Chapter 5

Basic Algorithms for Allocation Problems

5.1 Heuristic Search

- Most of the allocation problems mentioned in chapter 4 are combinatorial problems.

- Therefore, they belong to the class of **discrete optimization problems**:

$$\varphi(x_1, x_2, \ldots, x_n) \rightarrow \min$$

$$with \quad \sum_{j=1}^{n} x_j = C\,; \qquad x_j \in N, j = 1, \ldots, n$$

- In this general form they are NP-hard.
- Thus, for many practical problems optimal solutions cannot be found in acceptable time.

Heuristic Search

- A heuristic search is a smart exploration of parts of the solution space.

- Concentration on promising areas

- No guarantee of optimality

- Compromise between **effectiveness** (how good is the solution found) and **efficiency** (how fast do we find the solution)

Local Search

`solution ← initial_solution`	initialization
`finished ← ` **`false`**	
`while not` `finished` **`do`**	main loop
`    new_solution ← modify(solution)`	search step
`    delta_phi←phi(new_solution)-phi(solution)`	evaluation
`    if` `delta_phi < 0`	improvements only
`        then` `solution ← new_solution`	acceptance
`    finished ← f(?)`	termination criterion
`    end while`	
`end`	

Problems:

- What are search steps like?
- What are elementary search steps?
- How can we find at least a local minimum?

Digression: Combinatorics

- Combinatorics is the study of collections of objects. Specifically, **counting** objects, **arrangement**, **derangement**, etc. along with their mathematical properties.
- Counting objects is important in order to analyze algorithms and to compute discrete probabilities.
- Originally, combinatorics was motivated by gambling: counting configurations is essential to elementary probability.
  - Example: How many arrangements of a deck of 52 cards are possible?

Permutations

- A **permutation** of a set of distinct objects is an ordered arrangement of these objects.
- An ordered arrangement of *r* elements of a set of *n* elements is called an ***r*-permutation**
- The number of *r* permutations of a set of *n* distinct elements is

$$P(n,r) = \Pi_{i=0}^{r-1}(n-i) = n(n-1)(n-2)\cdots(n-r+1)$$

- It follows that $P(n,r) = \dfrac{n!}{(n-r)!}$
- In particular P(n,n) = n!
- Note here that **the order is important**. It is necessary to distinguish when the order matters and when it does not.

Permutations: Example

- How many pairs of dance partners can be selected from a group of 12 women and 20 men?
  - The first woman can partner with any of the 20 men, the second with any of the remaining 19, etc.
  - To partner all 12 women, we have

$$P(20,12) = 20!/8! = 20 \times 19 \times 18 \times \ldots \times 10 \times 9$$

Combinations

- Whereas permutations consider order, **combinations** are used when order does not matter.
- **Definition**: A ***k*-combination** of elements of a set is an **unordered** selection of *k* elements from the set.
  - (A combination is simply a subset of cardinality *k*.)
- The number of *k*-combinations of a set of cardinality *n* with $0 \leq k \leq n$ is

$$C(n,k) = \binom{n}{k} = \frac{n!}{(n-k)!k!}$$

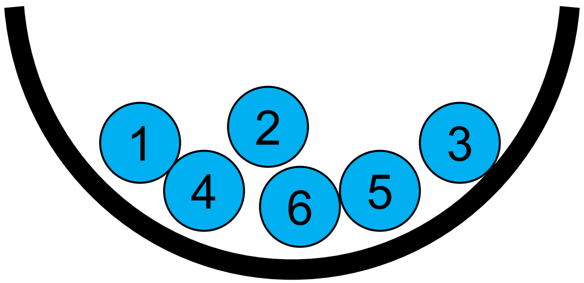
It is read '*n* choose *k*'.

- A useful fact about combinations is that they are symmetric:

$$\binom{n}{k} = \binom{n}{n-k}$$

Ball-in-urn experiment

- Given n balls in an urn or bowl. m times a ball is taken out.
- Question: How many different possibilities exist to take out a ball m times from n balls?



Differentiation:

1. Is the ball put back each time or not?
2. Does it make a difference in which order the balls are removed from the bowl or not?

Combinatorics: Example 2 out of 4

	With repetition	Without repetition
Order matters	(1,1),(1,2),(1,3),(1,4) (2,1),(2,2),(2,3),(2,4) (3,1),(3,2),(3,3),(3,4) (4,1),(4,2),(4,3),(4,4)	(1,2),(1,3),(1,4) (2,1), (2,3),(2,4) (3,1),(3,2), (3,4) (4,1),(4,2),(4,3)
Order does not matter	(1,1),(1,2),(1,3),(1,4) (2,2),(2,3),(2,4) (3,3),(3,4) (4,4)	(1,2),(1,3),(1,4) (2,3),(2,4) (3,4)

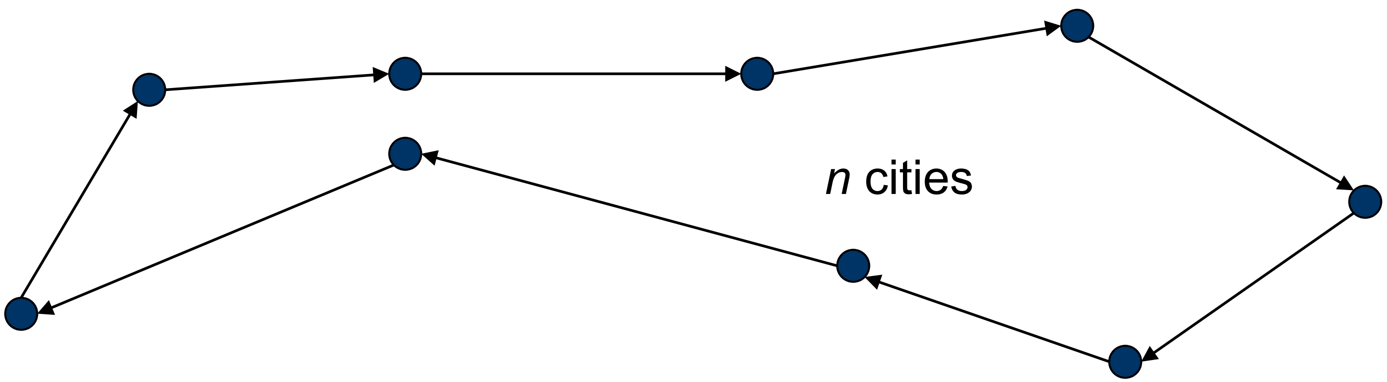
Combinatorics: Formulae

	With repetition	Without repetition
With considering order	"*k*-Sample" n^k	"*k*-Permutation" $P(n,k)=\binom{n}{k}\cdot k! = \frac{n!}{(n-k)!}$
Without considering order	"*k*-Selection" $C(n+k-1,n-1)=\binom{n+k-1}{n-1} = \frac{(n+k-1)!}{k!\,(n-1)!}$	"*k*-Combination" $C(n,k)=\binom{n}{k}=\frac{n!}{(n-k)!\,k!}$

