m))$
  - Example: $g(f(x_1), f(x_2), ..., f(x_m)) = \min_{i=1,...,m} \{f(x_i)\}$
- **Average performance (over all possible problems f)**
  - $\sum_{f \in F} g(H^y(f, m)) / |F|$

No-Free-Lunch Theorem

- **Assumption: F is closed under permutation**
  - All possible permutations Π of the search space X
    $$\Pi : X \rightarrow X,$$
    where $f^{\Pi}(x) = f(\Pi^{-1}(x))$
  - $f \in F \Rightarrow f^{\Pi} \in F$

  Example:

No-Free-Lunch Theorem

The average performance of any pair of algorithms across all possible problems is exactly identical
(independent on how we measure the performance of the algorithms)

Comments on NFL (1)

- Wolpert/Macready: *„We cannot emphasize enough that no claims whatever are being made in this paper concerning how well various search algorithms work in practice."*

- However, statements like „In general, metaheuristic H_1 is better than H_2 " make no sense.

- An algorithm's performance can be increased if (correct) problem-specific assumptions are made about the structure of the optimization problem and the algorithm is able to exploit these problem-specific properties.

Comments on NFL (2)

NFL holds for

- Needle-in-a-haystack problems
- Random problems with trivial topology, …

The NFL-Theorem does not hold for problems that are not closed under permutation

- Decomposable problems
- Problems with high locality
- Problems, where neighboring solutions have similar fitness (Christensen and Oppacher, 2001)

Difficult of Problems for Heuristics

- Exact optimization methods (like Branch&Bound, cutting plane, and others) have exponential effort for NP-complete problems.

- Heuristics are not optimal (no guarantee that optimal solution is found) but their effort can be adjusted by user. Solution quality is often good.

- Question:
  - What makes problems difficult (easy) for heuristics?
  - For which problems do metaheuristics perform better than random search?

Number of local optima

- One optimum: unimodal, can often be solved well by hillclimber

- Even functions with low number of local optima can be arbitrarily hard. Compare needle in haystack with sphere, or deceptive traps

- Number of optima no sufficient indicator for problem difficulty

Basins of attraction

- For several optima: basin of attraction of local optimum is the set of solutions from which the local optimum is reached by a hillclimber („part of the same peak")

- Performance of local search correlates inversely with size of basin of attraction

- Sphere function: one basin of attraction, usually simple to solve.

Measuring the locality of problems

- "Do similar solutions have similar fitness?"

- Locality describes how well the distances $d(x,y)$ between $x,y \in X$ correspond to the differences of the objective values $|f(x)\text{-}f(y)|$.

- The locality of a problem is high
  - if neighboring solutions have similar objective values
  - and difference of the objective values increases with larger distance

- The locality of a problem is low
  - if small distances do not correspond to small differences of the objective values.

Measuring the locality of problems

- The important determinants for the locality of a problem are
  - the metrics defined on the search space
  - and the objective function f

- For continuous decision variables, locality is known as causality. High and low locality correspond to strong and weak causality, respectively.

Locality and guided search

- Guided search methods: iteratively sample solutions and use the objective values of previously sampled solutions to guide the future search process

- In contrast to random search: distinguish between promising and non-promising areas in the fitness landscape

- New solutions are usually generated in the neighborhood of promising solutions with high-objective values.

- Most local search algorithms fall in this category

Locality and guided search

- The locality of optimization problems has a strong impact on their difficulty for guided search methods.

- High locality allows guided search to find high-quality solutions in the neighborhood of already found good solutions.

- Moving from low-quality solutions to high-quality solutions works well if the problem has high locality.

- If a problem has low locality, guided search can not make use of previous search steps

- Can not extract information that can be used for guiding the search

- For problems with low locality, guided search methods behave like random search.

Measures of Locality of Search Spaces

- Fitness-Distance Correlation
- Ruggedness

Fitness-Distance Correlation (FDC)

- FDC measures the difficulty of problems for guided search methods

- The difficulty of an optimization problem is determined by
  - how the objective values are assigned to the solutions $x \in X$ and
  - what metric is defined on X.

Fitness-Distance Correlation

The *fitness-distance correlation coefficient* is defined as

$$\rho_{FDC} = \frac{c_{fd}}{\sigma(f)\sigma(d_{opt})},$$

where

$$c_{fd} = \frac{1}{m}\sum_{i=1}^{m}(f_i - \langle f \rangle)(d_{i,opt} - \langle d_{opt} \rangle)$$

is the covariance of f and d_{opt}. $\langle f \rangle$, $\langle d_{opt} \rangle$, $\sigma(f)$, and $\sigma(d_{opt})$ are the means and standard deviations of f and d_{opt}, respectively. $d_{i,opt}$ is the distance of solution i towards the optimal solution x^*.

Fitness-Distance Correlation

- The fitness-distance correlation coefficient $\rho_{\text{FDC}} \in [\text{-}1,1]$ measures the linear correlation between the fitnesses of search points and their distances to the global optimum $x*$

- As ρ_{FDC} represents a summary statistic of f and d_{opt}, it works well if f and d_{opt} follow a bivariate normal distribution.

- For problems, where f and d_{opt} do not follow a normal distribution, using the correlation as a measurement of problem difficulty for guided search methods will not yield meaningful results.

Fitness-Distance Correlation

- Based on the FDC coefficient, we can classify fitness landscapes into three classes,
  - straightforward $(\rho_{FDC} \leq$ -0.15)
  - difficult   (-0.15$\leq \rho_{FDC} \leq$ 0.15)
  - misleading $(\rho_{FDC} \geq$ 0.15)
- Straightforward:
  - Fitness of a solution is correlated with the distance to the optimal solution.
  - With lower distance, the fitness difference to the optimal solution decreases.
  - The structure of the search space guides search methods towards the optimal solution
 - Such problems are usually easy for guided search method.

Fitness-Distance Correlation

- **Difficult:**
  - No correlation between the fitness difference and the distance to the optimal solution.
  - Fitness values of neighboring solutions are uncorrelated
  - The structure of the search space provides no information about which solutions should be sampled next by the search method.
- **Misleading:**
  - Fitness difference is negatively correlated with distance to optimal solution
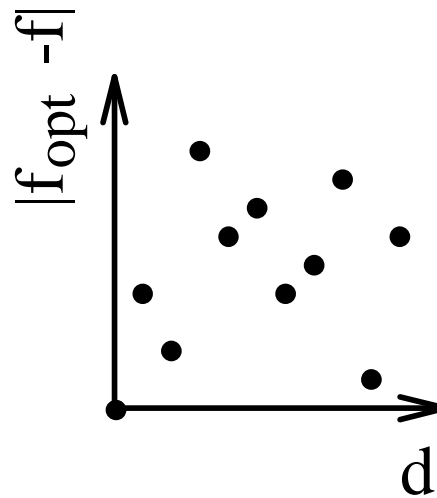 - Structure of the search space misleads a local search method to sub-optimal solutions.
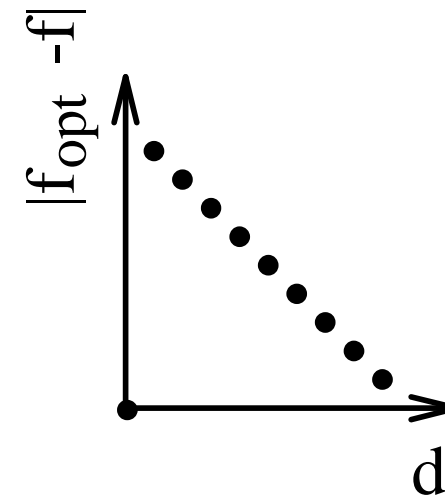
96

Fitness-Distance Correlation

- **Different classes of problem difficulty**
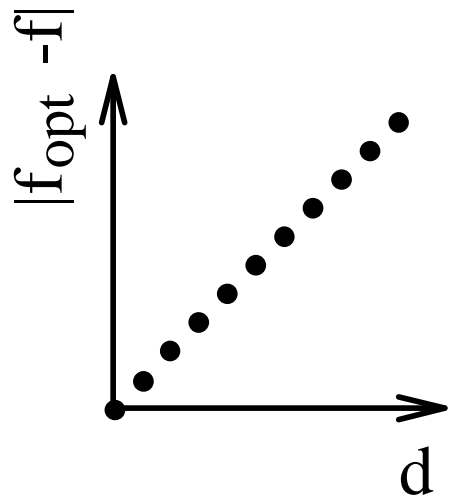
pos. correlation   uncorrelated   neg. correlation

97

Ruggedness

- For studying the FDC of problems, it is necessary to know the optimal solution.

- However, in general the optimal solution is not known.

- The difficulty of problems for guided search methods is influenced by properties of the fitness landscape like

  - the number of local optima or peaks in the landscape,

  - the distribution of the peaks in the search space, and

  - the height of the different peaks.

Random walks and ruggedness

- Correlation functions have been proposed to measure the ruggedness of a fitness landscape.

- Like in fitness-distance correlation, the idea is to consider the objective values as random variables and to obtain statistical properties on how the distribution of the objective values depends on the distances between solutions.

Ruggedness

- The autocorrelation function of a fitness landscape is defined as

$$\rho(d) = \frac{\langle f(x)f(y)\rangle_{d(x,y)=d} - \langle f\rangle^2}{\langle f^2\rangle - \langle f\rangle^2},$$

  where $\langle f\rangle$ denotes the average value of f over all $x \in X$ and $\langle f(x)f(y)\rangle_{d(x,y)=d}$ is the average value of $f(x)f(y)$ for all pairs $(x,y) \in S \times S$, where $d(x,y) = d$.

Ruggedness

- The autocorrelation function has the attractive property of being in the range [-1,1]. An autocorrelation value of 1 indicates perfect correlation (positive correlation) and -1 indicates prefect anti-correlation (negative correlation).

- For a fixed distance d, ρ is the correlation between the objective values of all solutions that have a distance of d.

Decomposability

- The decomposability of a problem describes how well a problem can be decomposed into several, smaller sub-problems that are independently of each other.

    - The decomposability of a problem is high if the structure of the objective function is such that not all decision variable must be considered simultaneously to calculate the objective function but there are groups of decision variables that can be set independently of each other.

    - It is low if it is not possible to decompose a problem into sub-problems that have little interdependencies between.

Decomposability

- If a problem can only be solved by considering all n variables at the same time, it is not separable

- If decisions about the next solution to visit can reliably be made with just considering $k<<n$ decision variables, the problem is separable

- Separable functions are often easier because less dimensions/variables depend on each other

- Reduces number of solutions, e.g. $2^k << 2^n$

Decomposability

- Decomposability is relevant for recombination-based search methods

- Recombination-based methods try different decompositions of the problem, solve the sub-problems, put together the solutions for these sub-problems.

- For such types of optimization methods, decomposability is meaningful as high decomposability results in low problem difficulty

- Solving smaller sub-problems is usually easier than solving the larger, original problem.

- Different approaches for measuring decomposability of problems:
  - Polynomial decomposition
  - Walsh Analysis
  - Schemata Analysis and Building Blocks

104

Polynomial Decomposition

- The linearity of an optimization problem can be measured by the polynomial decomposition of the problem.

- It measures how well a problem can be decomposed into smaller sub-problem

- For binary decision variables, each objective function f defined on l decision variables $x_i \in \{0,1\}$ can be decomposed in the form

$$f(x) = \sum_{N \subset \{1,\dots,l\}} \alpha_N \prod_{n \in N} e_n^T x,$$

where e_n contains 1 in n-th column and 0 elsewhere, T denotes transpose, the α_N are coefficients

Polynomial Decomposition

- The coefficients α_i describe the non-linearity of the problem

- If there are high order coefficients in the decomposition of the problem, the function is (highly) nonlinear.

- If the decomposition of a problem only has order 1 coefficients, then the problem is linear decomposable.

Polynomial Decomposition

- It is possible to determine the maximum non-linearity of $f(x)$ by its highest polynomial coefficients.

- The higher the order of the α_i, the more non-linear the problem is.

- There is some correlation between the non-linearity of a problem and the difficulty of a problem for recombination-based search methods

- The order of non-linearity can only give an upper limit on the problem difficulty.

- There could be high order α_i although the problem can still easily be solved by recombining search methods.

Polynomial Decomposition: Example

$$f(x) = \begin{cases} 1 & \text{if } x = 00, \\ 2 & \text{if } x = 01, \\ 4 & \text{if } x = 10, \\ 10 & \text{if } x = 11, \end{cases}$$

$f(x)=\alpha_0+\alpha_1 x_0+\alpha_2 x_1+\alpha_3 x_0 x_1=1+x_0+3x_1+5x_0 x_1$.

- Easy for recombining search methods: two decision variables can be solved independently of each other.

- The problem is (wrongly) classified as difficult.

- This misclassification of problem difficulty is due to the fact that the polynomial decomposition assumes a linear decomposition and could not appropriately describe non-linear (quadratic) dependencies.

Schemata Analysis and Building Blocks

- Schemata analysis is mainly used in the genetic algorithm domain

- Main search operator of genetic algorithms is recombination

- Schemata are usually defined for binary search spaces and thus schemata analysis is mainly applicable to problems with binary decision variables. However, the ideas of building blocks are also applicable to other search spaces.

- When using l binary decision variables $x_i \in \{0,1\}$, a schema $h=(h_1,h_2,...,h_l)$ is defined as a ternary string of length l

- $h_i \in \{0,1,*\}$. * denotes a "don't care" symbol and tells us that the lth decision variable is not fixed.

Schemata

- A position in a schema is fixed if there is a 0 or a 1
- The size or order $o(h)$ of a schema h is defined as the number of fixed positions (0s or 1s) in the schema string.
- The defining length $\delta(h)$ of a schema h is defined as the distance between (number of bits that are between) the two outermost fixed bits.
- The fitness $f(h)$ of a schema is defined as the average fitness of all instances of this schema and can be calculated as

$$f(h) = \frac{1}{||h||} \sum_{\boldsymbol{x} \in h} f(\boldsymbol{x})$$

- For example, x=01101 and y=01100 are instances of h=0*1**.
- The number of solutions that are an instance of a schema h can be calculated as $2^{l-o(h)}$.

Building Blocks

- "highly fit, short-defining-length schemata".
- A BB can be described as a solution to a subproblem that can be expressed as a schema. A thus-like schema has high fitness and its size is smaller than the length l of the binary solution.
- By combining BBs of lower order, recombining search methods like genetic algorithms can form high-quality over-all solutions.
- We can interpret BBs also from a biological perspective.
  - Using the notion of genes we can interpret BBs as genes. A gene consists of one or more alleles and can be described as a schema with high fitness. The alleles in a chromosome can be separated (decomposed) into genes which do not interact with each other and which determine one specific property of an individual like hair or eye color.
- BBs can be used to describe the difficulty of optimization problems for recombining search algorithms.
- If the sub-solutions to a problem (the BBs) are short (low $\delta(h)$) and of low order (low $o(h)$), then the problem is assumed to be easy.

