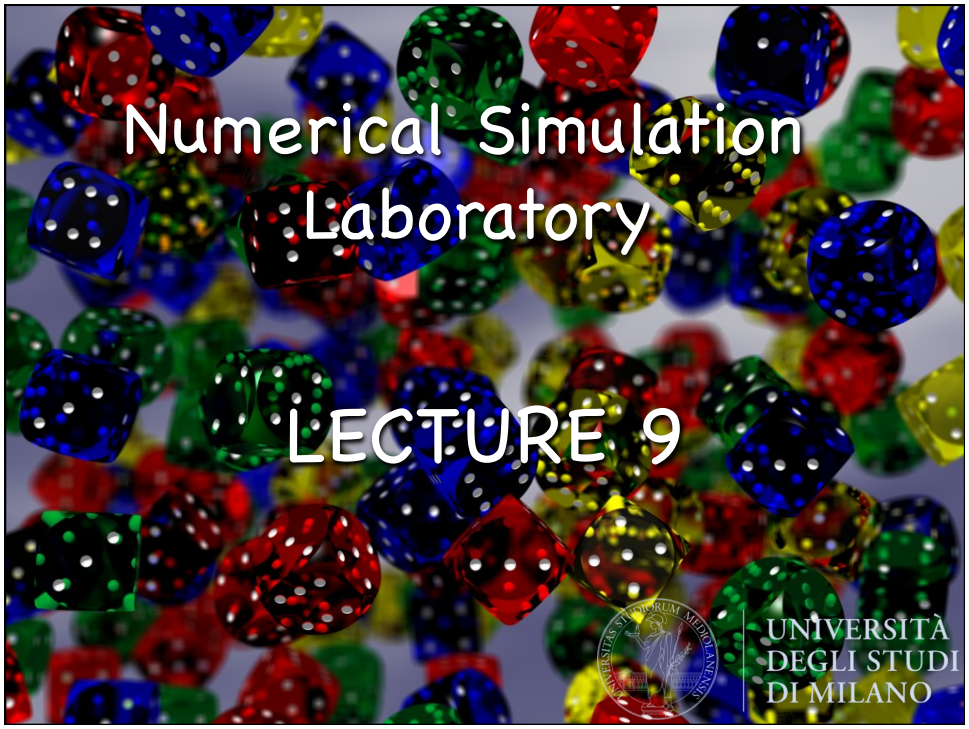
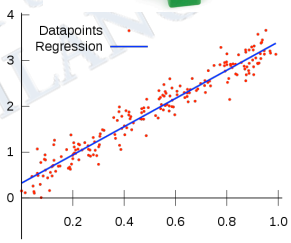


Outline

- Optimization
- Stochastic search and optimization
- Random search
- Simulated annealing
- Parallel tempering
- Genetic algorithms

Optimization

- Optimization problems are very common in everyday life and mathematical techniques of search and optimization are aimed at providing a formal means for making the best decisions in many real situations.
- For example frequently optimization is used as a tool: when a function with various parameters is fitted onto experimental data points, then one searches for the parameters which lead to the best fit.

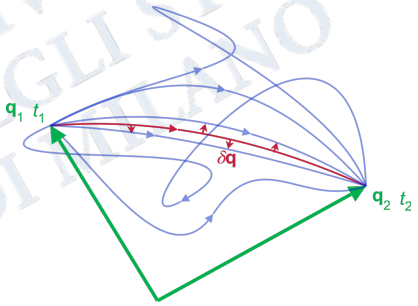


- During the last decades many problems in physics, and not only in physics, have turned out to be in fact optimization problems, or can be transformed into optimization problems
- In the simplest sense, an optimization can be considered as a minimization or maximization problem. E.g. the function $f(x)=x^2$ has a minimum $f_{min}=0$ at $x=0$. In general we can use the first derivative to determine the potential locations, and use the second derivative to verify if the solution is a maximum or minimum. However, for nonlinear, multimodal, multivariate functions, this is not an easy task.

Minimization principles

Note also that in Physics many systems are governed by minimization principles.

- In thermodynamics, a system coupled to a reservoir always equilibrate towards the state with minimal free energy
- The principle of least action is a variational principle that, when applied to the action of a mechanical system, can be used to obtain the equations of motion. This principle can be used even in general relativity where the Einstein–Hilbert action yields the Einstein field equations. Schwinger and Feynman, demonstrated how this principle can also be used in quantum calculations using path integration



- We have seen that in calculating the ground state behaviour of quantum systems, quite often a variational approach is applied: the energy of a test state vector is minimized with respect, to some parameters

