Knowledge-based systems in Bioinformatics, 1MB602

Lecture 9: Genetic algorithms and Genetic programming

Lecture overview

- Principles of genetic algorithms
- Multi-objective GAs
- Application guidelines
- Bioinformatics applications

- Genetic programming
- Bioinformatics applications

Optimization

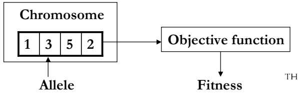
- In real-world optimization the search space is often HUGE (or infinite)
  - No option to examine all possible solutions to a problem
- One solution: local search methods or iterative improvement strategies
  - Requires some ‘smart’ strategy to avoid local optima
  - Or the need to run the selected algorithm multiple times
- A different strategy: Examine multiple solutions simultaneously and update them according to their values with respect to the other solutions
  - Genetic algorithms

Genetic algorithms

- Idea: use principles of evolution to discover better solutions to a problem given a random starting set of solutions
  - Reproduction
  - Selection
  - Crossover
  - Mutation
- Operators act on chromosomes (solutions) in the population (set of solutions) to yield a set of new solutions (next generation)
- Iteratively apply the operators to the population moving the algorithm from one generation to the next

Chromosomes

- GAs are general and flexible algorithm that can be used on many different problems
  - Accomplished through the use of chromosomes and objective functions
- Chromosome
  - Genetic representation of a single solution to the given problem
  - Usually a vector in numeric or binary format
- The performance (fitness) of a chromosome’s possibility of solving the problem is measured using an objective function



Chromosome

1 3 5 2

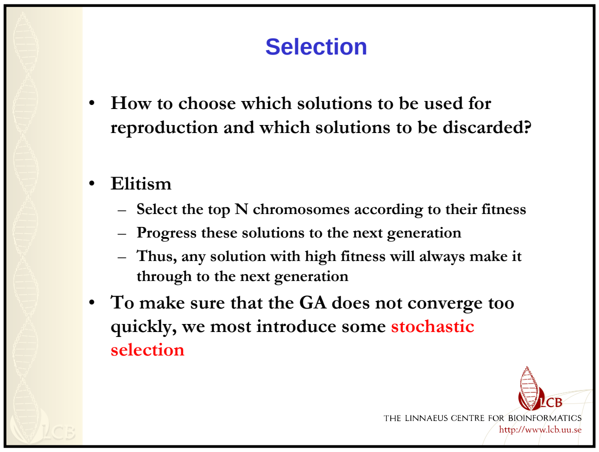
Objective function

Allele

Fitness

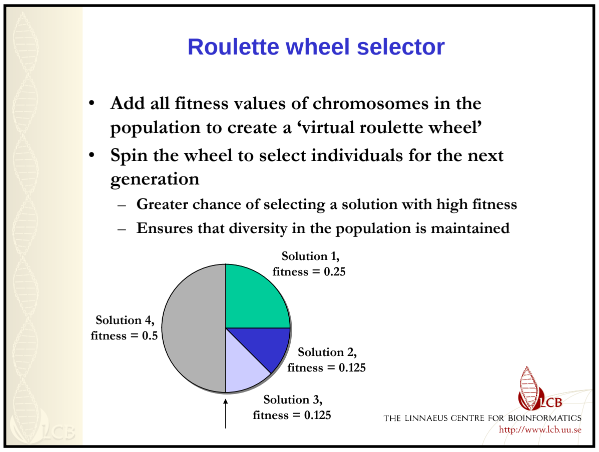
Operators

- The genetic operators that act on the population of solutions must drive the population to find new, and better solutions
  - Selection
  - Mutation
  - Crossover
- What makes the new generation?
  - Select a number of individuals from the current population
  - Combine them using crossovers to form new individuals
  - Possibly mutate some of the alleles in the individuals
  - Repeat the procedure until the next generation is filled



Selection

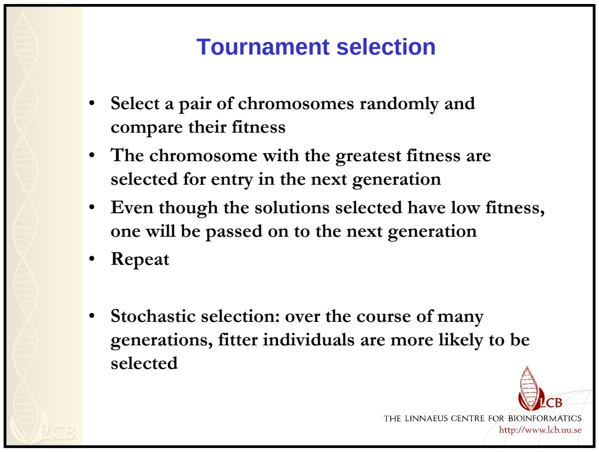
- How to choose which solutions to be used for reproduction and which solutions to be discarded?
- Elitism
  - Select the top N chromosomes according to their fitness
  - Progress these solutions to the next generation
  - Thus, any solution with high fitness will always make it through to the next generation
- To make sure that the GA does not converge too quickly, we most introduce some stochastic selection