5.2 Representation

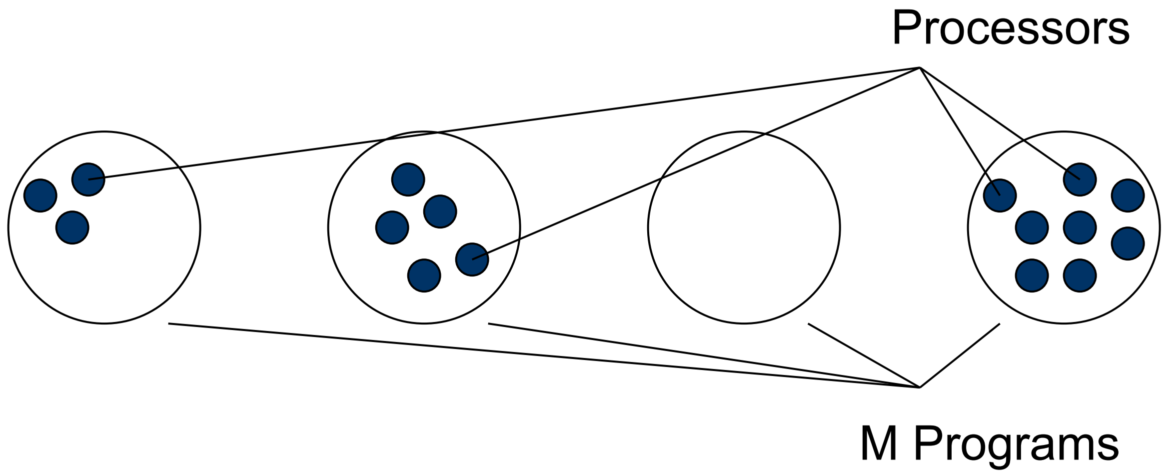
- For the general discussion of search problems, we represent a solution as a finite string from a finite alphabet.

Example: Traveling Salesman Problem



- Goal: Minimal round trip
- Coding: string of length n from the alphabet $\{1,\ldots,n\}$ without repetition (permutation)
  $x_i = k$: city k is visited at i-th place.
- Size of solution space (permutation): $n!$

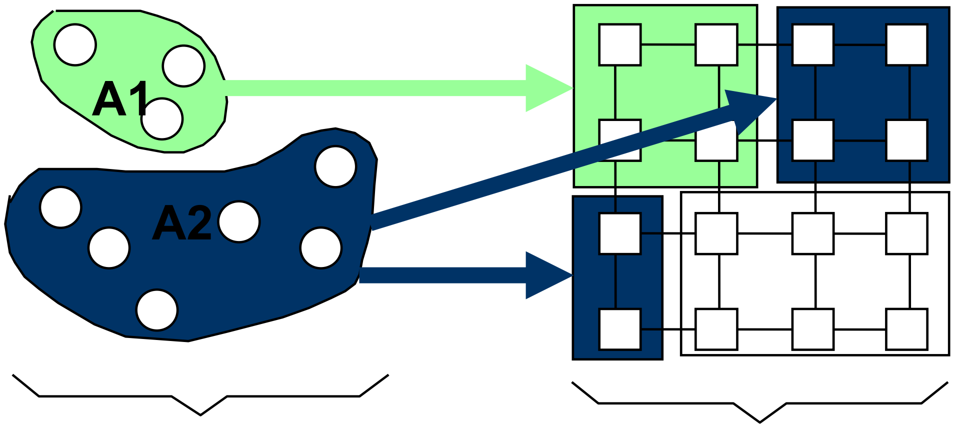
Quantitative Partitioning



- Goal: Partitioning of p processors into M programs so that the accumulated running time is minimized
- Coding: String of size M of alphabet $\{0,1,\ldots,p\}$ with sum of digits = p

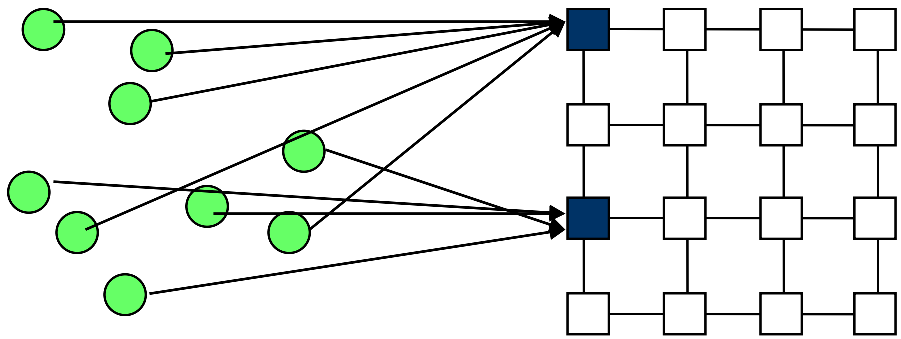
  $x_i = k$: program i is given k processors
- Size of solution space: $$\binom{p+M-1}{p} = \binom{p+M-1}{M-1}$$

Qualitative Partitioning



- Goal: Assignment of M programs to processor sets with minimal fragmentation and minimal interprocessor communication.
- Coding: string of size p of alphabet
  $\{0,1,\ldots,M\}$ with repetition
  $x_i = k$: processor i is occupied by program k
- Size of solution space: $(M+1)^p$

Contractive Allocation

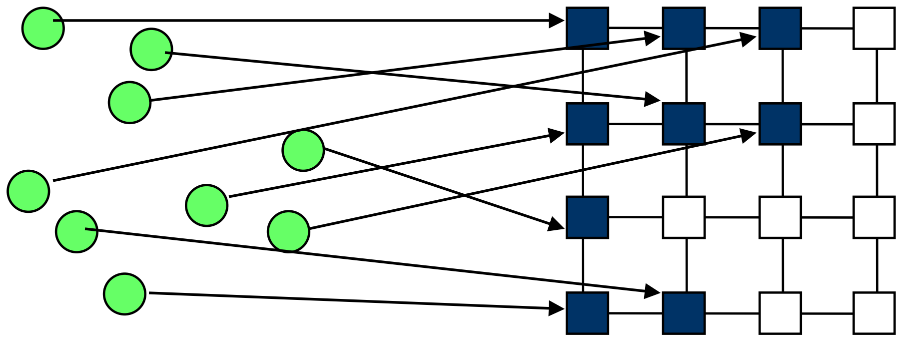


m threads

p processors

- Goal: Mapping of threads to processors with minimal interprocessor communication and balanced load.
- Coding: String of size m of alphabet
  $\{1,\ldots,p\}$ with repetition
  $x_i = k$: thread i is assigned to processor k
- Size of solution space: p^m