BB-based Problem Difficulty

- There are three types of problem difficulty:

- Difficulty within a building block (intra-BB difficulty)

- Difficulty between building blocks (inter-BB difficulty)

- Difficulty outside of building blocks (extra-BB difficulty)

Intra-BB difficulty

- If we count the number of schemata of order $o(h)=k$ that have the same fixed positions, there are 2^k different schemata.

- Viewing a BB of size k as a subproblem, there are 2^k different solutions to this subproblem.

- Such subproblems can not be decomposed any more and usually guided or random search methods are applied to find the correct solution BB for the decomposed subproblems.

- Deceptive Problems (see negative fitness distance correlation) are at the core of intra-BB difficulty

- The intra-BB difficulty of a problem can be measured by the maximum length δ(h) and size $k=o(h)$ of the BBs h.

Inter-BB and Extra-BB difficulty

- The contributions of different sub-problems to the objective function can be different. The sub-problems can have a non-uniform contribution to the overall objective value of a solution.

- Important for inter-BB difficulty.

- A problem can often not be decomposed into completely separated and independent sub-problems, but there are still some interdependencies between the different subproblems which are an additional source of inter-BB difficulty.

- Sources of extra-BB difficulty are factors like noise.

- Additional, non-deterministic noise can randomly modify the objective values of solutions and make the problem more difficulty for recombining search methods as no accurate decisions can be made on the optimal solutions for the different sub-problems.

- A similar problem occurs if the evaluation of the solutions is non-stationary. Non-stationary environments results into solutions that have different evaluation values at different moments in time.

Representations

JG|U

1. A Short Introduction to Representations

    1. Defining Representations

    2. Representations, Operators, and Metrics

    3. Direct and Indirect Representations

2. Design Guidelines for Representations

3. Properties of Representations

    1. High-Locality Representations

    2. Redundant Representations and Neutral Networks

Review: Modern heuristics

- **Modern heuristics**
  - Can be applied to a wide range of problems
  - Use intensification (exploitation) and diversification (exploration) steps
- **Intensification steps shall improve quality**
- **Diversification explores new areas of search space, also accepting complete or partial solutions that are inferior to current solution**

Review: Principles of Modern Heuristics

- Start with one or more random solutions
- In iterative steps modify solution(s) to generate one or more new solution(s)
- New solutions are created by search operators (variation operators)
- Regularly perform intensification and exploration phases
  - During intensification, use objective function value and focus variation on high-quality solutions
  - During diversification, usually objective function values are not considered. Modify solutions so that new areas of search space are explored

Genotypes and phenotypes

- Mendel recognized that nature stores information about an individual in pair-wise alleles

- Genetic information determines properties, appearance, shape of an individual

- Distinguish between genetic code and outward appearance

- There is a transformation between the genetic information (genotypes) and the outward appearance (phenotypes)

- Transformation is called a „representation"

- Representations map genotypes on phenotypes

Defining representations (1)

- A representation assigns genotypes to corresponding phenotypes.

- Every search and optimization algorithms needs a representation.

- The representation allows us to represent a solution to a specific problem.

- Different representations can be used for the same problem.

- Performance of search algorithm depends on properties of the used representation and how suitable is the representation in the context of the used genetic operators.

Defining representations (2)

- An optimization problem $f(x)$ can be separated into a genotype-phenotype mapping f_g and a phenotype-fitness mapping f_p

$$f_g(\boldsymbol{x}_g) : \Phi_g \to \Phi_p,$$
$$f_p(\boldsymbol{x}_p) : \Phi_p \to \mathbb{R},$$

  where $f = f_p \circ f_g = f_p(f_g(x_g))$

- A change of f_g also changes the properties of f

- The genetic operators mutation and crossover are applied to x_g whereas the selection process is based on the fitness of x_p

- $f_p(x_p)$ determines the fitness and complexity of the problem

- $f_g(x_g)$ determines the used representations

Standard genotypes: Binary genotypes

- Commonly used in Genetic Algorithms
- Recombination is main operator, mutation is background noise
- Search space is Φ_g = $\{0,1\}^l$ where l is length of a binary vector
  $x^g = (x^g_1, ..., x^g_l)$
- Representation depends on problem to be solved
- Often natural for combinatorial problems
- When using binary representations for integers, decide between unary, Gray, or binary.
- When using binary representations for floats, precision depends on number of bits in genotype.

Standard genotypes: Integer genotypes

- Use χ-ary alphabet instead of binary, where $\{\chi \in \mathbb{N} \,|\, \chi > 2\}$ can also be used in phenotypes

- Instead of coding 2^l solutions, size of search space becomes χ^l

- Recommended when phenotype is integer

Standard Genotypes: Continuous genotypes

- The search space is Φ_g = R^l where I is the size of the real-valued vector

- Often used in evolution strategies, nonlinear numerical optimization, rely on local search

- Can also encode permutations, trees, schedules, or tours.a

Representations make the difference

- Representations change the character and difficulty of optimization problems

- E.g. $f_p = x_p$, $x \in N$

- Different problem depending on the used representation
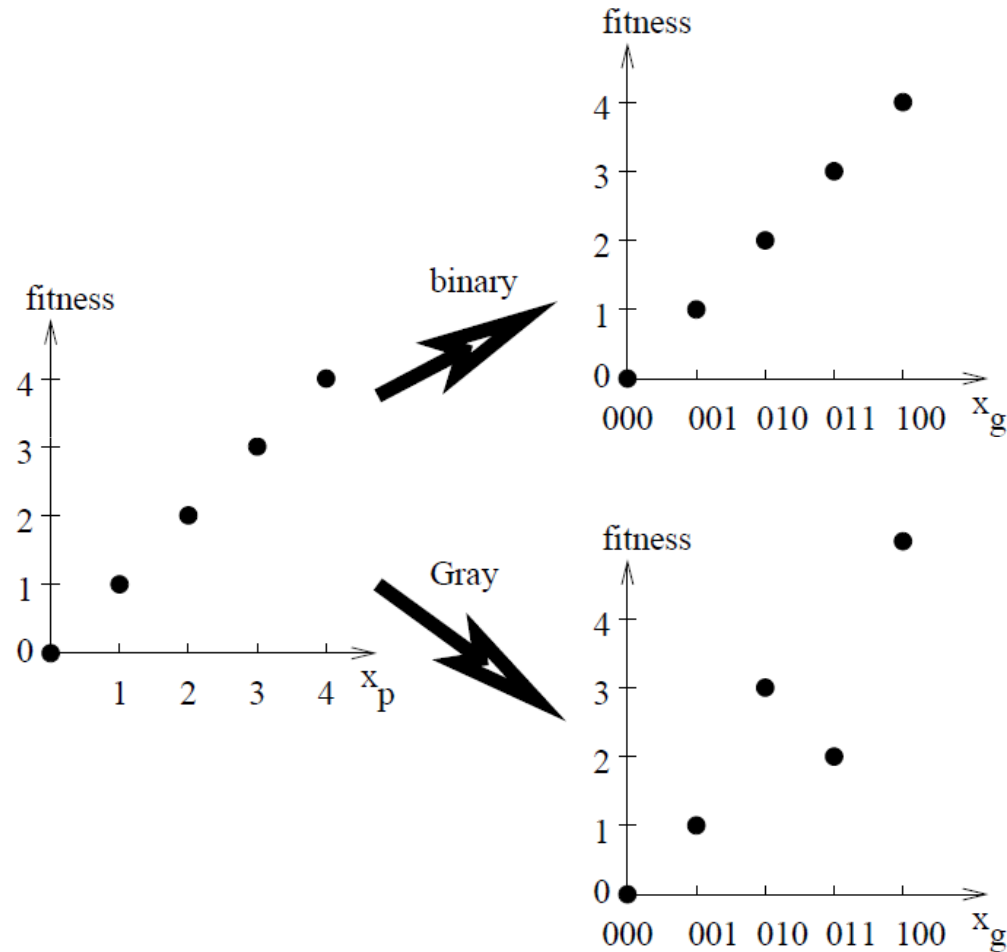
Representations make the difference (2)

- Phenotypic problem easy to solve for hill-climber.

- When using bit-flipping GA the Gray-encoded problem is easier to solve than the binary-encoded problem.

- Gray encoding induces less local optima when used on problems of practical relevance (compare Free Lunch theorem).

- Search performance depends on used search method. If other search methods (e.g. different operators) are used, then search performance is different

Representations, Operators, Metrics

- **Representation, metric defined on Φ_g and Φ_p, and genetic operators depend on each other and are closely related.**
  - A representation is just a mapping from Φ_g to Φ_p. It assigns any possible $x_g \in \Phi_g$ to an $x_p \in \Phi_p$
  - In both search spaces, Φ_g and Φ_p, a metric is or has to be defined. The metric determines the distances between the individuals and is the basis for measuring similarities between individuals. In general, the metric used for Φ_p is defined by the considered problem. The metric used for Φ_g is determined by the used search operators.
  - Genotypic operators like mutation and crossover are defined based on the used metric

Representations, Operators, Metrics (2)

- Mutation:
  - The application of mutation to an individual results in a new individual with similar properties. There is a small distance between offspring and parent.

- Crossover:
  - Crossover combines the properties of two or more parents in an offspring. The distance between offspring and parent should be equal or smaller than the distance between both parents.

Representations, Operators, Metrics (3)

- Results:

  - Metric on Φ_g and used operators depend on each other. The one determines the other.

  - Representations "transform" the metric on Φ_g to the (problem dependent) metric on Φ_p. (Compare locality, causality, and distance distortion)

Direct representations

- **If the genetic operators are applied directly to the phenotypes it is not necessary to specify a representation and the phenotypes are identical with the genotypes:**

$$f_g(\boldsymbol{x}_g) : \Phi_g \to \Phi_g,$$
$$f_p(\boldsymbol{x}_p) : \Phi_g \to \mathbb{R}.$$

This means, f_g is the identity function $f_g(x_g)=x_g$. Using direct representations do not neccessarily make life easier:

- – Design of proper operators is difficult
- – How can we apply specific types or EAs (like EDAs)?
- – Representation issues are not important any more ($\Phi_g = \Phi_p$ and $f_g(x_g)=x_g$).

129

Direct representations – Genetic Programming

- Representation issues are also relevant to Genetic Programming.

- Phenotypes: Programs, logical expressions.
  Genotypes: Parse trees, bitstrings, linear structures, …

- Neglecting proper genotype-phenotype mappings can result in low performance of GP approaches.

- Example: Standard GP (expression tree representation and subtree swapping crossover) cannot solve problems where optimal solutions require very full or very narrow trees. This is due to problems of the representation (interplay between genotypes and used search operators).

Benefits of Indirect representations

- The use of an explicite genotype-phenotype mapping has some benefits:

  – specific constraints can be considered.

  – Standardized genetic operators with known behavior and properties can be used.

  – An indirect representation is necessary if problem-specific operators are either not available or difficult to design.

 – Representation can make problem easier by incorporating problem-specific knowledge.

Specific constraints

- Example: Tree optimization problems
- A tree is a fully connected graph with exactly $n-1$ links (for an n node network). There are no circles in a tree.
- A graph can be represented by its characteristic vector.

0	1	0	0	0	1	0	1	0	1
A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

132
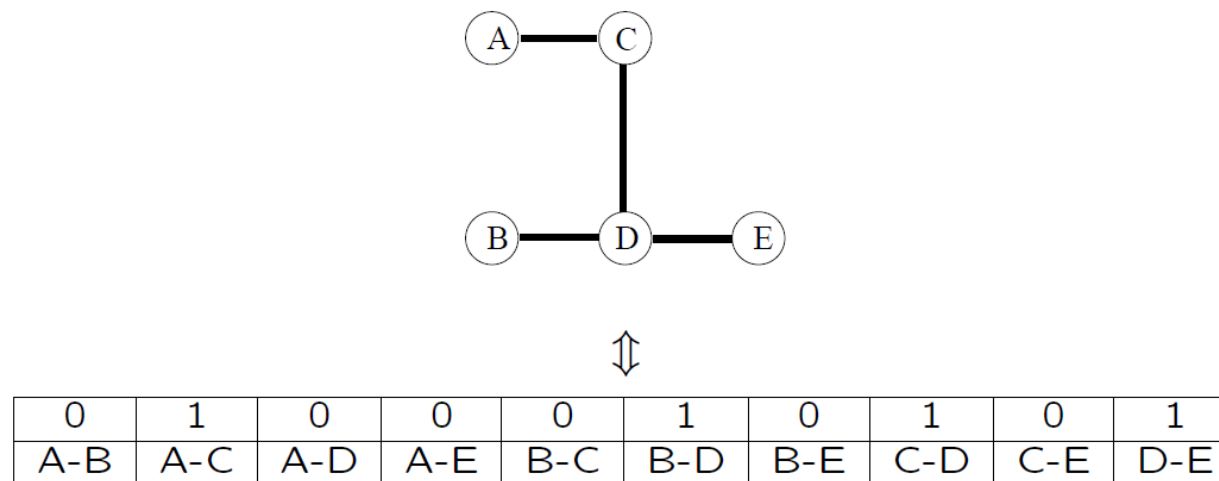
Specific constraints (2)

- Prüfer numbers are a one-to-one mapping between trees and a sequence of integers (like other Cayley codes). A tree with n nodes is represented by a string of length $n-2$ over an alphabet of n symbols.

- Therefore, using Prüfer numbers allows us to consider the constraint that the graph is a tree (For other representations repair operators are necessary).

Prüfer number:

2 2 3 3

133
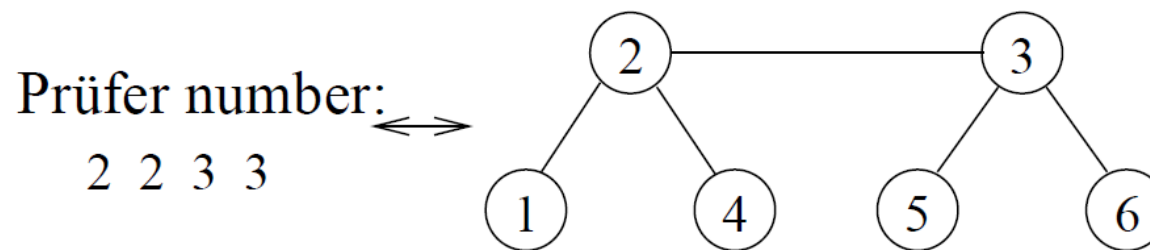
Standardized operators

- When mapping many different types of phenotypes on only a few types of different genotypes (binary, integer, or continuous representations), it is possible to use standardized operators.

- Behavior of EAs for standard representations like binary (simple GAs) or continuous (evolution strategies) representations well understood.

- Mapping phenotypes on binary genotypes allows the use of schemata and effective linkage learning GAs (under the assumption that the problem still remains decomposable and that binary encodings allow a natural encoding of the problem).