Roulette wheel selector

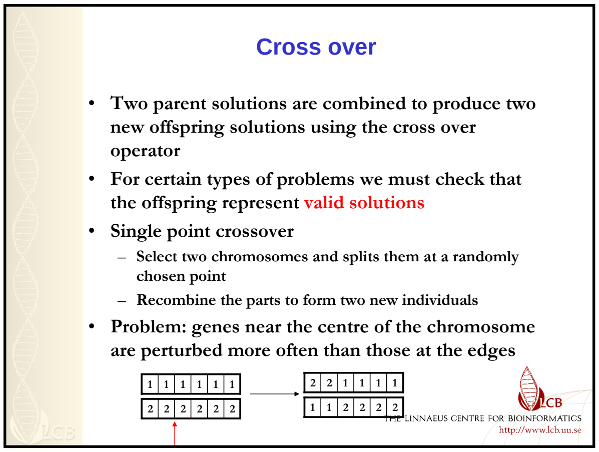
- Add all fitness values of chromosomes in the population to create a 'virtual roulette wheel'
- Spin the wheel to select individuals for the next generation
  - Greater chance of selecting a solution with high fitness
  - Ensures that diversity in the population is maintained

Solution 1,
fitness = 0.25
Solution 4,
fitness = 0.5
Solution 2,
fitness = 0.125
Solution 3,
fitness = 0.125



Tournament selection

- Select a pair of chromosomes randomly and compare their fitness
- The chromosome with the greatest fitness are selected for entry in the next generation
- Even though the solutions selected have low fitness, one will be passed on to the next generation
- Repeat
- Stochastic selection: over the course of many generations, fitter individuals are more likely to be selected

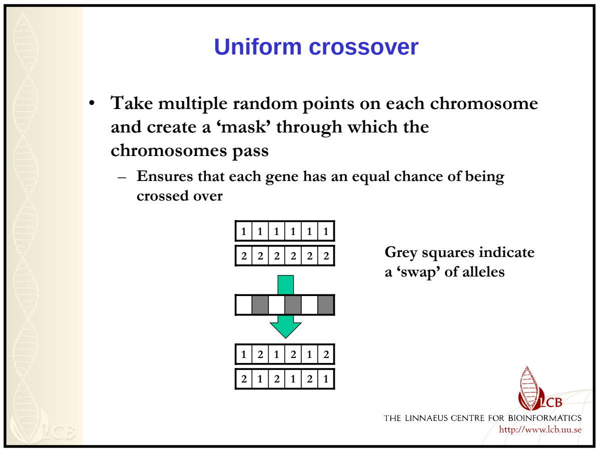


Cross over

- Two parent solutions are combined to produce two new offspring solutions using the cross over operator
- For certain types of problems we must check that the offspring represent valid solutions
- Single point crossover
  - Select two chromosomes and splits them at a randomly chosen point
  - Recombine the parts to form two new individuals
- Problem: genes near the centre of the chromosome are perturbed more often than those at the edges

1	1	1	1	1	1
2	2	2	2	2	2

2	2	1	1	1	1
1	1	2	2	2	2



Uniform crossover

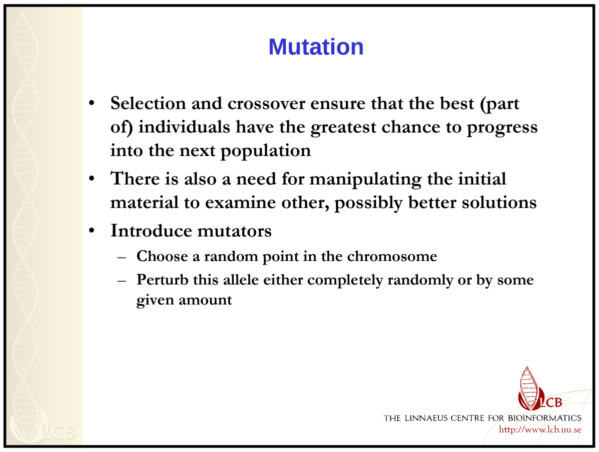
- Take multiple random points on each chromosome and create a 'mask' through which the chromosomes pass
  - Ensures that each gene has an equal chance of being crossed over