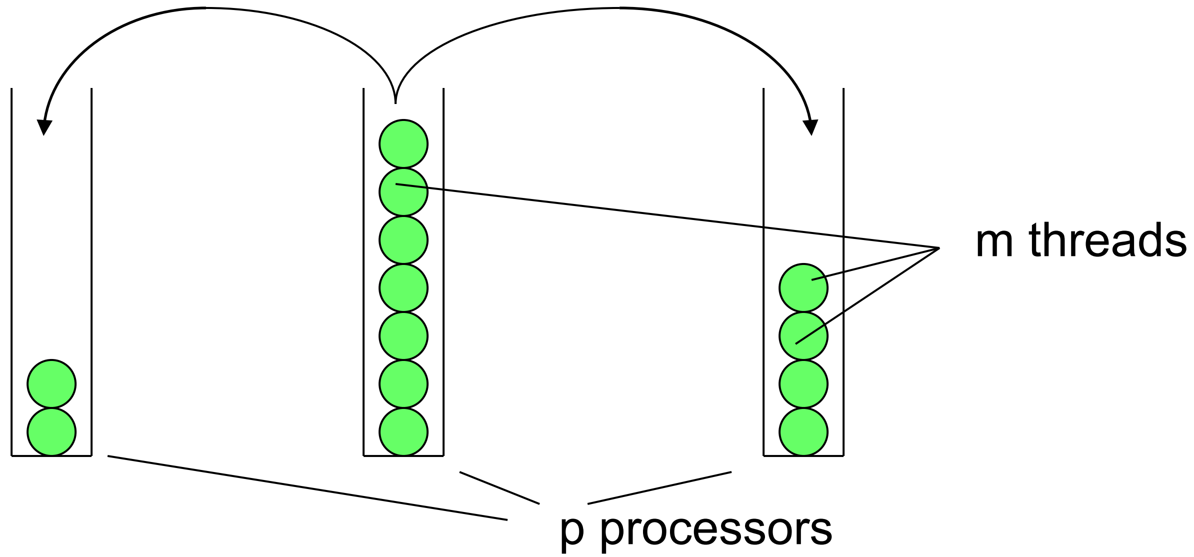
Injective Allocation



m threads p processors

- Goal: Injective mapping of *m threads to p processors with minimal* communication cost.
- Coding: String of size p of alphabet
  $\{0,1,\ldots,m\}$ without repetition
  $x_i = k$: processor i executes thread k
- Size of solution space: $\binom{p}{m} m!$

Load balancing



- Goal: Balanced distribution of m threads across p processors
- Coding: String of size *p* of alphabet
  $\{0,1,\ldots,m\}$ with sum of digits of m
  $x_i = k$: processor i receives a load of k threads

- Size of solution space: (same problem as slide 13) $$\binom{m+p-1}{m-1} = \binom{m+p-1}{p}$$

Intermediate result

- A solution is represented by a string of size n.
- The set of feasible solutions spans the solution space.
- Solutions are points in an n-dimensional solution space.
- Associated with each solution x is the value $\Phi(x)$ of the objective function to be optimized.
- Search algorithms scan the solutions space and try to find a solution with a very high (or small) value Φ.

Elementary search steps

- A search step is a (feasible) modification of the string representing a solution.
- Elementary search steps (examples):
  - Local value change $x_1, x_2, x_3, \ldots, x_i \pm \Delta, \ldots, x_{n-1}, x_n$
  - Exchange $x_1, x_2, x_3, \ldots, x_i, \ldots, x_k, \ldots, x_{n-1}, x_n$
  - Increment shift $x_1, x_2, x_3, \ldots, \overset{-1}{x_i}, \ldots, \overset{+1}{x_k}, \ldots, x_{n-1}, x_n$

- Problem: Between the positions and between the characters distances have to be defined.

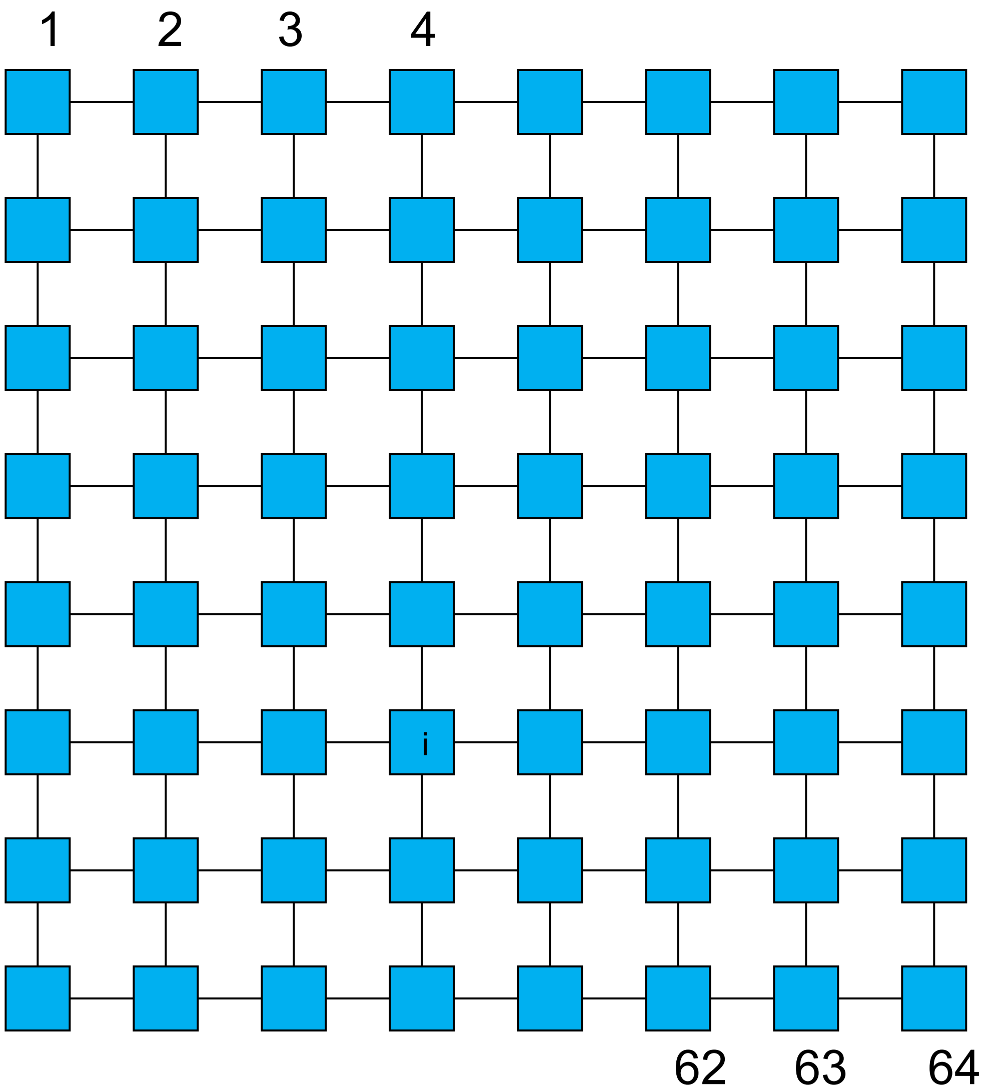
Neighborhood

- Depending on the problem and the algorithm used, elementary steps may be defined differently.
- The neighborhood of a point (a solution) denotes all points that can be reached with one elementary step.
- neighbors(x) := $\{x' \mid \exists$ elementary step $x \rightarrow x'\}$
- Usually, we have symmetry:

$$x \in \text{neighbors}(y) \Leftrightarrow y \in \text{neighbors}(x)$$

- Symmetry implies that search steps are reversible.