Problem-specific operators

- Developing of problem-specific operators is difficult and often additional repair mechanisms must be used to ensure a valid solutiona
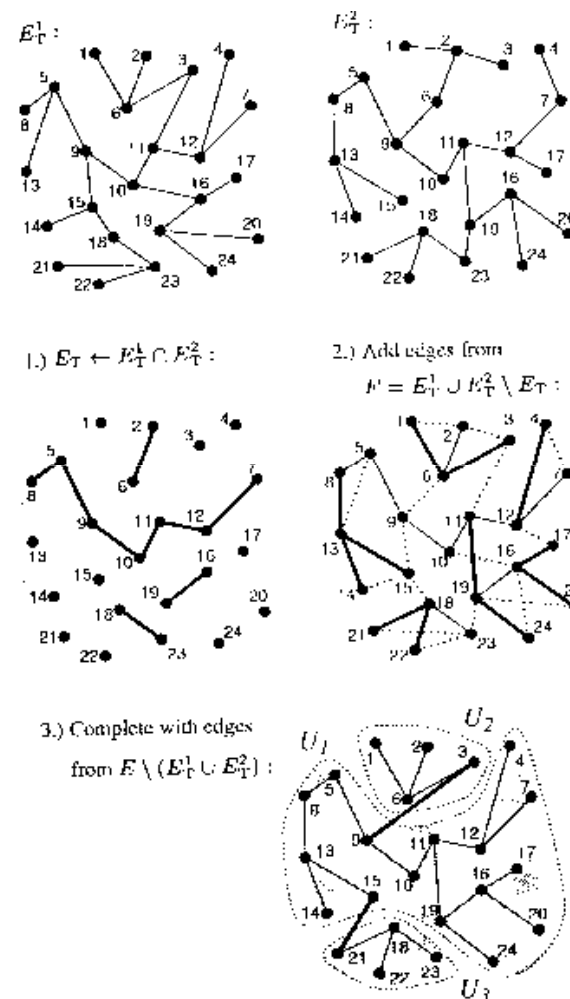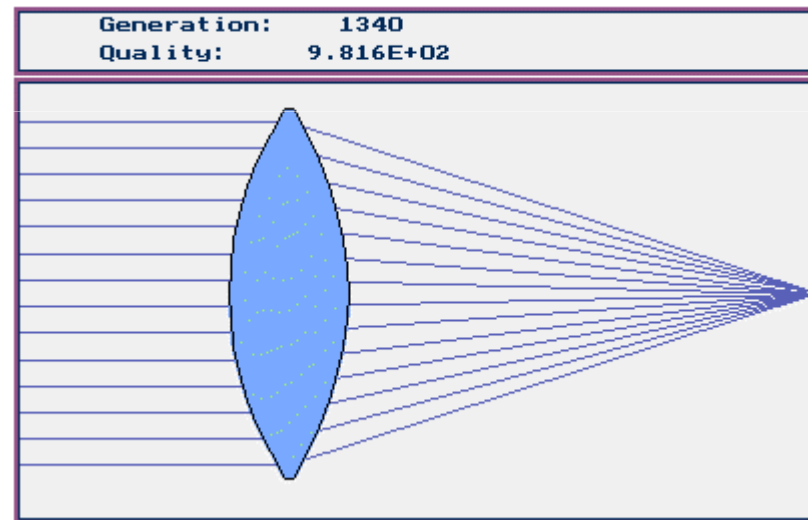

Figure 2: An example for edge crossover (d = 3).

Problem-specific operators (2)

- For some types of problems no problem-specific operators exist that can be applied to direct representations

Indirect Representations - Problem-specific Knowledge

- **Incorporating problem-specific knowledge in the representations to increase GA performance:**

  – Increase the initial supply of solutions that are similar to the optimal solution.

  – Use high-locality representations for easy problems.

  – Consider specific properties of the optimal solution (e.g. stars and trees).

  – Use representations that make a problem easier for a specificoptimization method.

Goldberg's Recommendations

- Principle of meaningful building blocks: The schemata should be short, of low order, and relatively unrelated to schemata over other fixed positions.

- Principle of minimal alphabets: The alphabet of the encoding should be as small as possible while still allowing a natural representation of solutions

138

Goldberg's Recommendations (2)

- The recommendations caused a lot of critics
    - What is a natural representation of a problem? (For example, is using binary representations for encoding real-valued phenotypes a natural representation?)
- Principles mainly aimed at binary representations and crossoverbased GAs that process schemata. No big help for other search methods like evolution strategies or evolutionary programming as these search methods do not process schema.

Radcliffe's recommendations

- Representation and operators belong together and can not be separated from each other.

- Design of representation-independent evolutionary algorithms is possible if the following properties are considered
  - Respect: Offspring produced by recombination are members of all formae to which both their parents belong.
  - Transmission: Every gene is set to an allele which is taken from one of the parents.
  - Assortment: Offspring can be formed with any compatible characteristics taken from the parents.
  - Ergodicity: Iterative use of operators allows the search method to reach any point in the search space.

Representation Invariant Genetic Operators

- Fact: Performance of genetic algorithms using one-point crossover depends on order of objects (e.g. knapsack problem). Thus, one-point crossover is not invariant under changes in the order of objects.

- Evolutionary operators are invariant with respect to a set of representations if EA performance is independent of used representation (how objects are encoded).

- Rowe proposes an approach to generate invariant search operators.

- Examples for appropriate (representation-independent) search operators for some types of problems (subset problems, permutation problems, and balanced partition problems).

Palmer's Recommendations

- An encoding should be able to represent all possible phenotypes.

- An encoding should be unbiased in the sense that all possible individuals are equally represented in the set of all possible genotypic individuals.

- An encoding should encode no infeasible solutions.

- The decoding of the phenotype from the genotype should be easy.

- An encoding should possess locality. Small changes in th genotype should result in small changes in the phenotyp (compare statements about metric).

Ronald's recommendations

- Encodings should be adjusted to a set of genetic operators in a way that the building blocks are preserved from the parents to the offspring

- Encodings should minimize nonlinearities in fitness functions. This means, representations should make the problem easier (for local search methods!).

- Feasible solutions should be preferred.

Ronald's recommendations (2)

- The problem should be represented at the correct level of Abstraction.

- Encodings should exploit an appropriate genotype-phenotype mapping process if a simple mapping to the phenotype is not possible.

- Isomorphic forms, where the phenotype of an individual is encoded with more than one genotype, should not be used.

Design Guidelines - Summary

- Based on observations for specific test problems there are some common, fuzzy ideas about what is a good representation.

- Some recommendations are too general to be helpful for designing or evaluating representations.

- Analytical models describing the influence of representations on EAs are on their way.

- To verify (or reject) observations analytical models are necessary.

Design Guidelines - Summary

- Based on observations for specific test problems there are some common, fuzzy ideas about what is a good representation.

- Some recommendations are too general to be helpful for designing or evaluating representations.

- Analytical models describing the influence of representations on EAs are on their way.

- To verify (or reject) observations analytical models are necessary.

Locality

- Representations (genotype-phenotype mappings) can change the neighborhood and the structure of the fitness landscapes.

- A neighbor can be reached directly by a move (mutation, crossover, etc). Therefore, the neighborhood depends on the used operator/metric.

- The set of neighbors can be different for genotypes and phenotypes.

- The distance between two individuals is determined by the number of moves between both individuals.
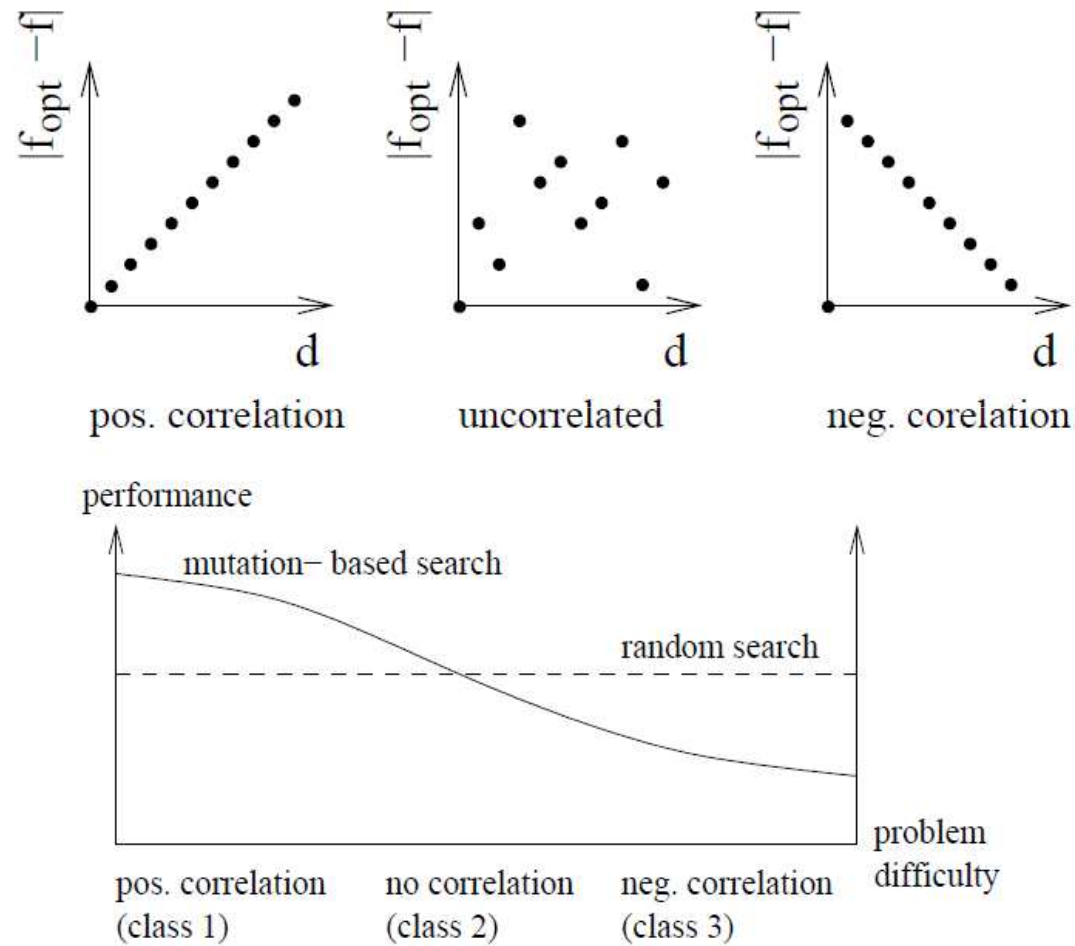
Locality of a Representation

- The locality of a representation describes how well neighboring genotypes correspond to neighboring phenotypes.

- Locality of a representation is high, if neighboring genotypes correspond to neighboring phenotypes.

- Locality, causality, and distance distortion describe how well the metric on Φ_p fits to the metric on Φ_g. If they fit well, locality is high.

- Representations f_g that change the distances between corresponding genotypes and phenotypes modify the performance of particular optimization problems (method performance(f) $\neq$ method performance(f_p)).

Different Phenotype-Fitness Mappings

- Class 1: Fitness difference to optimal solution is positively correlated with the distance to optimal solution. Structure of the search space guides local search methods to the optimal solution → easy for mutation-based search.

- Class 2: No correlation between fitness difference and distance to optimal solution. Structure of the search space provides no information for guided search methods → difficult for guided search methods.

- Class 3: Fitness difference is negatively correlated to distance to optimal solution. Structure of search space misleads local search methods to sub-optimal solutions → deceptive problems
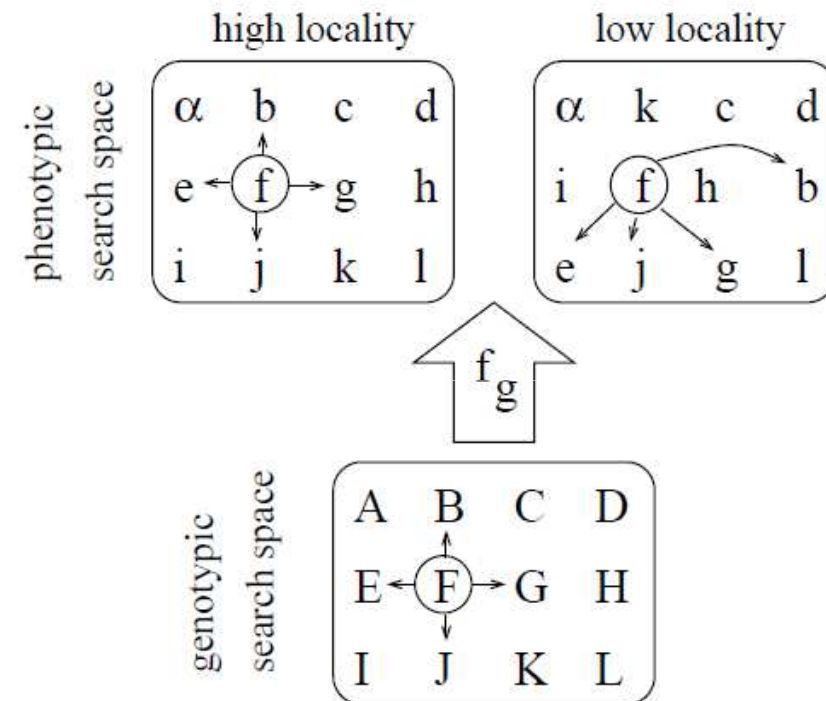
Different Phenotype-Fitness Mappings (2)

Low versus High-Locality Representations
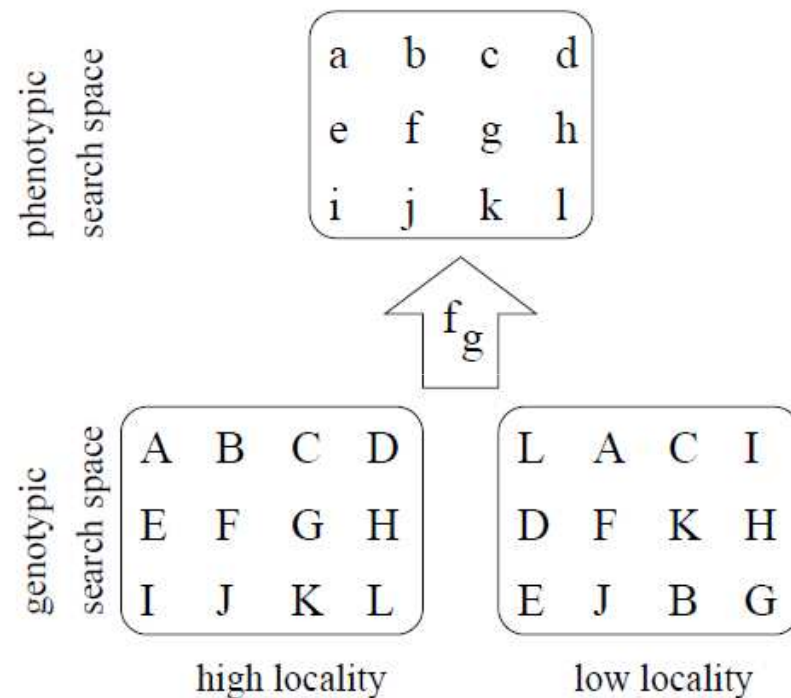
Influence of high versus low-locality representations on genotype-phenotype mappings