1	1	1	1	1	1
2	2	2	2	2	2

□	■	□	■	□	■

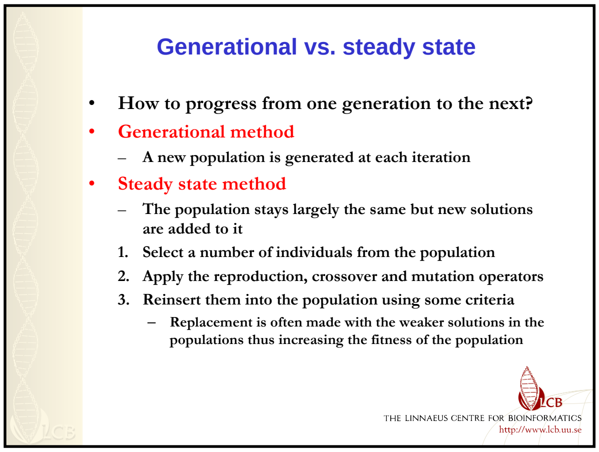
1	2	1	2	1	2
2	1	2	1	2	1

Grey squares indicate a 'swap' of alleles



Mutation

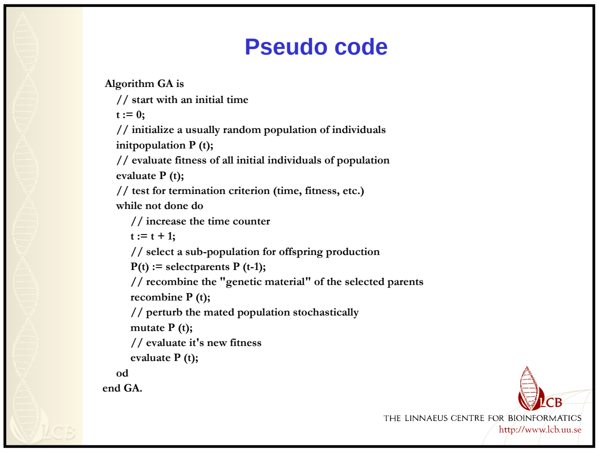
- Selection and crossover ensure that the best (part of) individuals have the greatest chance to progress into the next population
- There is also a need for manipulating the initial material to examine other, possibly better solutions
- Introduce mutators
  - Choose a random point in the chromosome
  - Perturb this allele either completely randomly or by some given amount



Generational vs. steady state

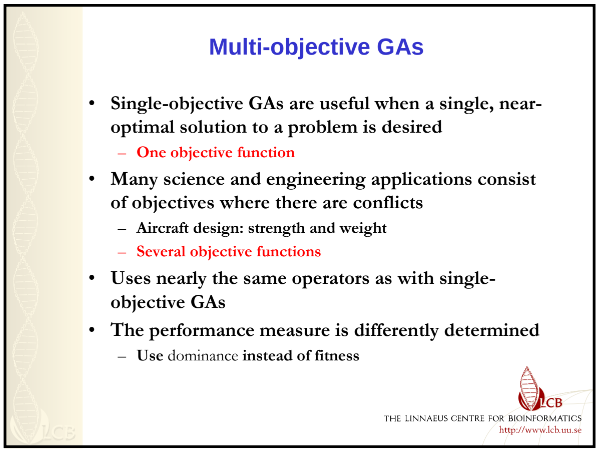
- How to progress from one generation to the next?
- Generational method
  - A new population is generated at each iteration
- Steady state method
  - The population stays largely the same but new solutions are added to it

1. Select a number of individuals from the population
2. Apply the reproduction, crossover and mutation operators
3. Reinsert them into the population using some criteria
   - Replacement is often made with the weaker solutions in the populations thus increasing the fitness of the population