Example: Contractive allocation of *m* threads on a 8x8-mesh



e.g. $x_i = 44$:

thread *i* is allocated to processor 44

Elementary step:

local:

move to adjacent processor

$|\text{neighbors}(x)| = 4 \cdot m$

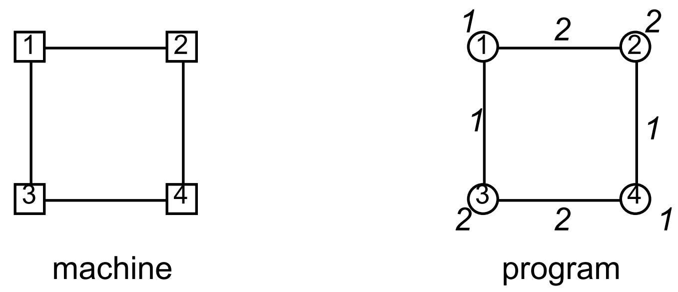
global:

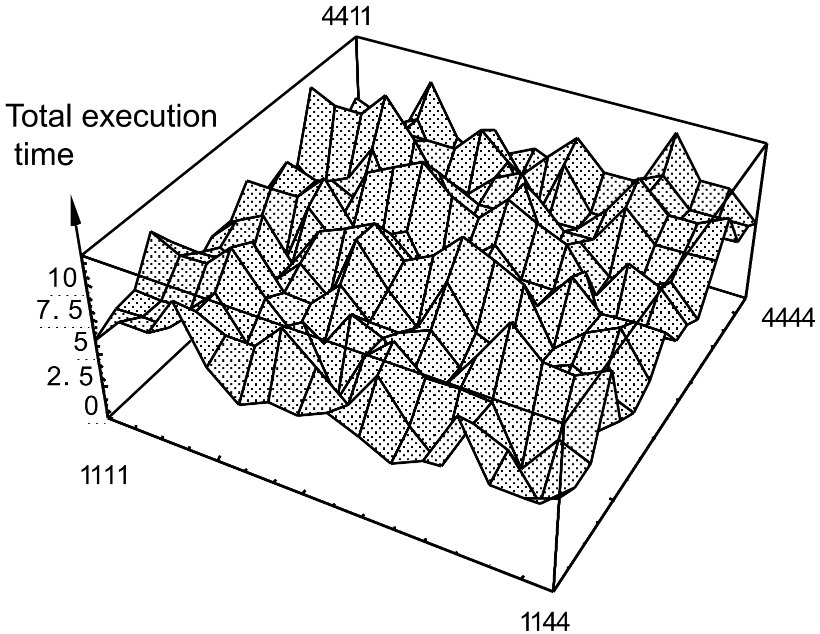
move to any place

$|\text{neighbors}(x)| = 63 \cdot m$

Example

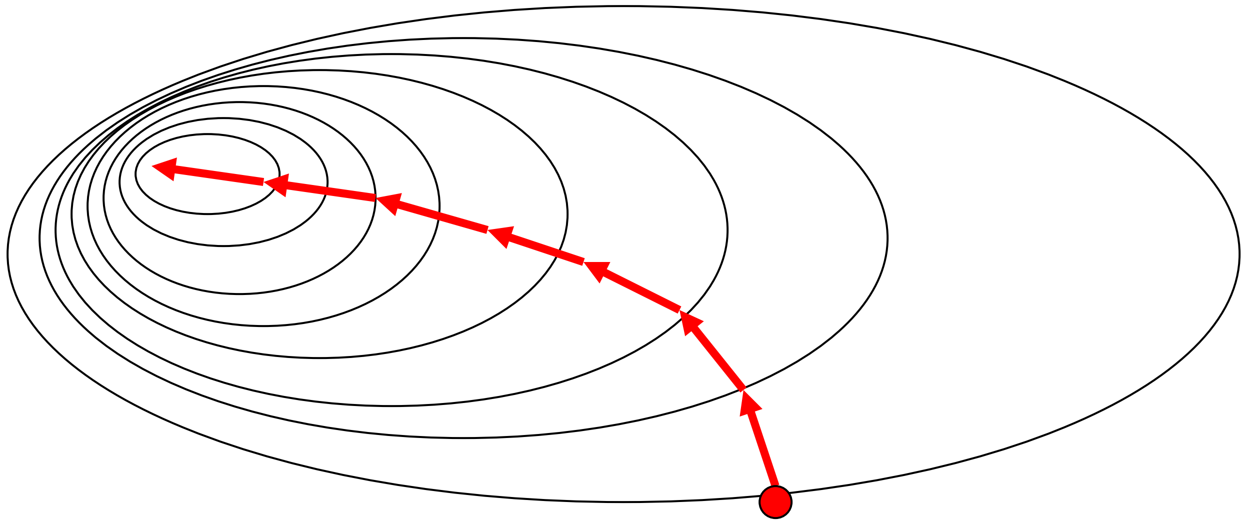
- A program consisting of 4 threads has to be assigned to a ring of 4 processors contractively.
- Objective function is the total execution time.
- Each solution is a 4-tuple (x_1, x_2, x_3, x_4) with $x_i=k$: thread *i* is assigned to processor *k*.





5.3 Gradient descent

- The gradient descent is an improved variant of the local search.
- Instead of performing an arbitrary elementary step, all possible elementary steps are evaluated (objective function) and that step with the largest gain is selected.
- We therefore proceed in the direction of the steepest slope (direction of gradient).