Effect of mutation for high versus low-locality representations

151

Low versus High-Locality Representations (2)

- Class 1:
  - High-locality representations preserve difficulty of problem. Easy problems remain easy for guided search.
  - Low-locality representations make easy problems more difficult. Resulting problem becomes of class 2.
- Class 2:
  - High-locality representations preserve difficulty of problem. Problems remain difficult for guided search.
  - Low-locality representations on average do not change class of problem. Problems remain difficult.
- Class 3:
  - High-locality representations preserve deceptiveness of problem. Traps remain traps.
  - Low-locality representations transform problem to class 2 problem. Deceptive problems become more easy to solve for guided search.
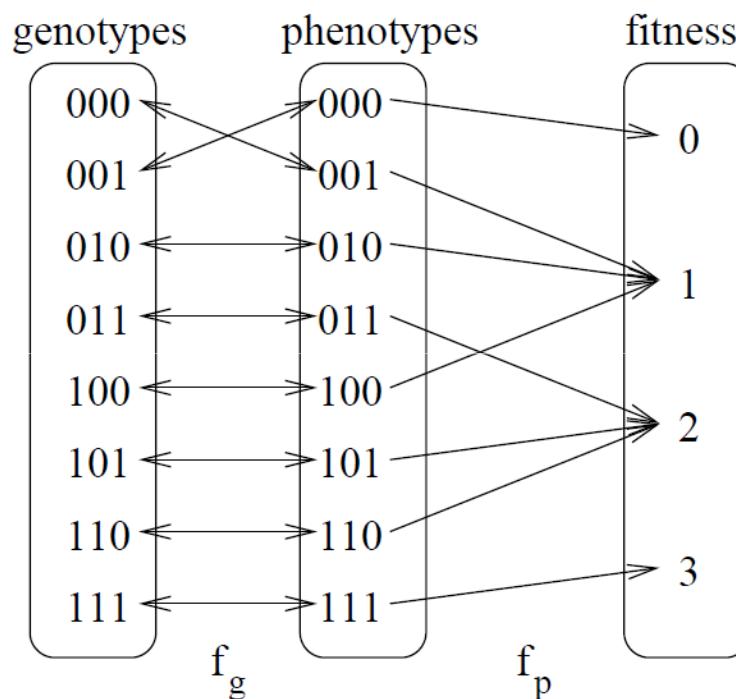
Example

- Both, genotypes and phenotypes are binary.

- We use the bit-flipping operator as a move (Hamming distance).

- One-max problem (class 1).

- All building blocks (regarding genotypes and phenotypes) are of size k=1. Therefore, problem is easy for selectorecombinative GAs
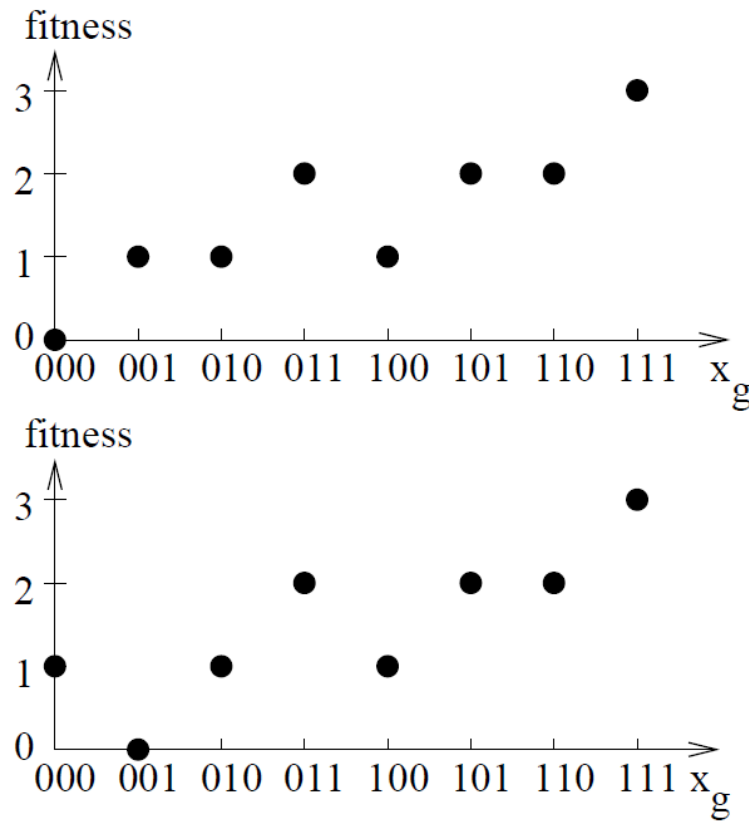

153
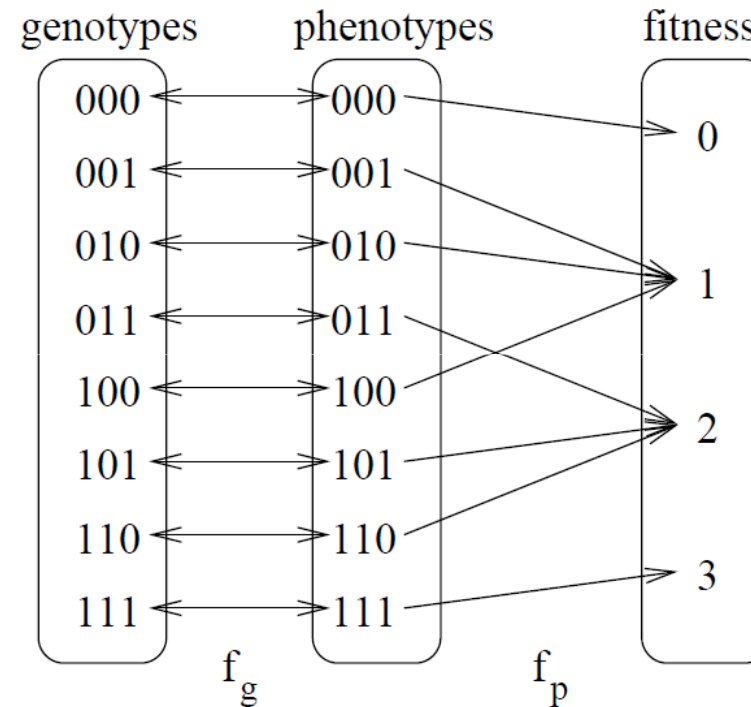
Example

- A representation with lower locality.

- The neighborhood structure changes.

- Not all genotypic building blocks are of size 1. Although, f_p remains unchanged, f becomes more difficult for guided search.


154

Example

- **High-locality representation.**
- **Problem easy for Selectorecombinative GAs.**

- **Different fitness for genotypes 000 and 001.**
- **Problem more difficult for selectorecombinative GAs.**
- **Neighborhood not preserved by representation.**

155

Example

- **Neighborhood structure of the genotypes**

```
              000
      001     010     100

      011     101     110
              111
```

- **Resulting neighborhood structure of phenotypes**

```
              000
      001 — 010     100

      011     101     110
              111
```
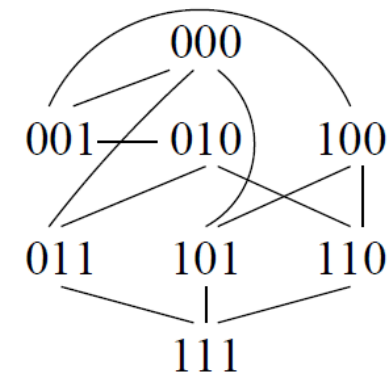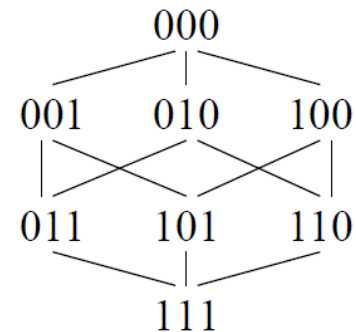
Comparing representations

- We compare the performance of selectorecombinative Gas over all different representations for the one-max problem.

- When focusing on binary bitstrings and assigning l-bit genotypes to l-bit phenotypes, there are $2^l!$ different representations.

- For l=3 there are 8 different genotypes, resp. phenotypes, and 8! = 40, 320 different representations.

- 36 different representations result in the same overall problem f (for the one-max problem).

Comparing representations

- To reduce problem complexity, x_g = 111 is always assigned to x_p=111. Therefore, there are 7! = 5040 different representations.

- We concatenate ten 3-bit problems and use a GA with tournament selection of size 2, uniform crossover, and N=16.

Comparing representations
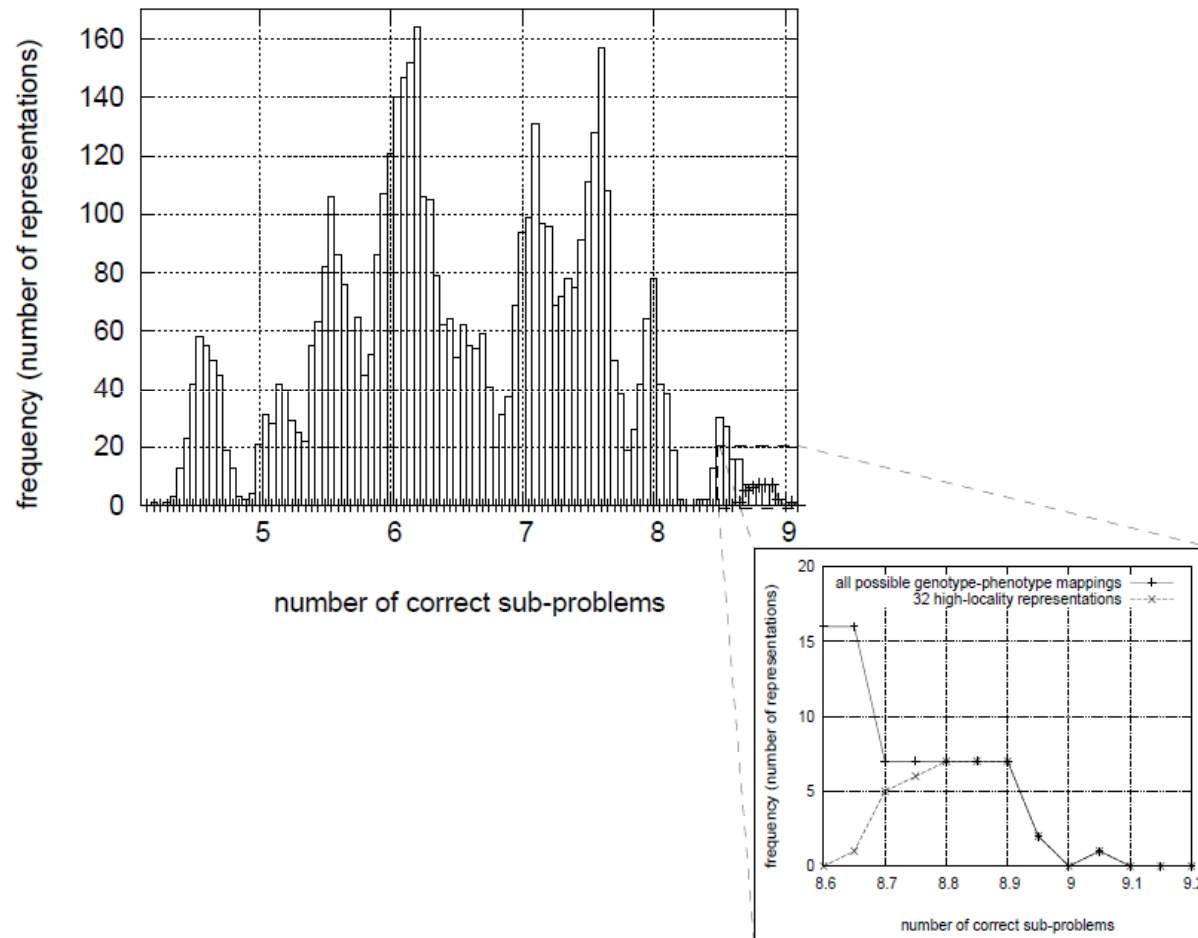
Summary

- When using high locality representations, genotypic neighbors correspond to phenotypic neighbors.

- High locality representations do not change the structure and difficulty of the problem.

  – Easy problems remain easy.

  – Difficult problems remain difficult.

  – Locality depends on the used distance metrics which depend on the used operators.

Redundant representations

- Representations are redundant if the number of genotypes is larger than the number of phenotypes.

  - Using redundant representations f_g means changing $f = f_p(f_g)$. There are additional plateaus in the fitness landscape.

  - Redundant representations are more "inefficient" encodings which use a higher number of alleles but do not increase the amount of encoded information.

  - Redundant representations are not an invention of AI researchers but are commonly used in nature.

Redundant representations (2)

- There are different opinions regarding the influence of redundant representation on the performance of EAs.

- Redundant representations reduce EA performance due to loss of diversity (Davis, 1989; Eshelman and Schaffer, 1991; Ronald et al., 1995)

- Redundant representations increase EA performance (Gerrits and Hogeweg, 1991; Cohoon et al., 1988; Julstrom, 1999)

Redundant representations (3)

- Large amount of work considers the neutral theory (Kimura, 1983). This theory assumes that not natural selection fixing advantageous mutations but the random fixation of neutral mutations is the driving force of molecular evolution.

- Following these ideas redundant representations (neutral networks) have been used in EAs with great enthusiasm.

- There was hope that increasing the evolvability of a system also increases the performance of the system

- This is not true!