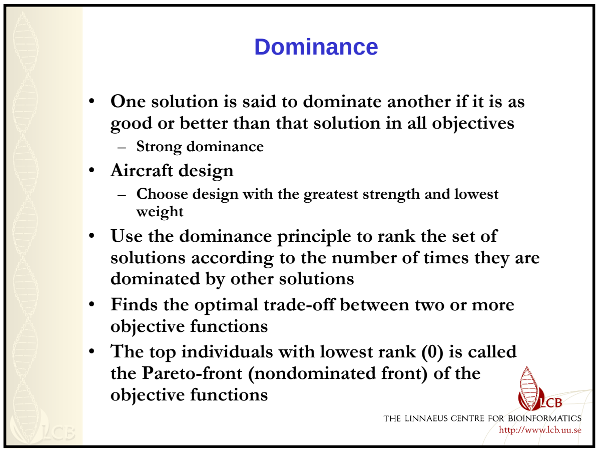
Pseudo code

```
Algorithm GA is
   // start with an initial time
   t := 0;
   // initialize a usually random population of individuals
   initpopulation P (t);
   // evaluate fitness of all initial individuals of population
   evaluate P (t);
   // test for termination criterion (time, fitness, etc.)
   while not done do
       // increase the time counter
       t := t + 1;
       // select a sub-population for offspring production
       P(t) := selectparents P (t-1);
       // recombine the "genetic material" of the selected parents
       recombine P (t);
       // perturb the mated population stochastically
       mutate P (t);
       // evaluate it's new fitness
       evaluate P (t);
   od
end GA.
```



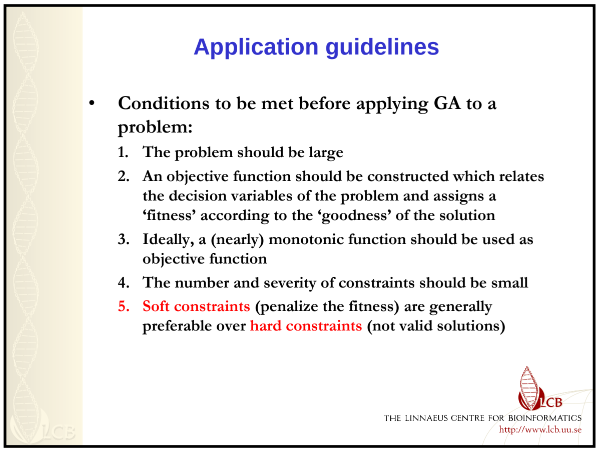
Multi-objective GAs

- Single-objective GAs are useful when a single, near-optimal solution to a problem is desired
  - One objective function
- Many science and engineering applications consist of objectives where there are conflicts
  - Aircraft design: strength and weight
  - Several objective functions
- Uses nearly the same operators as with single-objective GAs
- The performance measure is differently determined
  - Use dominance instead of fitness



Dominance

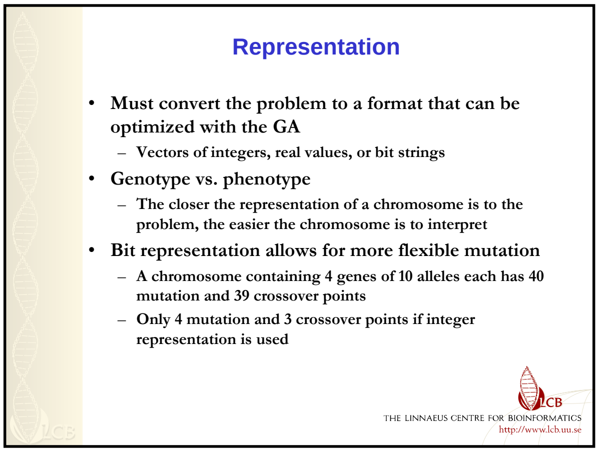
- One solution is said to dominate another if it is as good or better than that solution in all objectives
  - Strong dominance
- Aircraft design
  - Choose design with the greatest strength and lowest weight
- Use the dominance principle to rank the set of solutions according to the number of times they are dominated by other solutions
- Finds the optimal trade-off between two or more objective functions
- The top individuals with lowest rank (0) is called the Pareto-front (nondominated front) of the objective functions



Application guidelines

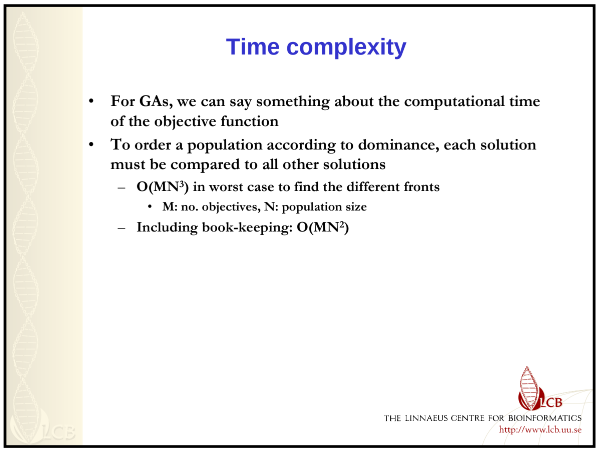
- Conditions to be met before applying GA to a problem:

1. The problem should be large
2. An objective function should be constructed which relates the decision variables of the problem and assigns a 'fitness' according to the 'goodness' of the solution
3. Ideally, a (nearly) monotonic function should be used as objective function
4. The number and severity of constraints should be small
5. Soft constraints (penalize the fitness) are generally preferable over hard constraints (not valid solutions)



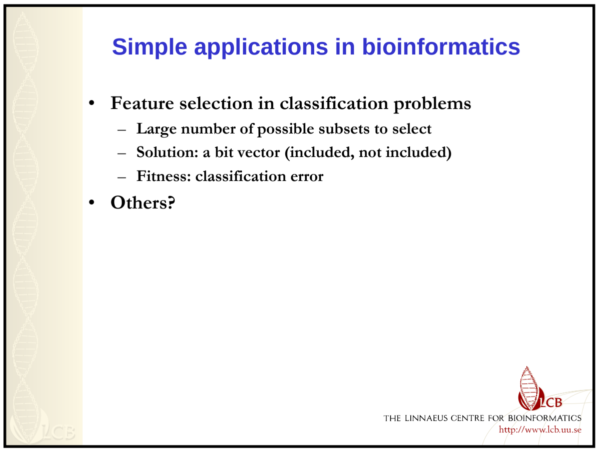
Representation

- Must convert the problem to a format that can be optimized with the GA
  - Vectors of integers, real values, or bit strings
- Genotype vs. phenotype
  - The closer the representation of a chromosome is to the problem, the easier the chromosome is to interpret
- Bit representation allows for more flexible mutation
  - A chromosome containing 4 genes of 10 alleles each has 40 mutation and 39 crossover points
  - Only 4 mutation and 3 crossover points if integer representation is used



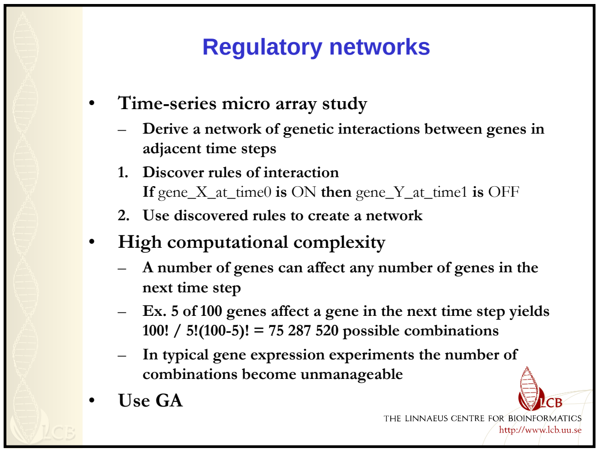
Time complexity

- For GAs, we can say something about the computational time of the objective function
- To order a population according to dominance, each solution must be compared to all other solutions
  - $O(MN^3)$ in worst case to find the different fronts
    - M: no. objectives, N: population size
  - Including book-keeping: $O(MN^2)$



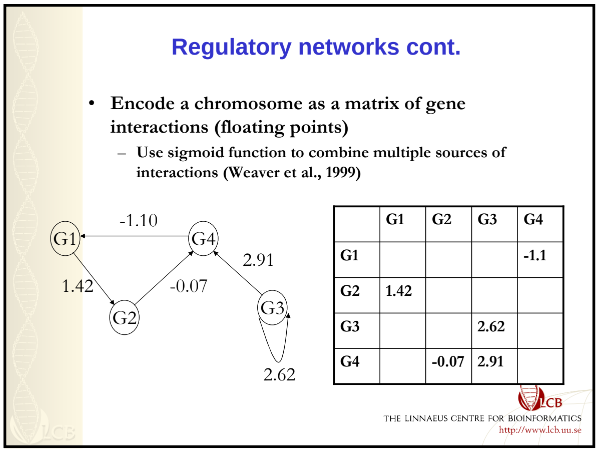
Simple applications in bioinformatics

- Feature selection in classification problems
  - Large number of possible subsets to select
  - Solution: a bit vector (included, not included)
  - Fitness: classification error
- Others?