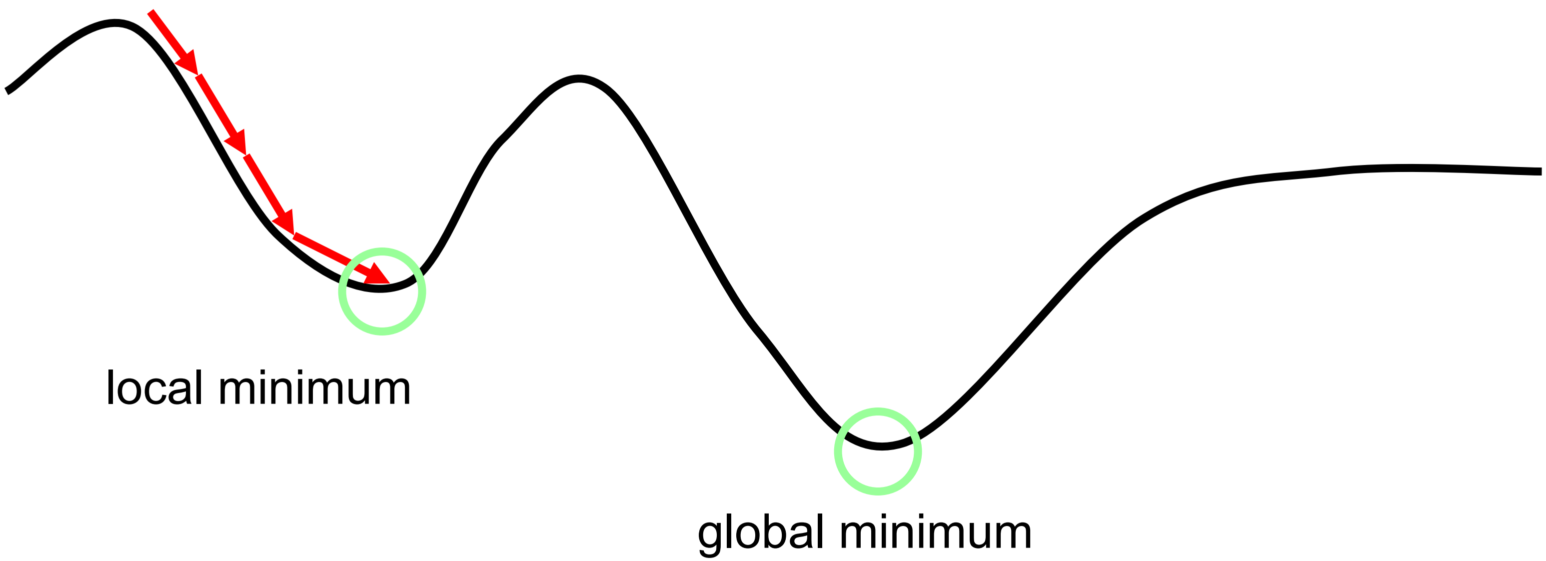
Gradient descent

Freie Universität Berlin

1	solution ← initial solution	initialization
2	finished ← **false**	
3	**while not** finished **do**	main loop
4	**for** all neighbors(solution) **do**	
5	calculate phi(neighbor)	test steps
6	**end** for	
7	new_solution←neighbor with min. phi(neighbor)	gradient descent
8	delta_phi←phi(new_solution)-phi(solution)	evaluation
9	**if** delta_phi < 0	
10	**then** solution ←new_solution	descent (improvement)
11	**else** finished ← **true**	local minimum
12	**end while**	
13	**end**	

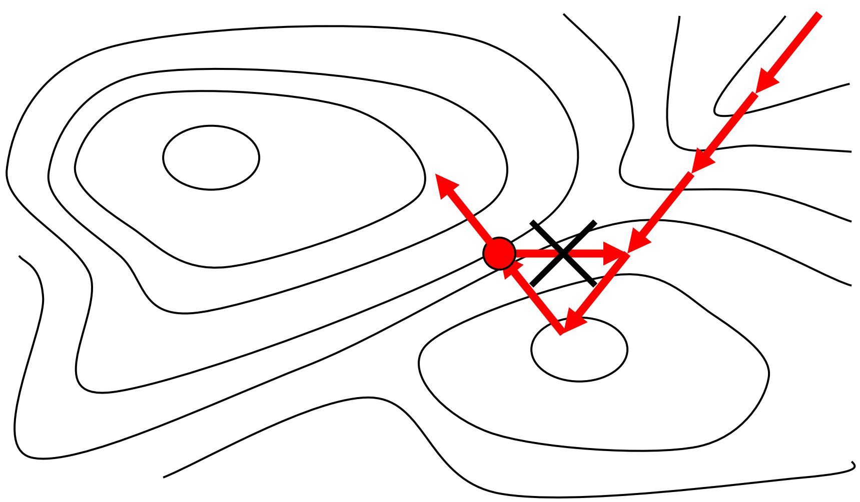
Local Minimum

- Local search algorithms exhibit the fundamental problem that they do find a local minimum but cannot escape it.
- Many heuristic approaches can be distinguished by the way they are able to leave local minima.



5.4 Taboo search

- Taboo search is another improved variant of local search.
- Occasionally also deteriorations are accepted.
- In contrast to the gradient descent it has a “memory”:
  - Best solution found so far
  - List of forbidden steps (taboo), e.g. to avoid cycles

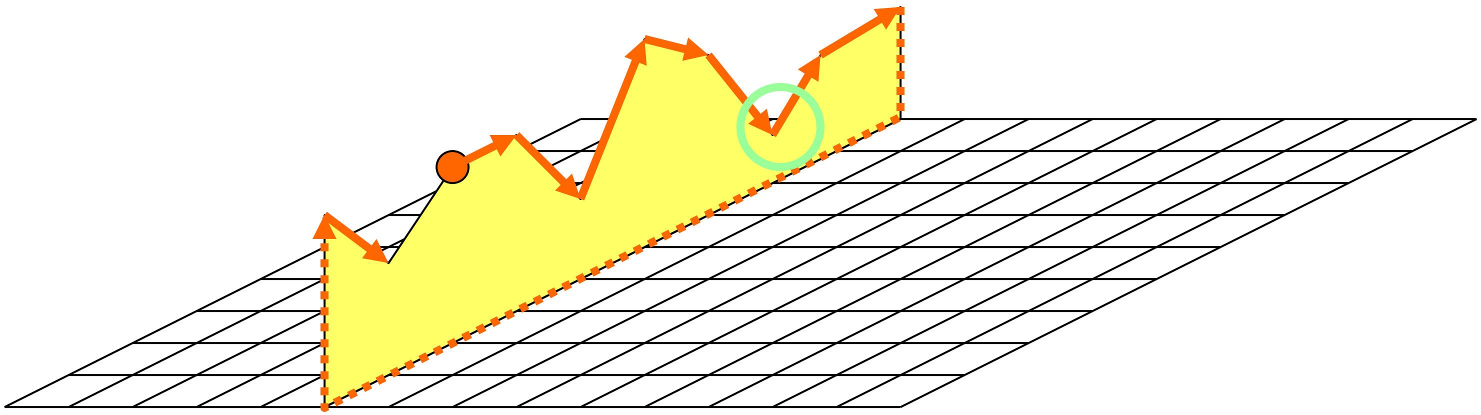


Taboo search (simplified)

solution ← initial solution	initialization
best_solution ← solution	
taboo_list ← {solution}	
finished ← **false**	
while not finished **do**	main loop
for all neighbors(solution) **do**	
calculate phi(neighbor)	test steps
end for	
solution←neighbor with min. phi(neighbor) and neighbor ∉ tabulist	calculate new solution
tabulist ← tabulist ∪ {solution}	update
if phi(solution) < phi(best_solution)	
then best_solution ← solution	store best solution
finished ← **f(?)**	termination criterion
end while	
end	