Redundant representations (4)

- ## Neutral Network: Set of genotypes connected by single-point mutations that map to the same phenotype

Genotype Space

Different phenotypes encountered along random neutral walk:

P_2 P_5 P_{12}
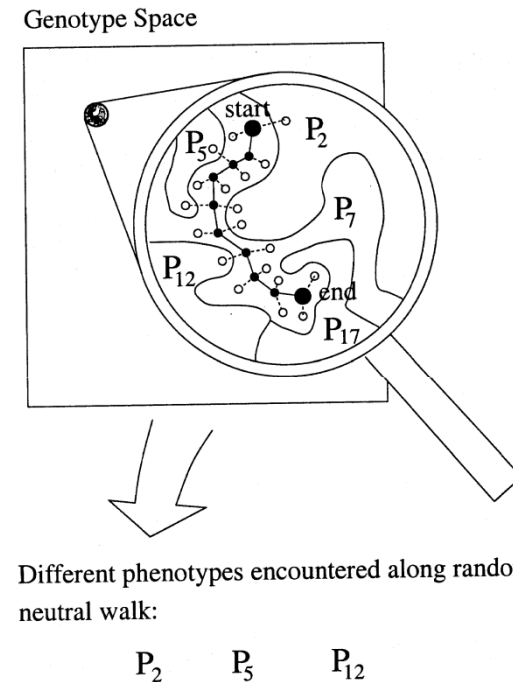
Guide

- ## In the following slides we study

  – how to distinguish between synonymously and non-synonymously redundant encodings

  – how synonymous redundancy changes performance of Eas (quantitative predictions), and

 – the properties of non-synonymously redundant representations

Redundant representations (5)

- **Benefits of Neutral Networks**
  - Population can drift along these neutral networks.
  - Reducing the chance of being trapped in sub-optimal solutions.
  - Population is quickly able to recover after a change has occurred.
  - Evolvability and connectivity of the system increases.
- **Problems**
  - Higher evolvability and connectivity $\rightarrow$ Randomization of search
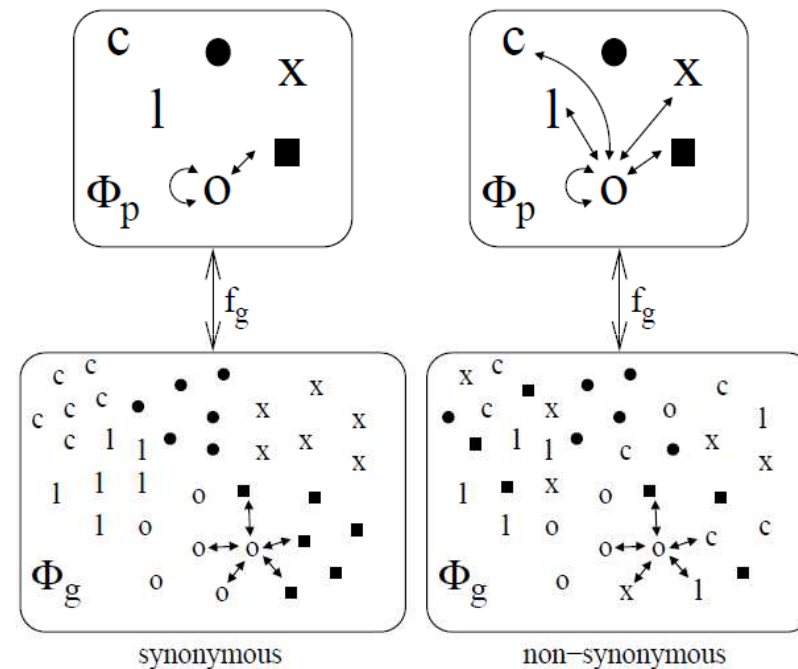 - Genetic drift?

Synonymously versus non-synonymously redundant representations

- **When using redundant representations it can be distinguished between:**

  - Synonymously redundant representations: All genotypes that encode the same phenotype are similar to each other.

  - Non-synonymously redundant representations: Genotypes that encode the same phenotype are not similar to each other.
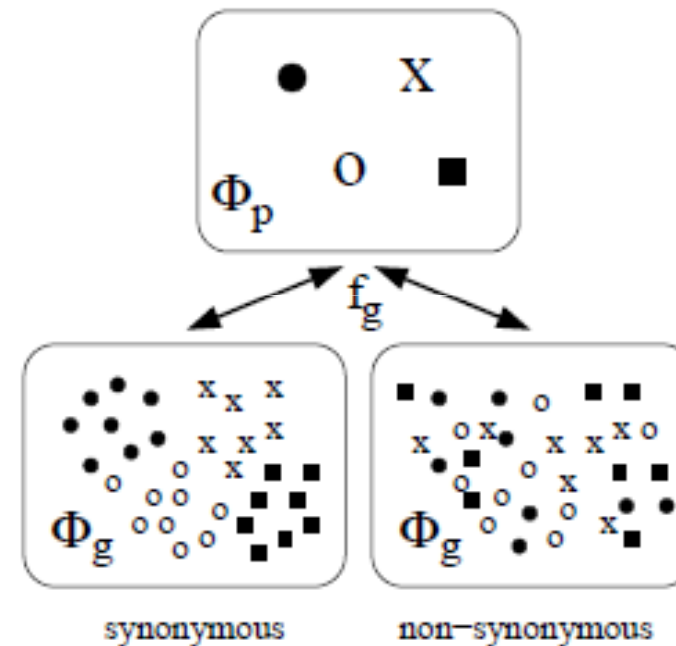
Synonymously versus non-synonymously redundant representations

- **Non-synonymously redundant representations do not allow guided search.**
  - EA search becomes random.
  - Similar effect as low locality representations.

Effects of small mutation steps

168

Synonymously versus non-synonymously redundant representations

- (Choi and Moon, 2003) defined uniformly redundant encodings that are maximally non-synonymous and proved that such encodings induce uncorrelated search spaces (fitness distance correlation is equal to zero).

- For a maximally non-synonymous redundant encoding, the expected distance between any two genotypes that correspond to the same phenotype is invariant and about equal to the problem size n.

- Normalization (transformation of one parent to be consistent with the other) can transform uncorrelated search spaces into correlated search spaces with higher locality.

Synonymously versus non-synonymously redundant representations

- Some selected examples for problems with maximally non-synonymous redundant encodings :
  - Partitioning problems in graphs: k subsets are represented by integers from 0 to $k-1$ where nodes are contained in the same group if they are represented by the same number.
  - Each phenotype is represented by $k!$ different genotypes. HIFF problems (Watson et al., 1998): binary encoding where each phenotype is represented by a pair of bitwise complementary genotypes.
  - TSP: Order-based crossover, in which vertices are indexed from 1 to n and each tour is represented by a permutation of the vertex indices. Each phenotype is represented by $2n$ genotypes

Modeling redundant representations

- Synonymously redundant representations can be described using
  - order of redundancy $k_r = \frac{\log|\Phi_p|}{\log|\Phi_g|}$
  - over-, resp. underrepresentation r of the optimal solution due to the problem representation f_g.

- When using the notion of BBs and binary representations:
  - $k_r = \frac{k_g}{k_p}$
  - r: Number of genotypic BBs of order k_g that represent the optimal phenotypic BB of order k_p.

Modeling redundant representations

- k=2 (order of phenotypic BBs)

- k_r=2 (One allele of a phenotype is represented using two alleles of a genotype)

- Uniform redundancy: r=4 (the best BB (e.g.. x_p = 11) is represented by four genotypic BBs)

genotypes x_g	x_p
00 00, 00 01, 01 00, 01 01	0 0
10 00, 10 01, 11 00, 11 01	1 0
00 10, 01 11, 00 11, 01 11	0 1
10 10, 10 11, 11 10, 11 11	1 1

Modeling redundant representations

- k=1 (order of phenotypic BBs)

- k_r=3 (One phenotypic allele is represented using three genotypic alleles)

- Non-uniform redundancy: r=1 (best BB (x_p = 1) is represented by one genotypic BB (x_g = 111))

genotypes x_g	x_p
000, 001, 010, 100, 101, 110, 011	0
111	1

Population sizing for GAs

- The Gambler's ruin model (Feller, 1957) can be used for modeling the iterated decision making in GAs.

- A gambler with initial stake x_o wishes to increase his funds to a total of N units by making a sequence of bets against a gaming house. Each bet has fixed probability p of winning (q =1$-p$ of losing), and we wish to know the probability of succeeding (getting N units) or failing (losing all units).

- Following (Harik et al., 1997) the probability that a GA with a population size N converges after tconv generations to the correct solution is
$$P_n = \frac{1-(q/p)^{x_0}}{1-(q/p)^N}$$

Population sizing for GAs (2)

- After some calculations we get:

$$N \approx -2^{k-1} \ln(\alpha) \frac{\sigma_{BB}\sqrt{\pi m'}}{d}$$

- N is the necessary population size, α = 1 − P_n the probability P_n that the optimal BB cannot be found (probability of failure) and k is the order of the BBs.

- σ_{BB} (variance of BBs), d (fitness difference between best and second best BB), m' = m−1 (number of BBs) and k are problem-dependent.
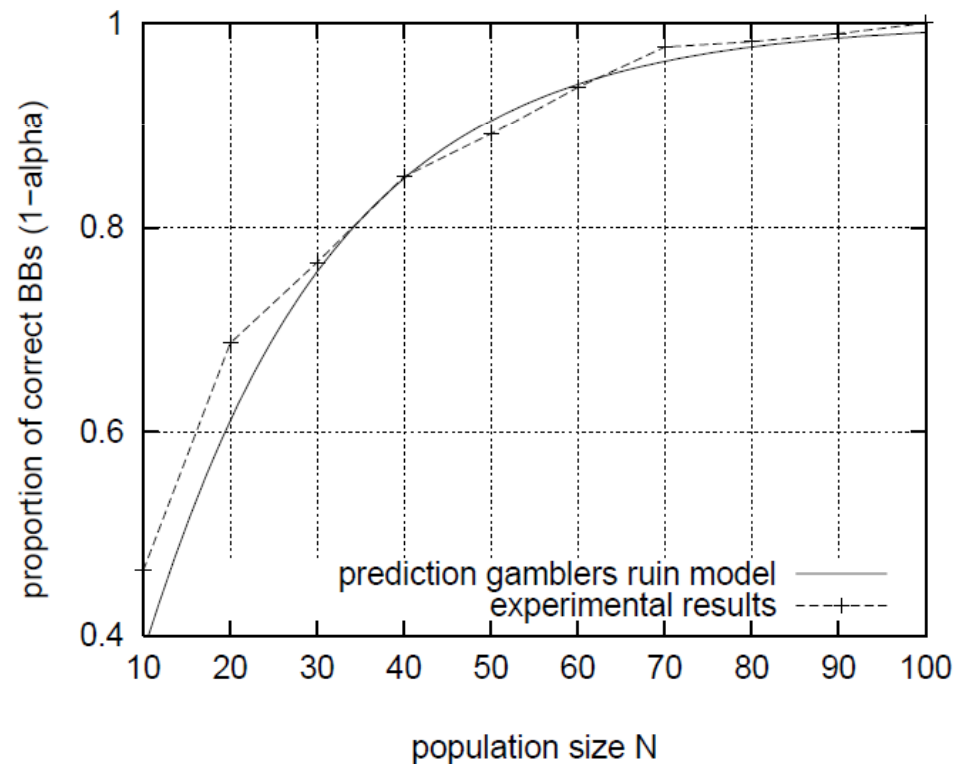
175

Population sizing for GAs (3)

- **150-bit one-max problem**
  (k=1, σ_{BB}=0.25, d=1 and m=150)
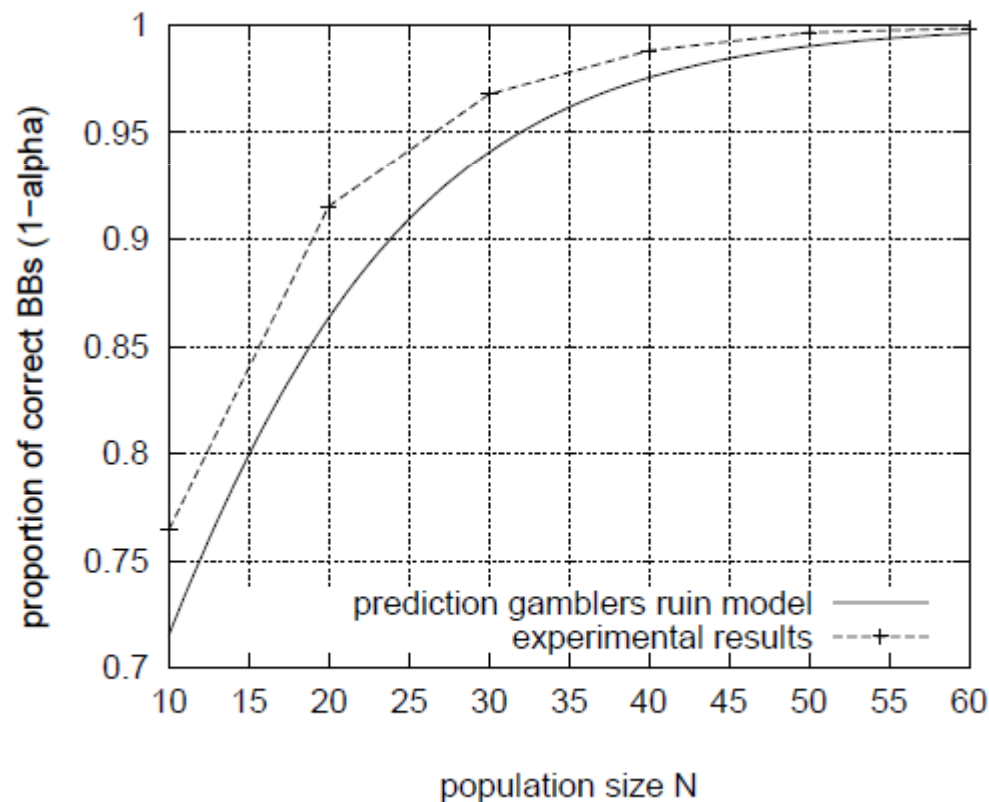
Population sizing for GAs (4)

- **Ten concatenated 3-bit deceptive traps (k=3, σ_{BB} = 1, d=1 and m=10)**

Population sizing for GAs (5)

- Now we have to ask how the redundancy of a representation influences GA performance?

- Observation: Redundant representation change the initial supply x_o of BBs.

- For binary problem representation:

$$x_0 = N \frac{r}{2^k k_r}$$

where N is the population size.

Population sizing for GAs (6)

- When using synonymously redundant representations the existing model can be extended:

$$N \approx -\frac{2^{k_r k - 1}}{r} \ln(\alpha) \frac{\sigma_{BB} \sqrt{\pi m'}}{d}$$

- The population size N that is necessary to find the optimal solution with probability $P_n = 1 - \alpha$

  goes with $O\left(\frac{2^{k_r}}{r}\right)$

Population sizing for GAs (7)

- **Conclusions from this model:**

    - Redundant representations can change the performance of EAs.

    - If representations are synonymously redundant:

        - Uniformly redundant representations do not change the performance of EAs!

        - If the optimal BB is overrepresented GA performance increases.

        - If the optimal BB is underrepresented GA performance decreases.

- **Redundant representations can not be used systematically if there is no problem-specific knowledge!**

Example: Trivial voting mapping

- The trivial voting mapping (TVM) assigns binary phenotypes to binary genotypes.

- One bit of the phenotype is represented by k_r genotypic bits.

- In general, a phenotypic bit is 0 if less than u genotypic bits are zero. If more than u genotypic bits are 1 then the phenotypic bit is 1.

- For u=k_r/2 the value of the phenotypic bit is determined by the majority of the genotypic bits (majority vote)

- In general:

$$
x_i^p = \begin{cases} 0 \text{ if } & \sum_{j=0}^{k_r-1} x_{k_r i+j}^g < u \\ 1 \text{ if } & \sum_{j=0}^{k_r-1} x_{k_r i+j}^g \geq u, \end{cases}
$$

where u$\in$ {1,...,k_r}.