Regulatory networks

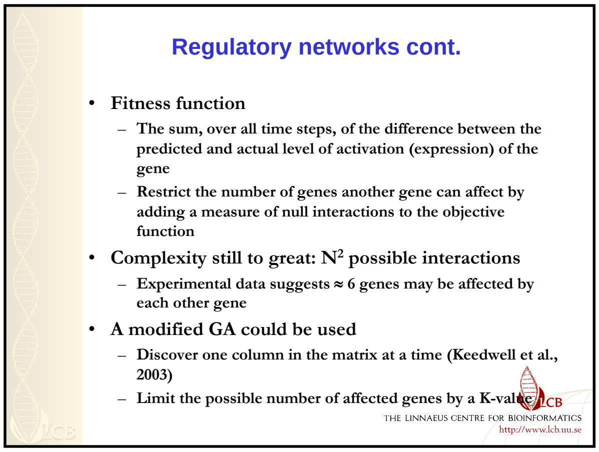
- Time-series micro array study
  - Derive a network of genetic interactions between genes in adjacent time steps
  1. Discover rules of interaction
     If gene_X_at_time0 is ON then gene_Y_at_time1 is OFF
  2. Use discovered rules to create a network
- High computational complexity
  - A number of genes can affect any number of genes in the next time step
  - Ex. 5 of 100 genes affect a gene in the next time step yields 100! / 5!(100-5)! = 75 287 520 possible combinations
  - In typical gene expression experiments the number of combinations become unmanageable
- Use GA



Regulatory networks cont.

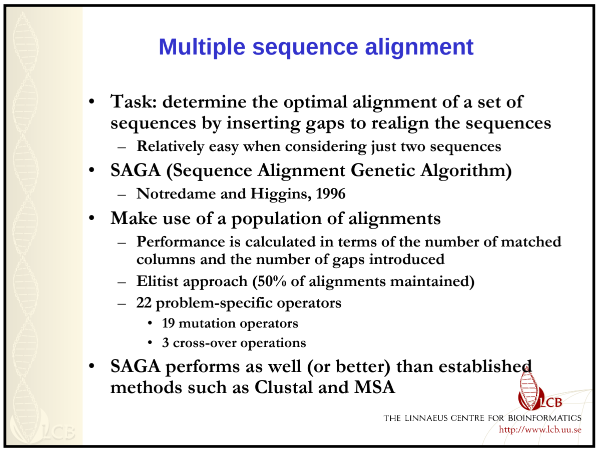
- Encode a chromosome as a matrix of gene interactions (floating points)
  - Use sigmoid function to combine multiple sources of interactions (Weaver et al., 1999)

	G1	G2	G3	G4
G1				-1.1
G2	1.42			
G3			2.62	
G4		-0.07	2.91	



Regulatory networks cont.

- Fitness function
  - The sum, over all time steps, of the difference between the predicted and actual level of activation (expression) of the gene
  - Restrict the number of genes another gene can affect by adding a measure of null interactions to the objective function
- Complexity still to great: N^2 possible interactions
  - Experimental data suggests ≈ 6 genes may be affected by each other gene
- A modified GA could be used
  - Discover one column in the matrix at a time (Keedwell et al., 2003)
  - Limit the possible number of affected genes by a K-value



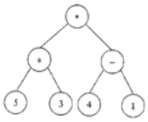
Multiple sequence alignment

- Task: determine the optimal alignment of a set of sequences by inserting gaps to realign the sequences
  - Relatively easy when considering just two sequences
- SAGA (Sequence Alignment Genetic Algorithm)
  - Notredame and Higgins, 1996
- Make use of a population of alignments
  - Performance is calculated in terms of the number of matched columns and the number of gaps introduced
  - Elitist approach (50% of alignments maintained)
  - 22 problem-specific operators
    - 19 mutation operators
    - 3 cross-over operations
- SAGA performs as well (or better) than established methods such as Clustal and MSA

Genetic programming

- One of the most recent techniques in AI
- Closely related to GA
  - Stochastic, population-based approach to search and optimization
  - Differs significantly in some operators used and in the representation of a solution
  - Selectors are the same as in GA
- Solution is represented by a parse tree
- Originally designed for ‘automatic programming’
  - Method for computers programming themselves
- The programs they derive can be used to represent a range of equations and functions which are based on the tree representation
- Successfully used in electronic circuit board design and automated programming tasks