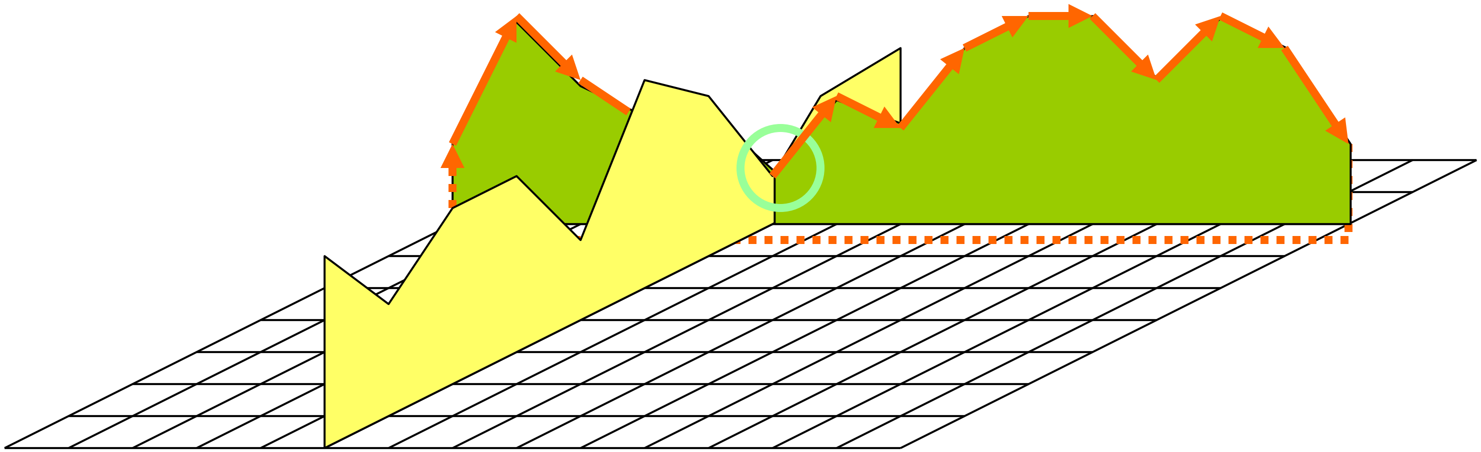
5.5 Probing paths

- Another idea to escape from local minima is to run a complete path (e.g. along some dimension) and only then accept the minimum of the path as the next solution.



Probing paths

- Starting at the minimum found we can then proceed with a new probing path along a new dimension.



Probing paths

1	solution ← initial solution	initialization
2	finished ← **false**	
3	**while not** finished **do**	main loop
4	select probing_path	
5	i ← 0	
6	step[0] ← solution	
7	**while not** (end_of_path) **do**	test steps
8	step[i] ← next(step[i-1])	
9	phi_s[i] ← phi(step(i))	store values of objective function
10	**end** while	
11	i ← minimum(phi_s)	index of minimum along path
12	new_solution ← step[i]	gradient descent
11	delta_phi←phi(new_solution)-phi(solution)	evaluation
12	**if** delta_phi < 0	
13	**then** solution ←new_solution	descent (improvement)
14	**else** finished ← **true**	local minimum
15	**end while**	

5.6 Simulated Annealing (SA)

- Idea 1: Accept steps „uphill" with some probability.
- Idea 2: At the beginning search in large areas and then gradually restrict the scope of the search.

- Approach: Prob(accept deterioration by $\Delta\varphi$) = $\exp(-\Delta\varphi/T)$

- T controls the "acceptance of deterioration" and is reduced stepwise e.g.:
  - logarithmic decrement: $T(i) := T_0 / \log(i)$, $i = 1,2,\ldots$
  - geometric decrement: $T(i) := T_0 / q^i$ $(q>1)$, $i = 1,2,\ldots$
  - linear decrement: $T(i) := T_0\,(1- i/m)$, $i = 0,1,2,\ldots,m$

- **Remark**: Idea goes back to computer simulation of cooling down procedures (annealing) of specific material (spin glasses), that are fluid at high temperature (high mobility of molecules) and by careful annealing achieve a homogeneous grid structure at minimum energy.

Simulated Annealing (Variant)

Also here it is rewarding to save the best solution found so far

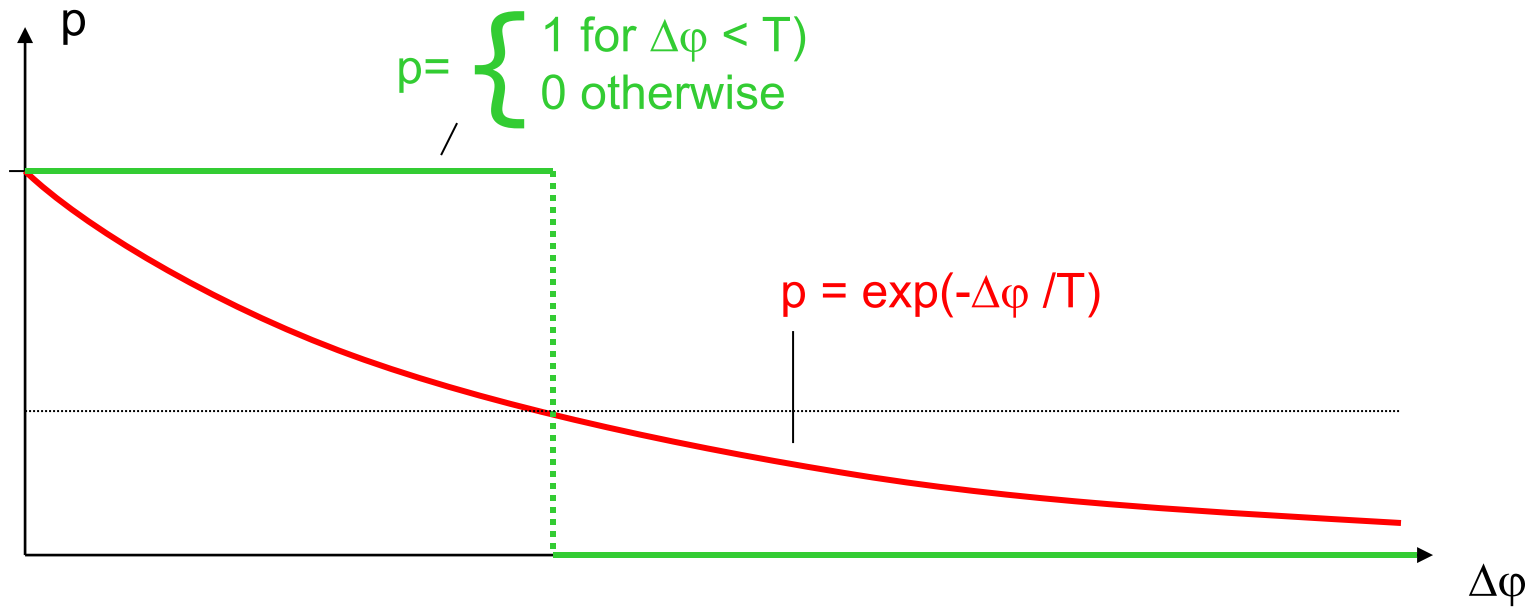
solution ← random initial solution	
T ← T0	initialize threshold
best_solution ← solution	
for i ← 0 **to** m **do**	m=number of stages
for j ← 1 **to** n **do**	n =number of steps per stage
new_solution ← modify(solution)	
delta_phi←phi(new_solution)-phi(solution)	
if phi(new_solution) < phi(best_solution)	
then best_solution ← new_solution	save new best solution
if delta_phi < 0	
then solution ← new_solution	improvement
else with prob exp(-delta_phi/T)	deterioration
solution ← new_solution	
end for	
T ← decrement(T)	decrement threshold
end for	

Threshold Accepting

- Experimental insight: acceptance can be controlled deterministically without loss of solution quality:

  Accept, if $\Delta\varphi < T$

- Results in a variant of Simulated Annealing.