Examples

- k=1

- k_r=3

- u=2

genotypes x_g	x_p
000, 001, 010, 100	0
110, 101, 011, 111	1

- k=1

- k_r =3

- u=1

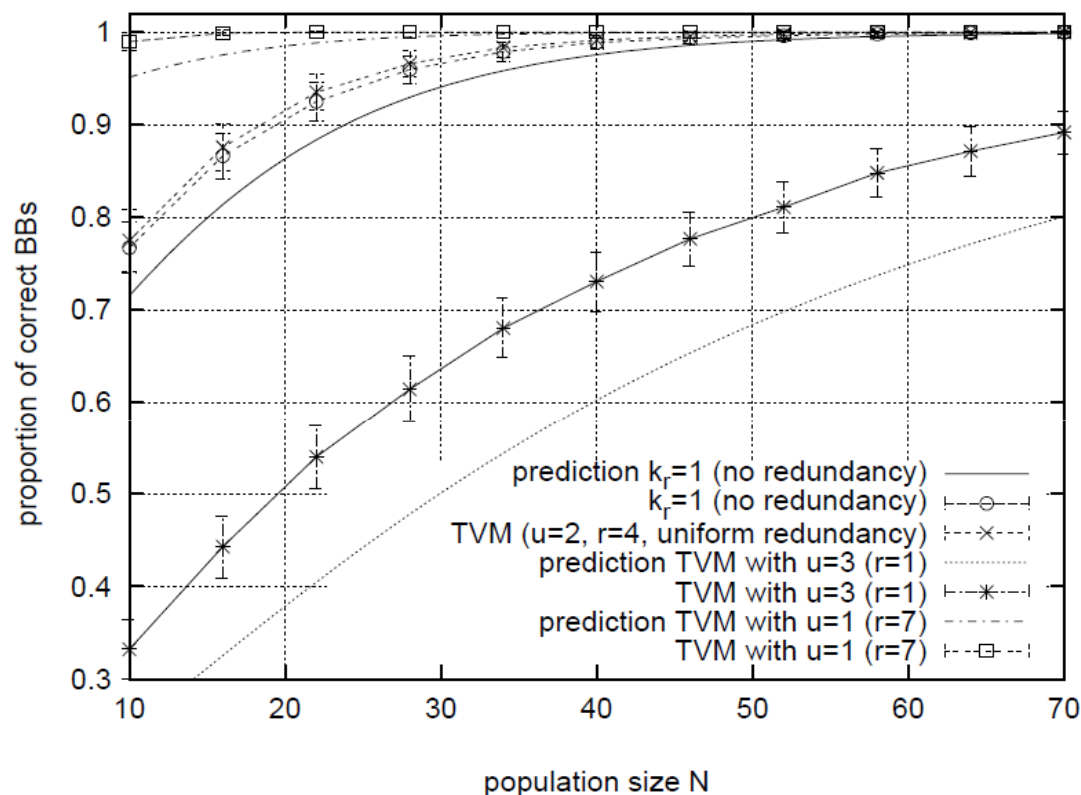| genotypes x_g	x_p
000	0
001, 010, 100,110, 101, 011, 111	1

Trivial voting mapping (3)
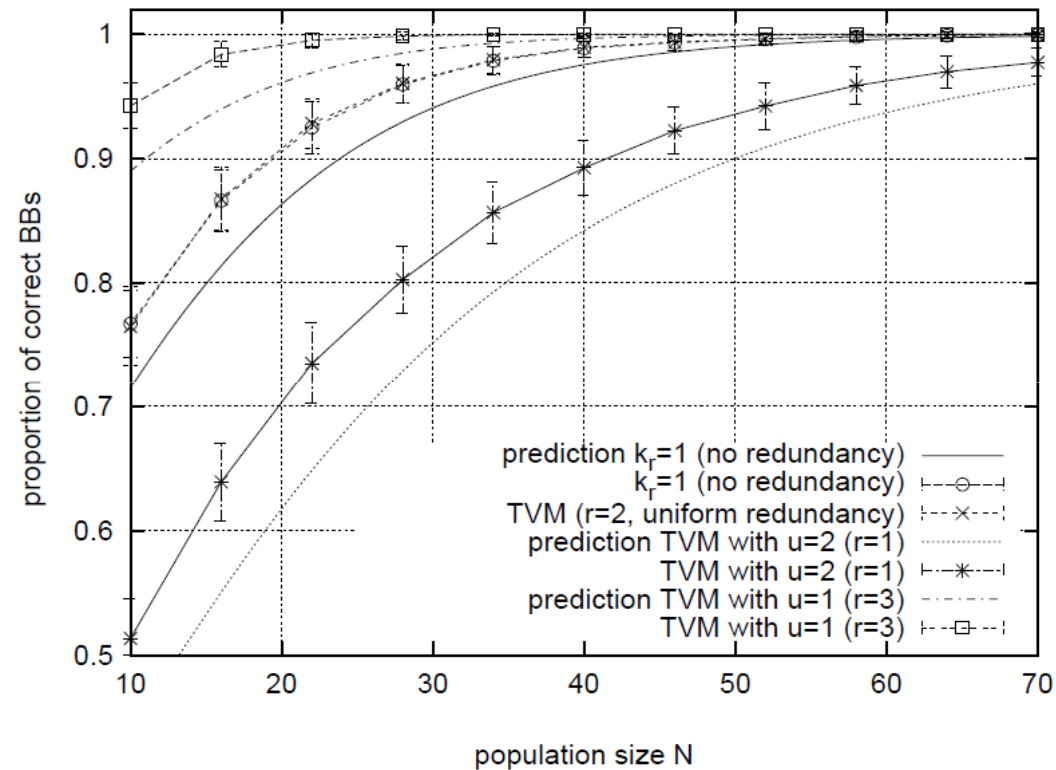
- Experimental and theoretical results of the proportion of correct BBs on a 150-bit one-max problem using the trivial voting mapping for k_r=2.
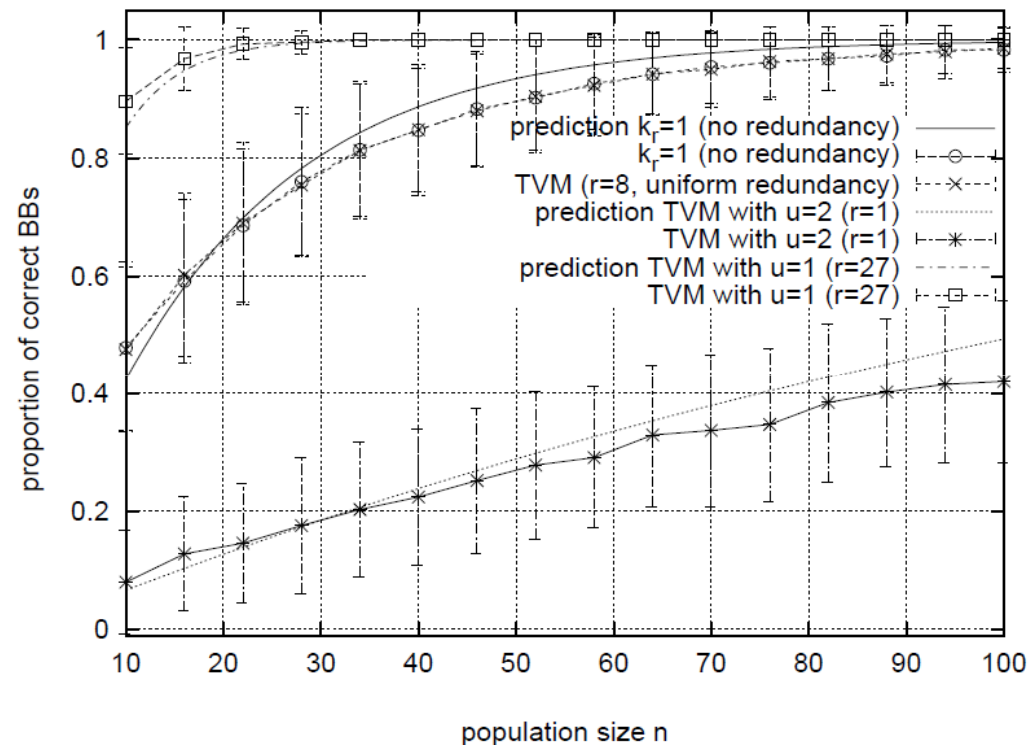
Trivial voting mapping (4)

- Experimental and theoretical results of the proportion of correct BBs on a 150-bit one-max problem using the trivial voting mapping for k_r=3.

Trivial voting mapping (5)

- Experimental and theoretical results of the proportion of correct BBs for ten concatenated 3-bit deceptive traps and $k_r = 2$.
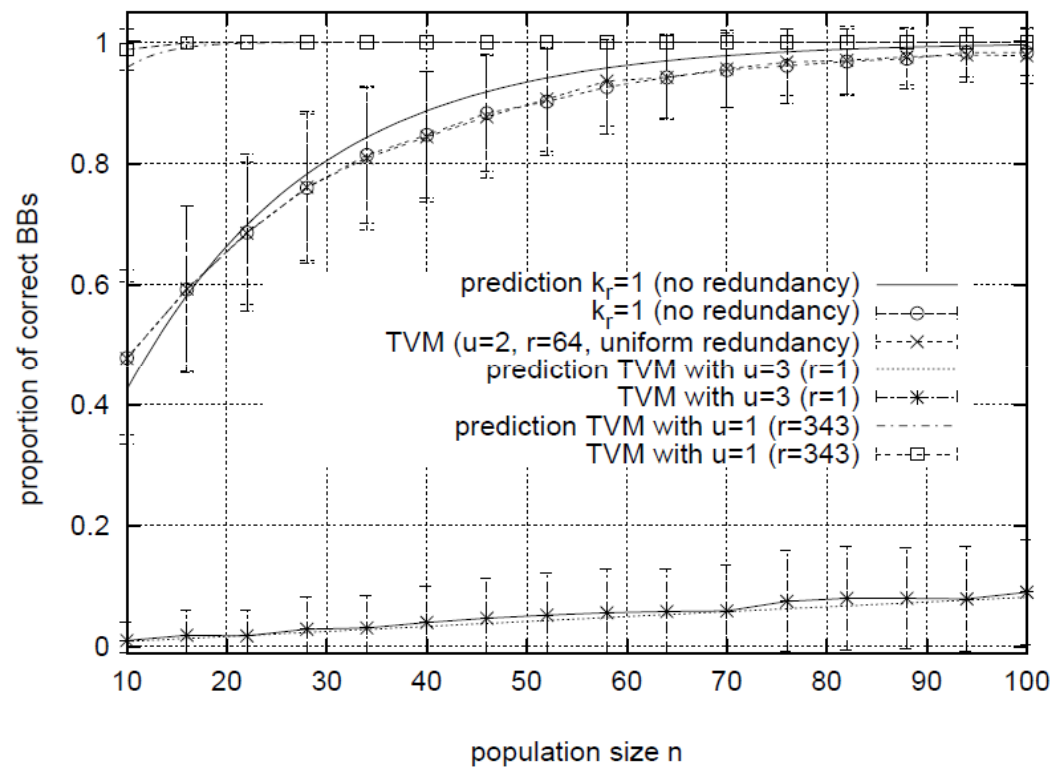
Trivial voting mapping (6)

- Experimental and theoretical results of the proportion of correct BBs for ten concatenated 3-bit deceptive traps and k_r = 3.

Population sizing for GAs (8)

- ## What must be considered when using redundant representations?

  - ### How does the used representation change the size of the search space?

  - ### Is the representation synonymously redundant?

  - ### Are some solutions overrepresented?

- ## Examining these properties allows the user to increase the performance of EAs!

Summary

- There are theoretical models that allow us to predict the expected GA performance when using redundant representations ($N = \mathrm{O}(2^{kr}/r)$).

- There are guidelines for the design of redundant representations:

  - Do not use non-synonymously redundant representations!

  - If you use redundant representations you have to investigate:

    - How does the representation change the size of the search space?

    - Are solutions similar to the optimal solution overrepresented?

- If there is no knowledge about the optimal solution use a uniformly redundant representation.

Search Operators

1. Design Principles
   1. Local Search Operators
   2. Recombination Operators
2. Standard Search Operators

Recap: Design of Search Operators

- Mutation:
  - The application of mutation to an individual results in a new individual with similar properties. There is a small distance between offspring and parent.

- Crossover:
  - Crossover combines the properties of two or more parents in an offspring. The distance between offspring and parent should be smaller than the distance between both parents.

Local search operators

- Goal: find fitter individual by performing neighborhood search
- Local search creates offspring that are similar to parents
- Metric and operator thus depend on each other
- A metric defines possible local search operators and a local search operator determines the metric
- Assumptions:
  - structure of metric/fitness landscape has to guide search towards optimal solution
  - Good solutions can be found by a series of small steps
  - Good solutions are typically clustered, so that they can be found in the neighborhoods of other good solutions

Local search for binary genotypes

- Distance between two solutions often measured by Hamming distance $d(x,y)$

- Local search usually generates solutions with d=1

- „Standard mutation" or „bit flipping"

- l-bit string has l neighbors

Local search for integer genotypes

- **For different metrics, different operators are necessary**

- **Binary Hamming metric**
    - Two solutions are neighbors if they differ in one decision variable.
    - Operator based on this metric can randomly change one decision variable.
    - Solution $x \in \{0,...,k\}^l$ has lk neighbors.
    - Example: x=(0,0) with x_i = {0,1,2} has four neighboring solutions ((1,0),(2,0),(0,1),(0,2)).