Tree representation



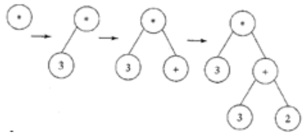
- Two types of elements
  - Terminals
    - Variables in a computer program
    - Constants
  - Operators
    - Perform operations on terminals
    - Binary operators (+,-,*,…)
    - Unary operators (log10, exp, sqrt, …)
- Rules:
  - The first node is an operator
  - An operator must have the number of arguments specified by its arity
  - An operator can take any operator or any terminal as an argument

Tree construction

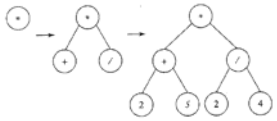
- Two commonly used methods for constructing a tree:
  - ‘Grow’ method:
    - First node is an operator
    - Elements are taken randomly from the complete set and added to the tree
    - Stops if tree has reached maximum depth or if all operators have terminals
  - ‘Full’ method:
    - First node is an operator
    - Operators are randomly added to the tree until maximum depth is reached
    - Terminals are then added to the final level of operators
- ‘Grow’ quicker than ‘full’ but may not be of desired depth
- ‘Grow’ may generate asymmetrical trees

Tree construction

- Grow



- Full



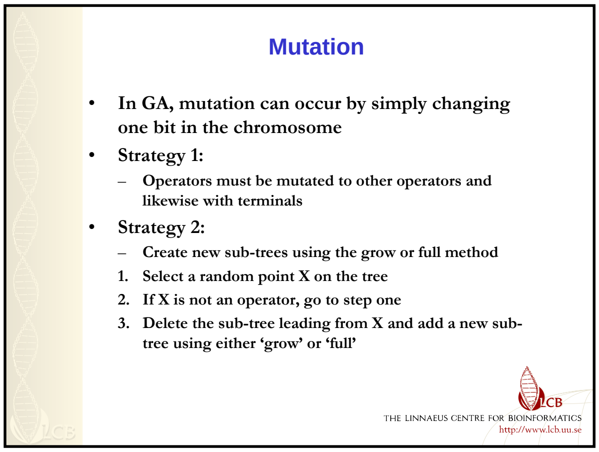
Fitness evaluation

- Different from GA
- A tree must be executed to give a result
  - Single or multiple values
  - The result(s) is compared to the required result of the user-defined objective function
- If the user can specify WHAT is required of a program in an objective function, GP can be used to generate a tree (program) that describes HOW to produce what is required
- Using GAs, we are interested in the actual solution
- Using GP, we are interested in the sequence of instructions for producing the solution
  - Desired and specific program behavior

Crossover

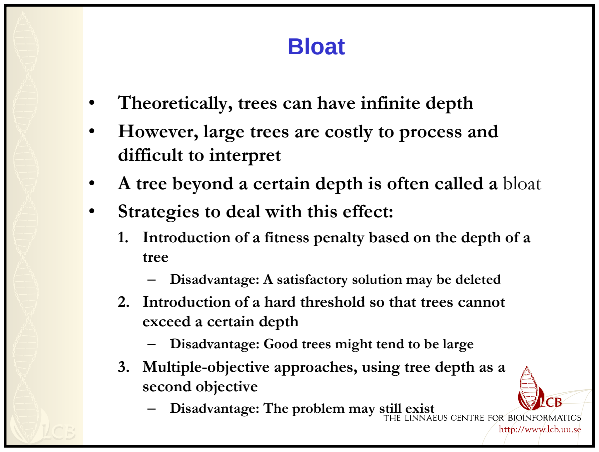
- Cut the trees at certain points and exchange genetic material
  - Operators must have the required number of terminals to operate correctly
  - Crossover location in both chromosomes corresponds to a sub-tree in each individual

1. Choose a random point on the chromosome
2. Evaluate whether this point is the start of a sub-tree, if not return to 1
3. Execute 1-2 on the second chromosome
4. Swap the sub-trees



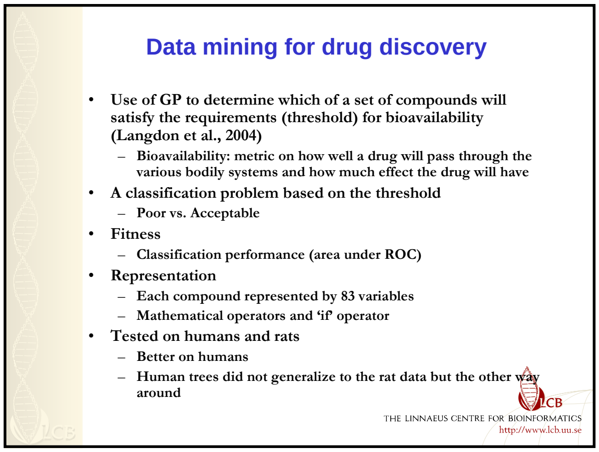
Mutation

- In GA, mutation can occur by simply changing one bit in the chromosome
- Strategy 1:
  – Operators must be mutated to other operators and likewise with terminals
- Strategy 2:
  – Create new sub-trees using the grow or full method
  1. Select a random point X on the tree
  2. If X is not an operator, go to step one
  3. Delete the sub-tree leading from X and add a new sub-tree using either 'grow' or 'full'