Threshold Accepting

solution ← random initial solution	
T ← T0	initialize threshold
best_solution ← solution	
for i ← 0 **to** m **do**	m=number of stages
for j ← 1 **to** n **do**	n =number of steps per stage
new_solution ← modify(solution)	
delta_phi←phi(new_solution)- phi(solution)	
if phi(new_solution)< phi(best_solution)	
then best_solution ← new_solution	save best solution
if delta_phi < T	
then solution ← new_solution	acceptance
end for	
T ← decrement(T)	decrement threshold
end for	

Deluge Algorithm

Further variant: Instead of relative comparing with previous solution, compare with best solution

`solution ← random initial solution`	
`T ← T0`	initialize threshold
`best_solution ← solution`	
for `i ← 0` **to** `m` **do**	m=number of stages
for `j ← 1` **to** `n` **do**	n =number of steps per stage
`new_solution ← modify(solution)`	
`delta_phi←phi(new_solution)-phi(solution)`	
if `phi(new_solution)< phi(best_solution)`	
then `best_solution ← new_solution`	save best solution
if `phi(new_solution)<phi(best_solution)+T`	maximally by T worse than best solution
then `solution ← new_solution`	acceptance
else `solution ← best_solution`	variant: backtracking
end for	
`T ← decrement(T)`	reduce threshold
end for	

5.7 Iteration

- The solution found by a heuristic search algorithm depends on the initial solution.
- Starting with another initial value we may find another, better final solution.
- Thus: Iteration across different initial solutions.

`best_solution ← initial solution`	initialization
for `i ← 1` **to** `n` **do**	iteration
`solution ← new_initial solution`	new initialization
`{heuristic Search}`	delivers loc. minimum (solution)
if `phi(best_solution) > phi(solution)`	
then `best_solution ← solution`	improvement
end `for`	
end	

5.8 Parallelism

- Since the particular iterations are independent of each other, they can be executed in parallel.

- Two variants:
  - Variant A
    - Disjoint partitioning of solution space
    - Heuristic search in each partition (subspace) in parallel
    - Calculating minimum after termination of all search activities
  - Variant B
    - No partitioning
    - All „minimum searcher“ search in the whole area
    - Calculating minimum after termination of all search activities

5.9 Combination of solutions

- Evolutionary algorithms:
- Idea: If we proceed with many solutions simultaneously, why not combine their „beneficial“ features.
- Reverting to Darwin's evolution strategy: crossover, mutation, „survival of the fittest“

1	`solution_set ← initial solution_set`	initialization
2	`finished ←` **`false`**	
3	**`while not`** `finished` **`do`**	main loop
4	`combination_set ← select(solution_set)`	survival of the fittest
5	`new_solution_set← recombine(combination_set)`	cross over, mutation
15	`calculate phi(new_solution_set)`	evaluation
7	`finished ← f(?)`	termination criterion ?
8	**`end`**	

Genetic Algorithms (GA)

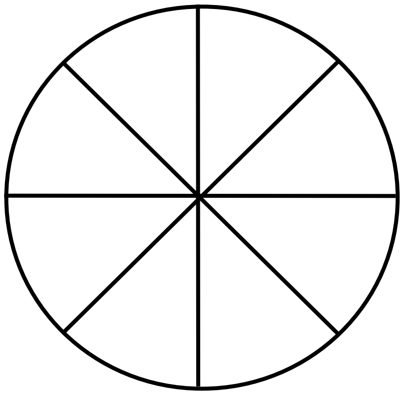
- Problem: How to find suitable operations?
  - How many solutions should be involved?
    - one: mutation
    - two: crossover
    - n: n-fold crossover
  - How do we combine or modify?
    - Permutation of strings
    - Exchange of substrings
    - Blend of substrings
- **Genetic Algorithms** are a subclass of Evolutionary Algorithms.
- They require a representation of solutions as binary strings of fixed length.
- They use two operators only :
  - binary mutation: flipping individual bits in solution string
  - binary crossover: combination of solution strings

- To explain the way GA work we need some auxiliary functions:
  - **random_select(set):**
    selects an element of a *n*-element set with equal probability ($p=1/n$).
  - **p_select(p, set):**
    selects an element of a n-element set according to a given discrete probability density function p_i ($i = 1,..,n$).
  - **head(x,j)** and **tail(x,j)**

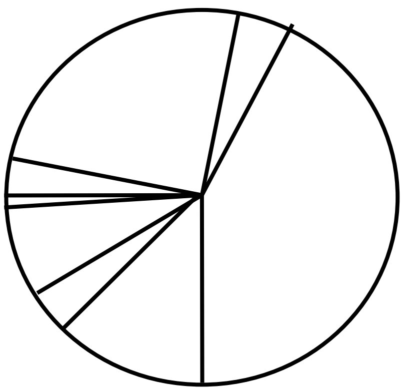
    $$\underbrace{x_1, x_2, x_3, \ldots, x_j}_{\text{head}(x,j)}, \underbrace{x_{j+1}, \ldots\ldots, x_n}_{\text{tail}(x,j)}$$
  - **mutation(x,j)**
    Bit *j* in bit string *x* is inverted.

Comparison of the two selection procedures

- The two selection procedures can be regarded as turning the roulette wheel with the elements as sectors.
- With random_select all sectors are of the same size, with p_select the probabilities are proportional to the sector angles.



random_select(M)



p_select(p,M)

Genetic Algorithms: Example

solution_set ← choose initial solution_set of size n	initialization (maximization problem)
while not finished **do**	main loop (generations)
for i ←1 **to** n **do**	all solution
sp[i] ←ϕ[i]/Σϕ[i]	selection probability sp
mating ← ∅	
for i ←1 **to** n **do**	
s ←p_select(sp,solution_set)	random selection according to sp
mating_pool ← mating_pool ∪{s}	
end for	
mating_pairs ← ∅	
While mating_pool ≠ ∅ **do**	random mating
s ← random_select[mating_pool]	
mating_pool ← mating_pool -{s}	
t ← random_select [mating_pool]	
mating_pool ← mating_pool-{t}	
mating_pairs ←mating_pairs ∪{(s,t)}	
end while	

Genetic Algorithms: Example (continued)