- **City-block metric:**
    - Local search can slightly decrease or increase one of the decision variables (adding +/-1).
    - Each solution of length l has at most $2l$ neighbors. Example: x=(0,0), $x_i \in$ {0,1,2} has 2 neighbors ((1,0),(0,1))

Local search for integer genotypes

- Different when operator exchanges values of two decision variables x_i, x_j

- Using Hamming distance: two neighbors have distance d=2, each solution has at most $\binom{l}{2}$ different neighbors

- x=(3,5,2) has 3 neighbors ((5,3,2),(2,5,3),(3,2,5))

194

Local search for continuous genotypes

- Analogue to integer genotypes

- Hamming distance: assign random variable
  $x_i \in [x_{min}, x_{max}]$ to i-th decision variable

- We can also define exchange operator

- Using city-block metric is a bit more complex
  - Search step should not be too small (we want progress)
  - ... and not too large (offspring should be similar to parent)
  - Add random variable with zero mean: usually Gaussian with μ=0, and standard deviation σ controlling „step-size"
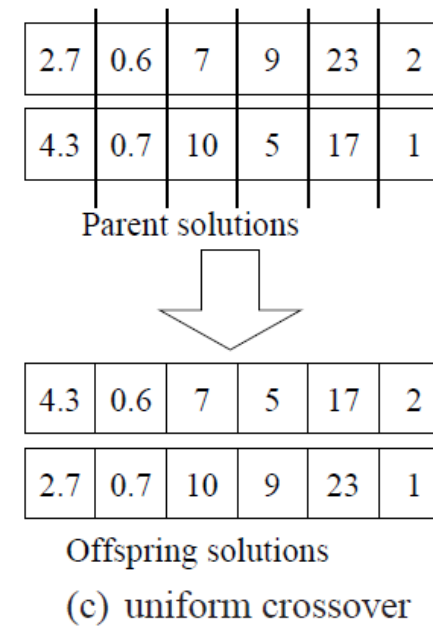
Recombination operators

- Requires a population of solutions
- Goal is to combine meaningful properties from >1 parents
- Like local search, recombination-based search is based on a metric
- Given two parents x^{p1} , x^{p2} and one offspring x^o, recombination should be designed such that

$$\max(d(x^{p1} , x^o), d(x^{p2} , x^o)) \leq d(x^{p1}, x^{p2}))$$

- Offspring should be „between the parents"
- Why use recombination?
  - Real-world problems are often decomposable
  - Large problems can be solved by decomposing it into smaller sub-problems (that are usually easier to solve) and combined to form overall solution

Common recombination operators

(a) one-point crossover

(b) 2-point crossover

(c) uniform crossover
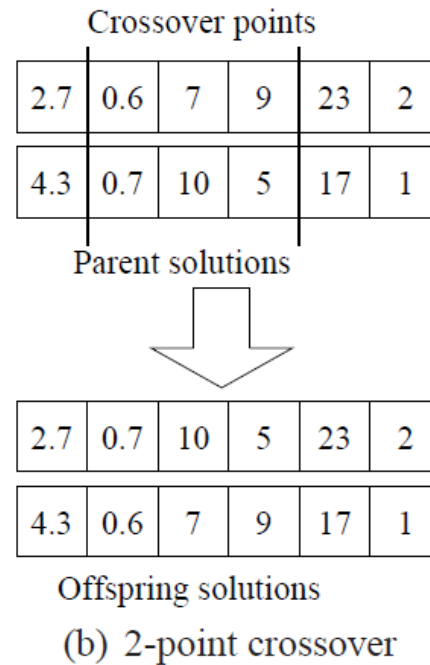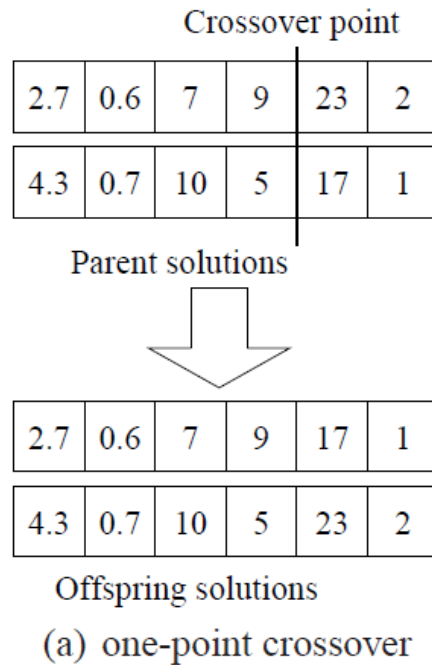
Intermediate recombination

- Uniform and n-point crossover can be applied independent of type of decision variable (binary, discrete, continuous…)

- In contrast, intermediate recombination operators try to blend/average over several parents. We explain arithmetic crossover.

- For two parents x^{p1}, x^{p2}, offspring is
$$x_i^o = \alpha x_i^{p1} + (1-\alpha)x_i^{p2}, \alpha \in [0,1]$$

- For m parents, $x_i^o = \sum_{i=1}^{m} \alpha_i x_i^{pi}, \sum_{i=1}^{m} \alpha_i = 1$

- Takes weighted average of parents' decision variables

Standard search operators

- We provide an overview of operators for standard search spaces

- Can be genotypes or phenotypes (direct representation)

- Ordered by increasing complexity

Strings and Vectors

- Ordered lists of decision variables of fixed or variable length

- Often used

- Appropriate for sequences of characters or patterns. objects are modeled as

  - text,

  - characters,

  - patterns.

- Can use standard local search and recombination-based operators based on (binary) Hamming metric

- If length is variable, Levenshtein distance can be used

Coordinates and Points

- Represent locations in geometric space

- Integer or continuous

- Often: locations of cities or other spots on a 2d-grid

- Appropriate for problems that work on

  - sites,

  - Positions, or

  - locations.

- Standard local and recombination operators for continuous decision variables or integers

- Euclidean metric

Graphs

- Represent relationships between arbitrary objects
- Structure can be described as list of edges (with n nodes, there are $n(n$-1)/2 possible edges)
- Appropriate for problems that seek a
  - network,
  - graph, or
  - relationship
- Common genotype is binary list of length $n(n$-1)/2
- Standard operators are based on Hamming metric: number of different edges
- If no additional constraints: standard search operators are applicable

Subsets

- Selections from a set of object; order of elements in set does not matter

- Given n objects, there are $\binom{n}{k}$ subsets of size k and 2^n different subsets.

- Subset of fixed size k can be represented by an integer vector x of length k, where the x_i indicate the selected objects and $x_i \neq x_j$ for $i \neq j$ and $i,j \in [1,k]$

- Appropriate for problems that seek a

    – cluster, collection, partition, group, packaging, or selection.

- Can use standard local search if each selected object is unique

- Recombination operators are difficult (Falkenauer , 1998; Choi and Moon, 2003)

    – Each subset is represented by $k!$ different genotypes

    – Redundancy

Permutations

- Orderings of items,

- $n!$ permutations of n objects

- Many permutation problems are relevant but NP-hard

- Used in problems that seek an

    – arrangement, tour, ordering, or sequence.

- Design of operators is demanding

- Often, an integer genotype of length n is used, where x_i denotes an object and has a unique value

- Standard operators fail: offspring usually is no permutation

Permutations

- **Permutation-specific operators are based on absolute or relative ordering.**
  - Absolute ordering: Two solutions are similar, if objects have same absolute position
  - Relative ordering: Two solutions are similar, if relative order of pairs is similar

- **Order crossover, partially matched crossover, … there are many operators specifically designed for permutation problems.**

- **See Whitley (1997), Mattfeld (1996), or Choi and Moon (2008)**

Fitness Function

1. Design Guidelines
2. Examples

Fitness function and objective function

- Fitness function is quality of solution as „seen by the heuristic"

- Objective function (evaluation function) is based on problem model

- In general, fitness function and objective function are the same, but we can modify fitness function to make search easier for a modern heuristic

- Then, we would not use original objective function from model, but a variant thereof

Ordinal and numerical ranking

- Fitness and objective functions can be ordinal or numerical
- Ordinal functions order solutions in a sequence
  - Allow us to compare quality (best, second-best, …, worst)
  - No absolute value of quality available
  - Often used when fitness is evaluated by human experts who rank alternatives
- Numerical objective functions assign a real-valued objective value to all solutions
  - Ordering possible
  - Absolute value is available
  - Standard for most mathematical models of cost, profit, lengths...

Design of objective function

- Make sure:
  - Best solution should have highest quality
  - Should make problem straightforward for local search
  - Should make problem decomposable for recombining methods
- In general, dissimilarity (measured by problem metric) should be positively correlated with difference in objective values

Example 1: Needle in haystack

- Search space X of size n

- Objective function assigns highest value to best solution ($f(max_x\ x)$=n)

- All other n-1 solutions get random objective function in {1,...,n-1}

- No guidance for local search

- Guided search methods will perform like random search

Example 2: Maximum Satisfiability (SAT)

- **Instance is Boolean formula with three elements**
  - Set of n variables x_i , i = {1,...,n}
  - Set of literals. A literal is a variable or a negotiation of it
  - Set of m distinct clauses {C_1, ..., C_m}. Each clause consists only of literals combined by logical *or* operators

- **SAT is decision problem: It asks whether an assignment to the x_i exists, so that $C_1 \wedge C_2 \wedge$ ... C_m is true.**

211

Fitness functions for SAT

- **Obvious choice:**
  - Fitness function of 1 for all assignments that satisfies compound statement
  - 0 otherwise
  - Needle in a haystack!

- **Better choice:**
  - Measure number of satisfied clauses, use as fitness
  - Smoother landscape

Fitness smoothing

- Fitness functions with large plateaus can be made easier for guided search if we modify objective function and consider objective function value of neighbors

- Smoothing the fitness landscape is a possible way to achieve this
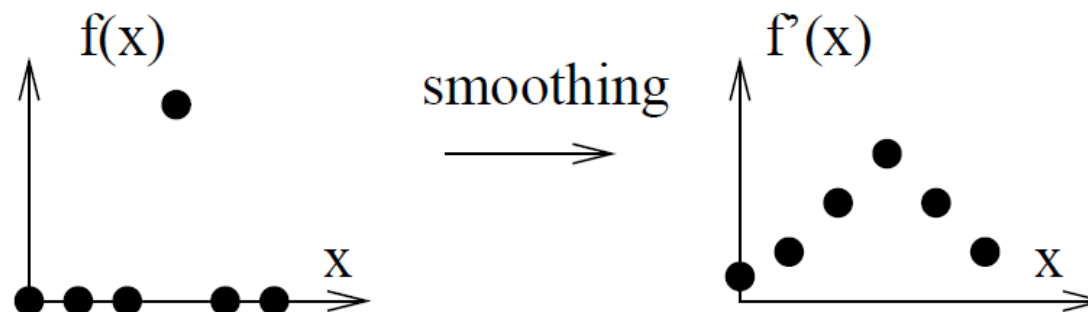
Constraint handling

- We have to assign fitness values to solutions that are infeasible

- Necessary if we cannot exclude all infeasible solutions from search space

- Simplest choice: penalize violation of constraints

- Penalty functions depend highly on parameters

- Proper design of fitness function for constrained problem is demanding

Accuracy-efficiency tradeoff

- **Evaluation of fitness functions must be fast**

- **If calculation of fitness is time-consuming**
  - Rough approximation of fitness function could be used at beginning
  - Increase precision during run
  - Invest more time at the end of run
  - Local search: compute only fitness change (if faster and no need for complete fitness evaluation exists)

Initialization

Impact of initialization

- Proper choice of initial solutions has large effect on efficiency of modern heuristics

- Initial solutions are starting points for search

- For guided search: single solution

- For recombining methods: population of solutions

Random initialization

- Proper choice of initial solutions depends on amount of problem-specific knowledge

- If we know no properties of high- or low-quality solutions, we recommend random initialization

- All solutions are created with same probability, covering the entire search space

- Unbiased sampling
  - might be hard to ensure for direct representations
  - easier for standard genotypes (integer, binary...)

Using problem specific knowledge

- If we know something about structure of high- or low-quality solutions like
  - variable ranges,
  - Dependencies, or
  - good solution parts

  we can use such solutions as initial solutions
- Advantage: guides the search into direction of optima
- Disadvantage: reduces search range
- Too much bias can lead to premature convergence because
  - Solutions are too similar (recombining methods)
  - search focuses too much on certain parts of search space (recombining methods, and local search)

Search Strategies

1. Diversification and Intensification
2. Example: VNS
3. Example: Genetic Algorithm

Intensification and Diversification

- Different strategies for controlling search differ in the design and control of the intensification and diversification phases

- Search strategies must balance intensification and diversification during search and to allow search methods to escape from local optima.

- This is achieved by various diversification techniques based on the **representation**, **search operator**, **fitness function**, **initialization**, or **explicit diversification steps** controlled by the search strategy.

221

Local and Recombination-based Search

- Two fundamental concepts for heuristic search:
    - local search methods versus (high-locality problems)
    - recombination-based search methods (decomposable problems)
- Many real-world problems have high locality and are decomposable
- Direct comparisons between local and recombination-based search is only meaningful for particular problem instances
- General statements on the superiority of one or other of these basic concepts are unjustified as method performance depends on the specific characteristics of the problem (locality versus decomposability).

Strategies for Intensification

- Intensification steps use the fitness of solutions to control search and usually ensure that the search moves in the direction of solutions with higher fitness.

- Keep the high-quality solutions or discard the low-quality ones

- No heuristic search possible without selection

- Intensification too strong (high selection pressure)
  - Premature convergence
  - Search gets stuck in local optimum

- Intensification too weak (low selection pressure)
  - Drift
  - High running times and low progress

- Optimal strength of intensification is problem-specific (example: evolution strategies)

Strategies for Diversification: Representation and Search Operators

- Choosing a combination of representation and search operators is equivalent to defining a metric on the search space

- Representation/operator combination defines which solutions are neighbors.

- By using different types of neighborhoods, it is possible to escape from local optima and explore larger areas of the search space.

- Different neighborhoods can be the result of different genotype-phenotype mappings or search operators applied during search.

- Standard examples for local search approaches that use modifications of representations or operators to diversify the search are variable neighborhood search, problem space search, the rollout algorithm, or the pilot method.

Strategies for Diversification: Fitness Function

- Fitness function measures the quality of solutions.

- Modifying the fitness function has the same effect as changing the representation as it assigns different fitness values to the problem solutions.

- Variations and modifications of the fitness function lead to increased diversification

- Common example is guided local search (it systematically changes the fitness function with respect to the progress of search)

Strategies for Diversification: Initialization

- Search trajectory depends on the choice of the initial solution (for example, greedy search always finds the nearest local optimum)

- Diversification can be the result of search heuristics using different initial solutions.

- Multi-start search approaches explore a larger area of the search space and lead to higher diversification.

- Variants of multi-start approaches include iterated descent, large-step Markov chains, iterated Lin-Kernighan, chained local optimization, or iterated local search.

Strategies for Diversification: Search Strategy

- The search strategy can control the sequence of diversification and intensification steps.

- Diversification steps that do not move towards solutions with higher quality can either be the results of random, larger, search steps or based on information gained in previous search steps.

- Examples of search strategies that use a controlled number of search steps towards solutions of lower quality to increase diversity are simulated annealing, threshold accepting, or stochastic local search.

- Representative examples of search strategies that consider previous search steps for diversification are tabu search or adaptive memory programming.
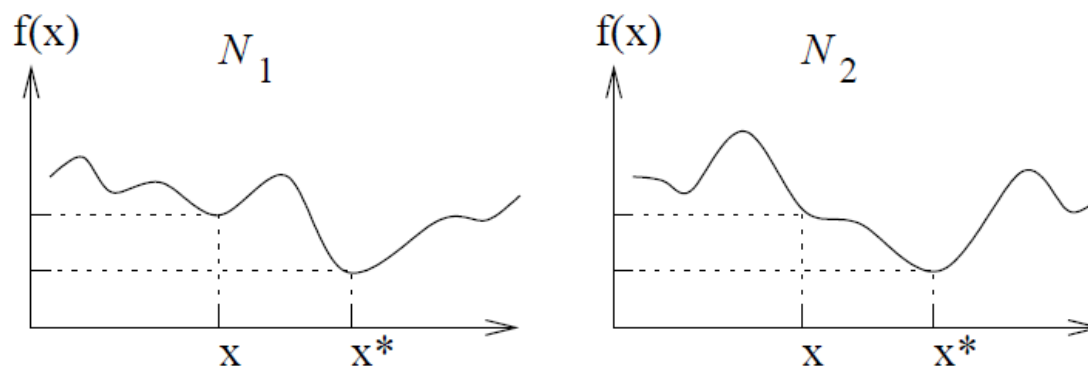
Variable neighborhood search (VNS)

- Combines local search with dynamic neighborhood structures that are changed depending on the progress of search

- Based on the following observations:

  - Local minimum for neighborhood A is not necessarily one for neighborhood B. Different neighborhoods result in different metrics, result in different fitness landscapes.