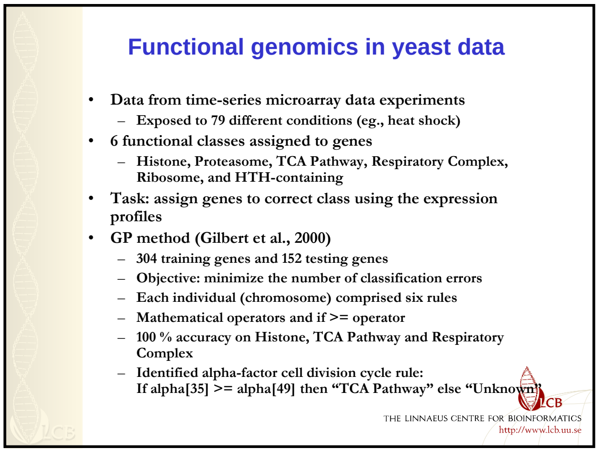
Bloat

- Theoretically, trees can have infinite depth
- However, large trees are costly to process and difficult to interpret
- A tree beyond a certain depth is often called a bloat
- Strategies to deal with this effect:
  1. Introduction of a fitness penalty based on the depth of a tree
     – Disadvantage: A satisfactory solution may be deleted
  2. Introduction of a hard threshold so that trees cannot exceed a certain depth
     – Disadvantage: Good trees might tend to be large
  3. Multiple-objective approaches, using tree depth as a second objective
     – Disadvantage: The problem may still exist



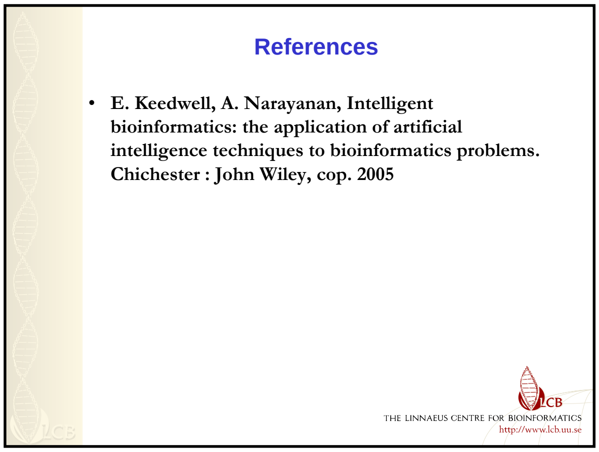
Data mining for drug discovery

- Use of GP to determine which of a set of compounds will satisfy the requirements (threshold) for bioavailability (Langdon et al., 2004)
  – Bioavailability: metric on how well a drug will pass through the various bodily systems and how much effect the drug will have
- A classification problem based on the threshold
  – Poor vs. Acceptable
- Fitness
  – Classification performance (area under ROC)
- Representation
  – Each compound represented by 83 variables
  – Mathematical operators and 'if' operator
- Tested on humans and rats
  – Better on humans
  – Human trees did not generalize to the rat data but the other way around



Functional genomics in yeast data

- Data from time-series microarray data experiments
  – Exposed to 79 different conditions (eg., heat shock)
- 6 functional classes assigned to genes
  – Histone, Proteasome, TCA Pathway, Respiratory Complex, Ribosome, and HTH-containing
- Task: assign genes to correct class using the expression profiles
- GP method (Gilbert et al., 2000)
  – 304 training genes and 152 testing genes
  – Objective: minimize the number of classification errors
  – Each individual (chromosome) comprised six rules
  – Mathematical operators and if >= operator
  – 100 % accuracy on Histone, TCA Pathway and Respiratory Complex
  – Identified alpha-factor cell division cycle rule:
    If alpha[35] >= alpha[49] then "TCA Pathway" else "Unknown"



References

- E. Keedwell, A. Narayanan, Intelligent bioinformatics: the application of artificial intelligence techniques to bioinformatics problems. Chichester : John Wiley, cop. 2005