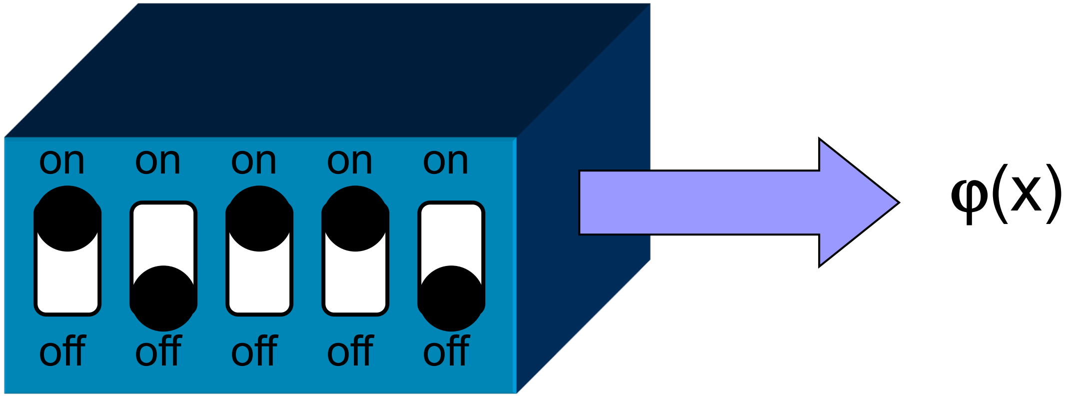
new_solution_set ← ∅	
for all (x,y) ∈ mating_pairs **do**	crossover
j ← random_select({1,2,..,k})	substring selection
new_solution1 ← head(x,j)+tail(y,j)	build generation of offsprings by
new_solution2 ← head(y,j)+tail(x,j)	crosswise exchange of substrings
new_solution_set ← new_solution_set ∪ {new_solution1,new_solution2}	parents are replaced by offsprings
end for	
for all s ∈ new_solution_set **do**	
with probability mp **do**	modification probability mp
j ← random_select({1,2,..,k})	random feature selection
modify(s,j)	feature j is modified
end with	
end for	
solution_set ← new_solution_set	
end while	
end GA	

Example

$\varphi(n) = n^2 \to$ max ; $\quad n \in \{0,1,2,...,31\}$

Representation as a GA problem:

$$n = \sum_{i=0}^{4} x_i 2^i \qquad x_i \in \{0,1\}$$



Example (continued)

1. Iteration

No.	Solution set	x	φ(x)	Selection probabili-ty	Sele ction	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)
1.	0 1 1 0 1	13	169	0.15	1	0 1 | 1 0 1	2	2	0 1 0 0 0	8	64
2.	1 1 0 0 0	24	576	0.51	2	1 1 | 0 0 0	1	2	1 1 1 0 1	29	841
3.	0 0 1 0 0	4	16	0.02	0	1 1 0 0 | 0	4	4	1 1 0 0 1	25	625
4.	1 0 0 1 1	19	361	0.32	1	1 0 0 1 | 1	3	4	1 0 0 1 0	18	324
S			1122								1854
Max			576								841

Example (continued)

2. Iteration

No.	Solution set	x	φ(x)	Selection probabili-ty	Sele ction	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)
1.	0 1 0 0 0	8	64	0.03	0	1 1 1 | 0 1	3	3	1 1 1 0 1	29	841
2.	1 1 1 0 1	29	841	0.45	2	1 1 | 1 0 1	4	2	1 1 0 1 0	26	676
3.	1 1 0 0 1	25	625	0.34	1	1 1 0 | 0 1	1	3	1 1 0 0 1	25	625
4.	1 0 0 1 0	18	324	0.18	1	1 0 | 0 1 0	2	2	1 0 1 0 1	21	441
S			1854								2583
Max			841								841

Example (continued)

3. Iteration

No.	Solution set	x	φ(x)	Selection probability	Selection	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)
1.	1 1 1 0 1	29	841	0.33	2	1 | 1 1 0 1	4	1	1 1 1 0 1	29	841
2.	1 1 0 1 0	26	676	0.26	1	1 1 0 | 1 0	3	3	1 1 0 0 1	25	625
3.	1 1 0 0 1	25	625	0.24	1	1 1 0 | 0 1	2	3	1 1 0 1 0	26	676
4.	1 0 1 0 1	21	441	0.17	0	1 | 1 1 0 1	1	1	1 1 1 0 1	29	841
S			2583								2983
Max			841								841

Example (continued)

Freie Universität Berlin

No.	Solution set	x	φ(x)	Selection probabili-ty	Sele ction	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)

4. Iteration

1.	1 1 1 0 1	29	841	0.28	2	1 | 1 1 0 1	2	1	1 1 1 0 1	29	841
2.	1 1 0 0 1	25	625	0.21	0	1 | 1 1 0 1	1	1	1 1 1 0 1	29	841
3.	1 1 0 1 0	26	676	0.23	1	1 1 0 | 1 0	3	3	1 1 0 0 1	25	625
4.	1 1 1 0 1	29	841	0.28	1	1 1 1 | 0 1	4	3	1 1 1 1 0	30	900
S			2983								3207
Max			841								900

Example (continued)

5. Iteration

No.	Solution set	x	φ(x)	Selection probabili-ty	Sele ction	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)
1.	1 1 1 0 1	29	841	0.26	1	1 1 1 0 | 1	3	4	1 1 1 0 0	28	784
2.	1 1 1 0 1	29	841	0.26	1	1 1 1 | 0 1	4	3	1 1 1 1 0	30	900
3.	1 1 0 0 1	25	625	0.20	0	1 1 1 1 | 0	1	4	1 1 1 1 1	31	961
4.	1 1 1 1 0	30	900	0.28	2	1 1 1 | 1 0	2	3	1 1 1 0 1	29	841
S			3207								3486
Max			900								961

Example (continued)

6. Iteration

No.	Solution set	x	φ(x)	Selection probability	Selection	Mating pool after selection	No of mate (rand.)	Substring selection (random)	New solution set	x	φ(x)
1.	1 1 1 0 0	28	784	0.22	0	1 1 | 1 1 1	3	2	1 1 1 1 1	31	961
2.	1 1 1 1 0	30	900	0.26	1	1 1 1 1 | 0	4	4	1 1 1 1 1	31	961
3.	1 1 1 1 1	31	961	0.28	2	1 1 | 1 1 1	1	2	1 1 1 1 1	31	961
4.	1 1 1 0 1	29	841	0.24	1	1 1 1 0 | 1	2	4	1 1 1 0 0	28	784
S			3486								3667
Max			961								961

Summary of Techniques

- Proceeding along the most promising direction (gradient descent)
- Memory, e.g., prevention of cycles (Taboo search)
- Evaluation of sequence of steps instead of single steps (probing paths)
- Acceptance of deterioration (Simulated Annealing)
- Backtracking
- Iteration with new initial solution
- Search in parallel
- Combination of solutions (Genetic Algorithms)

Further references

- Dueck,G.; Scheuer,T.; Wallmeier,H.-M.: *Toleranzschwelle und Sintflut: neue Ideen zur Optimierung.* Spektrum der Wissenschaft (März 1993) S.42-51 (in German)
- Goldberg,D.E.: *Genetic Algorithms.* Addison Wesley (1989)
- Kirkpatrick,S.; Swendsen,R.H.: *Statistical Mechanics and Disordered Systems.* CACM 28 (1985), S. 363-373
- Rayward-Smith, V. J. (Editor): *Modern Heuristic Search Methods*, John Wiley, 1996
- Pham, D.; Karaboga, D.; *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*, Springer, Berlin, 2000
- Michiels,W.; Aarts, E.; Korst,J.: *Theoretical Aspects of Local Search,* Springer, 2006