  - A global minimum is a global minimum with respect to all possible neighborhoods. Neighborhoods change definition of solution similarity, but not the fitness.

  - Global optimum is not affected by search operators, but only local optima are affected by search operators!

Variable neighborhood search (VNS)

- Local optima for different neighborhoods are often close to each other

- Local optima have structure and properties that are also relevant for global optimum (similar to decomposability: parts of solution need to be recombined)

- Local optima are not randomly scattered through search space but are clustered together

- Changing the neighborhood from N1 to N2 allows local search to find optimal solution

VNS pseudo-code

Algorithm 2 Variable Neighborhood Search

Select a set of neighborhood structures N_k, $k \in \{1, \dots, k_{max}\}$
Create initial solution x
while termination criterion is not met **do**
    $k = 1$
    while $k < k_{max}$ **do**
        Shaking: choose a random neighbor $x' \in N_k(x)$
        Local search: perform a local search starting with x' and return x'' as the local optimum
        with respect to N_k
        if $f(x'') < f(x)$ (minimization problem) **then**
            $x = x''$
            $k = 1$
        else
            $k = k + 1$
        end if
    end while
end while

VNS in a nutshell

- Define k different neighborhoods
- $|N(x)|_k$ is average number of neighbors
- Usually we order the neighborhoods in increasing $|N(x)|_k$
- VNS iteratively performs a
  - Shaking phase: select a random solution w.r.t. current neighborhood. Avoids cycling and explores new region
  - Local search phase: Perform local search until local optimum is found
  - Switch to next neighborhood
  - Track best found local optimum

231

Intensification and diversification in VNS

- Local search focuses search
- Shaking and switching neighborhoods are diversification steps
- Since average number of neighbors grows with k, diversification gets stronger
  - Local search can select from more neighbors
  - Covers larger areas of search space
  - Sizes of basins of attraction increase
- Although VNS is recently quite popular, the underlying ideas are actually old (see Design of Modern Heuristics, p. 136)

Pilot Method

- "Preferred Iterative LOok ahead Technique"

- Combines greedy construction heuristics with a greedy measure to estimate the global impact of local choice

- Constructs solution step by step (Master solution)

- Decides about the steps by completing the solution using a fast construction heuristic, the so-called pilots

- Pilots look ahead. Similar to A* search

- Master solution evolves, Pilots are iteratively computed

Pilot Method for TSP (1)

- **Combinatorial optimization problem defined on fixed set of elements** E

- **Cost function** $c{:}E{\rightarrow}R$

- **Find minimal cost subset** $S^*{\subset}E$ **satisfying some constraints**

- **TSP: find set of edges with minimal cost that forms a cycle**

- **Assume that a heuristic for the problem is known that can complete any partial solution**

Pilot Method for TSP (2)

- Master solution M contains elements e

- For each $e \in M$ extend partial solution M to a complete solution such that *e* is part of it

- Let $p(e)$ denote objective function value of this solution

- e_0 is most promising pilot: minimal value $p()$ from all $e \in M$.

- Include e into M. Start over.

- Terminate when M is complete.

Properties

- **Greedy construction heuristic that "looks ahead"; no search through a search space**

- **Completion heuristic is designed such that it exploits problem knowledge**

- **No explicit diversification steps**

- **See chapter on design principles (ordinal representation)**

Evolution strategies (ES)

- Local search for continuous search spaces

- Developed by Rechenberg and Schwefel, TU Berlin

- First applications: optimize shape of a bent pipe

- The main search operator in ES is mutation. To exchange information between solutions, recombination operators are used in population-based ES as background search operators.

(1+1)-ES

- n-dimensional continuous vector $x \in R$ is solution

- Creates offspring x' by adding n-dimensional random variable, mostly Gaussian:

    $x_i' = x_i + \sigma \, \mathsf{N}_i(0,1)$

- Offspring replaces parent if it is better

- Stochastic hillclimber

Two standard problem models

- **Corridor model (x is far from optimum)**

$$f_{\text{corr}}(x) = c_0 + c_1\, x_1 \;\; \forall\, i \in \{2,...,n\}: \text{-}b/2 \le x_i \le b/2$$

- **Sphere model (x is close to optimum)**

$$f_{\text{sphe}}(x) = c_0 + c_1 \sum_i (x_i - x_i{*})^2$$

1/5th success rule

- Define $\xi(t)$ as ratio of successful steps over all t search steps

- For corridor and sphere model, $\xi(t)$=0.2 maximizes convergence speed

- If $\xi(t)$>0.2 reduce σ, increase otherwise

$(\mu+\lambda)$, (μ,λ)- evolution strategies

- μ parents generate λ new solutions.

- μ offspring are chosen either from $\mu+\lambda$ $((\mu+\lambda)$-ES) or from the λ new solutions $((\mu,\lambda)$-ES)

- Each individual has an own σ.

- Self-adaptation of strategy parameters (σ is not externally controlled).

- Step size is adapted (learned) during the run

Properties

- ES incorporate the most important parameters of the strategy, e.g. standard deviations, into the search process. Thus, optimization not only takes place on object variables, but also on strategy parameters (self-adaption).

- In population-based ES, intensification is a result of the selection mechanism which prefers high-quality solutions.

- Recombination is a background operator which has both diversifying and intensifying character.
    - By recombining two solutions, new solutions are created which lead to a more diversified population.
    - However, especially intermediate crossover leads to reduced diversity during an ES run since the population converges to the mean values.

- Like for (1+1)-ES, the main source of diversification is mutation. With larger standard deviations, diversification gets stronger as the step size increases. The balance between diversification and intensification is maintained by the self-adaptation of the strategy parameters.

State of the art

- ES are state-of-the-art for many nonlinear continuous functions with medium dimensionality (n<100)

- CMA-ES (Covariance Matrix Adaptation Evolution Strategy) from Hansen and co-workers
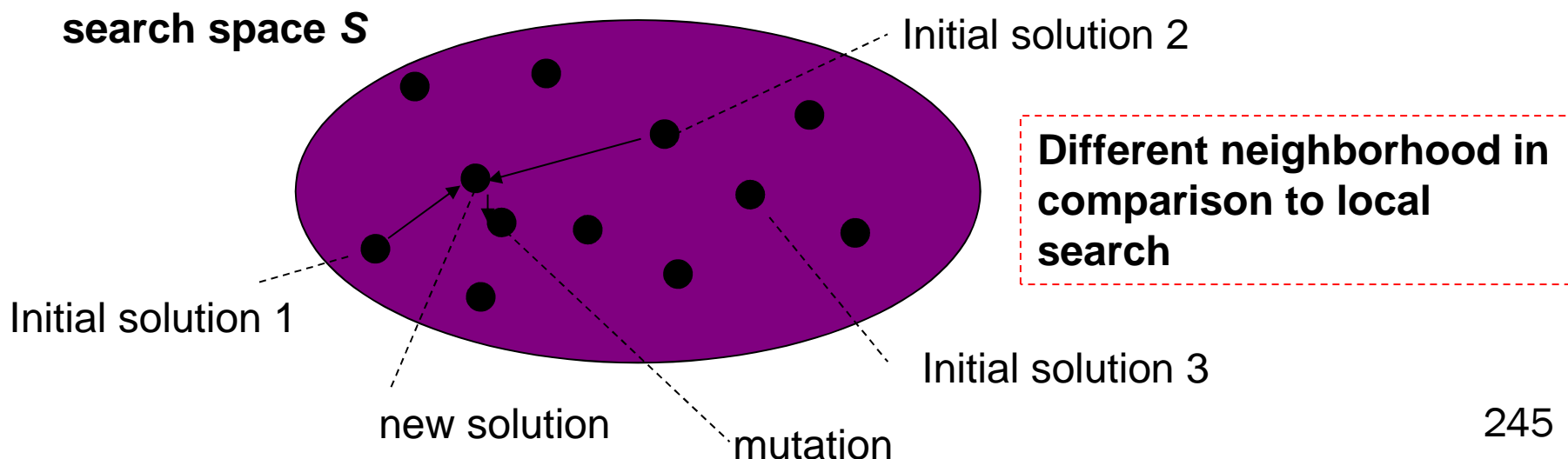

243
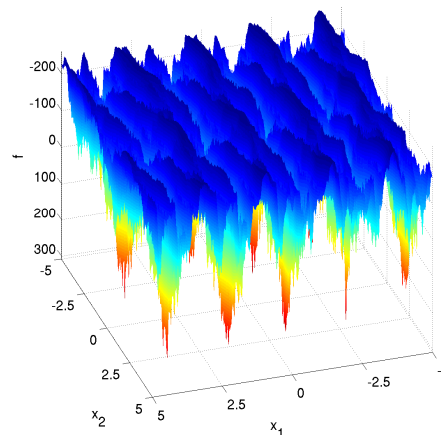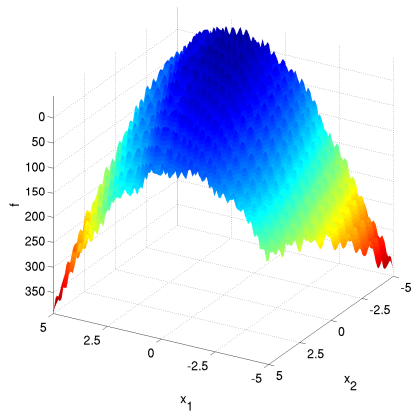
Principles of Genetic Algorithms

- **Population of solutions.**
  - properties of a solution are evaluated based on the phenotype
  - variation operators are applied to the genotype.
  - some of the solutions are removed from the population if the population size exceeds an upper limit.
- **Variation operators**
  - create new solutions with similar properties to existing solutions.
  - main search operator is recombination
  - mutation serves as background operator
- **Selection**
  - High-quality individuals are selected more often for reproduction.

Functionality

- **Generate set of different initial solutions**

- **Repeat until termination :**

  – Repeat within a population

    - Combine several solutions to form a new one s'

    - Create a random neighboring solution s' in the neighborhood $M(s)$ of s

  – Select only a fraction of the newly created solutions s'.

search space S

Initial solution 2

Different neighborhood in comparison to local search

Initial solution 1

Initial solution 3

new solution

mutation

245

How does a GA work?

```
pop = random_population();
while (not done)

    parents = {};
    while (|parents| < |pop|)
        x = tournament_winner(pop);
        parents = parents + x;
    end;                                        Selection
    offspring = {};


    while (|offspring| < |pop|)
        x = get_random_with_deletion(parents);
        y = get_random_with_deletion(parents);
        (x',y') = crossover(x,y);
        x" = mutation(x');
        insert(x",offspring);
        y" = mutation(y');
        insert(y",offspring);
    end;                                        Variation
    pop = offspring;

end;                                            Generation
```

246

Genetic algorithms

- Two or more solutions create offspring using a recombination operator

- Assumption:

    – Solutions have characteristic properties

    – Recombination has to identify relevant characteristics (building blocks) and combine those characteristics in an offspring.

    – Population necessary to ensure that different characteristica are available.

- Motivation: local optima for local search are no local optima for recombination-based search.

- However: Resulting neighborhood structure is non-intuitive

Design choices

- Choose a proper representation and corresponding search operators for the problem (ensure high locality for mutation and recombination)

- Design a mechanism that compares quality of different solutions.

- Set GA-specific parameters:

  - Population size (with increasing population size solution quality increases; running time (number of generations) is independent of pop size!).

  - High recombination probability

  - Low mutation probability (on allele level) to create **similar** solutions

Intensification

- Due to selection

- In each selection step, the average fitness of a population increases as only high-quality solutions are chosen for the mating pool

- Due to selection, the population converges after a number of generations

- Continuing recombination-based search after the population has converged (hopefully to the global optimum) makes no sense as diversity is minimal.

Diversification

- **Main source of diversification is the initial population**
- **Therefore, large population are used**
- **Recombination operators**
  - Can create new solutions
  - No active diversification. Recombination reduces diversity as the distances between offspring and parents are usually smaller than the distance between parents
  - Iterative application of crossover alone reduces the diversity of a population as
    - some solution properties can become extinct in the population (drift)) or
    - the decision variables converge to an average value (especially for continuous decision variables).

Diversification (2)

- Mutation has diversifying character
- Neighborhood structure does not remain constant during search, as mutation does not generate only neighboring solutions with small distance but can reach all solutions in the search space in only one mutation step.
- Mutation is iteratively applied to all l decision variables with probability p_m.
- On average, $p_m l$ alleles are mutated
- For large values of p_m, the mutation operator can mutate all l decision variables and, thus, reach all possible points in the solution space.
- The diversifying character of mutation increases with increasing p_m, and for large p_m, SGA behaves like random search.

Design Principles

1. High Locality
2. Bias

Design Guidelines

Vast majority of real-world optimization problems are

- neither deceptive nor difficult and

- have high locality (used metric is meaningful)

- Design of modern heuristics should not destroy the high locality of a problem.

- Local search operators must generate neighboring solutions

- Recombination operators must re-combine solutions

Biasing modern heuristics

- **If we know something about good solutions, we can seed such information into the modern heuristic**
  - Representation: incorporate construction heuristics, or use redundant encodings
  - Search operators can distinguish between good and bad solution features (building blocks)
  - We can also bias the fitness function, the initial solutions, the search strategy

Incorporating Construction Heuristics in Representations

- Early example: ordinal representation (Grefenstette et al. 1985) for the TSP.
- Encodes a tour (permutation of n integers) by a genotype x^g of length n, where $x^g_i \in \{1,\dots,n-i\}$ and $i \in \{0,\dots,n-1\}$. For constructing a phenotype, a predefined permutation x^s of n integers representing the n different cities is used. x^s can be problem-specific and, for example, consider edge weights. A phenotype (tour) is constructed from x^g by subsequently adding (starting with i=0) the x^g_ith element of x^s to the phenotype (which initially contains no elements) and removing the x^g_ith element of x^s. Problem-specific knowledge can be considered by choosing an appropriate x^s as genotypes define perturbations of x^s and using small integers for the x^g_i results in a bias of the resulting phenotypes towards x^s. For example, for $x^g_i=1$ ($i \in \{0,\dots,n-1\}$), the resulting phenotype is x^s.

Biased representation

- Number of genotypes exceeds number of genotypes

- Redundant representations

- Redundant representations are biased if some phenotypes are represented by a larger number of genotypes

- Biased representation: overrepresentation of good solutions, good solutions take larger share of search space

Biased search operators

- Search operators are biased if they generate or select certain solutions with higher probability

- Integration of structural features into new solutions

- Example: optimal TSP solutions do not cross
  - Operators should not generate crossing solutions because they have bad quality

Biased fitness function

- Fitness function is objective function as the heuristic sees it

- Improve fitness of solutions that share a desired feature

- Penalize solutions that do not share a certain feature

- Handle constraints: penalize solutions that violate certain constraints

Biased search strategy

- If problem is known to be unimodal, we can favor intensification and use less exploration

- If problem is known to be multimodal, local minima are more important and we must diversify

- Example: use different starting temperatures in Simulated Annealing approach, depending on multimodality of search space

Done