Formalization

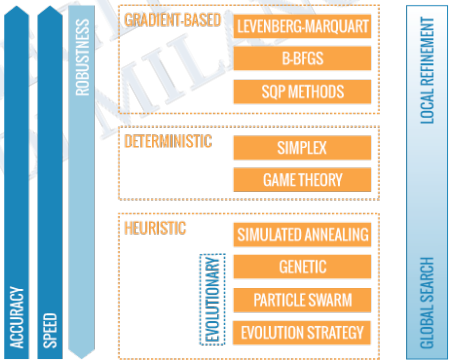
- Let $\mathbf{x} = (x_1,\ldots, x_n)$ be a vector with n elements which can take values from a domain $S(\equiv X^n)$: $x_i \in X$. The domain X can be either discrete, for instance $X = \{0,1\}$ or $X = \mathbb{Z}$ or X can be continuous, for instance $X = \mathbb{R}$
- A general optimization problem could be formalized as:

$$\text{minimize } L_1(\mathbf{x}), \ldots, L_I(\mathbf{x}), \mathbf{x}=(x_1, \ldots, x_n),$$
$$\text{subject to } h_j(x)=0, (j=1, \ldots, J) \text{ and } g_k(x)\leq 0, (k=1, \ldots, K)$$

- Where $L_1, \ldots, L_I$ are the objectives or cost/loss functions, while h_j and g_k are the equality and inequality constraints, respectively. Obviously, the simplest case of optimization is unconstrained function optimization. In the case when $I=1$, it is called single-objective optimization problem. When $I\geq 2$, it becomes a multi-objective optimization problem
- In general, all the functions L_i , h_j and g_k are nonlinear. In addition, some of the functions L_i may have discontinuities, and thus derivative information cannot be obtained. This may pose various challenges to the optimization procedure
- To solve an optimization problem, efficient search or optimization algorithms are needed. There are many optimization algorithms which can be classified in many ways, depending on the focus and characteristics

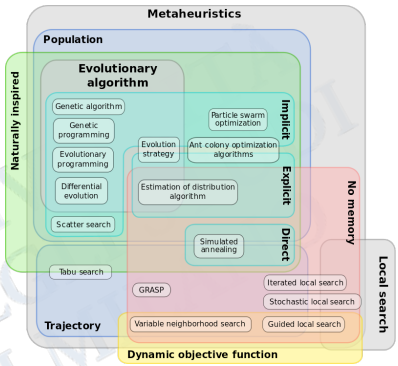
Classification

- If the derivative or gradient of a function is the focus, optimization can be classified into gradient-based algorithms and derivative-free or gradient-free algorithms. Gradient-based algorithms use derivative information, and they are often very efficient. Derivative-free algorithms use the values of the function itself.
- Optimization algorithms can also be classified as deterministic or stochastic/heuristic. If an algorithm works in a mechanical deterministic manner without any random nature, it is called deterministic. For such an algorithm, it will reach the same final solution if we start with the same initial point. On the other hand, if there is some randomness in the algorithm, the algorithm will usually reach a different point every time the algorithm is executed, even though the same initial point is used.



Optimization is huge!

7

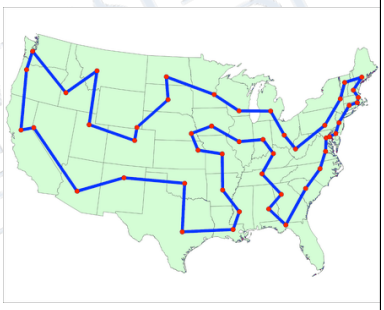


- Optimization algorithms can be classified also into trajectory-based and population-based. A trajectory-based algorithm typically uses a single agent at a time, which will trace out a path towards a solution as the iterations continue. On the other hand, population-based algorithms use multiple agents which will interact and trace out multiple paths
- Search capability can also be a basis for algorithm classification: local search and global search algorithms. Local search algorithms typically converge towards a local optimum, not necessarily (often not) the global optimum, and such an algorithm is often deterministic and has no ability to escape from local optima. On the other hand, for global optimization, local search algorithms are not suitable, and global search algorithms should be used. In essence, randomization is an efficient component for global search algorithms.
- Obviously, algorithms may not exactly fit into each category. It can be a so-called mixed type or hybrid, which us some combination of deterministic components with randomness, or combines one algorithm with another so as to design more efficient algorithms.

Example: the traveling salesman problem

8

A combinatorial optimization problem is an optimization problem, where an optimal solution has to be identified from a finite set of solutions



- The traveling salesman problem (TSP): consider n cities distributed randomly in a plane. The optimization task is to find the shortest round-tour through all cities which visits each city only once. The tour stops at the city where it started.
- The problem is described by:

$$X = \{1, 2, \cdots, n\} \qquad L(\vec{x}) = \sum_{i=1}^{n} d(x_i, x_{i+1})$$

where $d(x_i,x_{i+1})$ is the distance between cities x_i and x_{i+1} (and $x_{n+1}=x_1$).

- The optimum order of the cities for a TSP depends on their exact positions, i.e. on the random values of the distance matrix d, which represent an instance of the general optimization problem
- The constraint that every city is visited only once can be realized by constraining the vector **x** to be a permutation of the sequence [1,2,…,n]

9

- Because there is a finite number of possible paths, this is a combinatorial (discrete & finite) optimization problem, although one with a potentially very large number of elements
- A tour with n ≥ 3 cities has (n-1)!/2 possible unique tours (Unique here refers to fundamentally different ordering. For example, if n = 3, the tour 1-2-3-1 is considered equivalent to 1-3-2-1 by the symmetry of the cost between cities, where the indicated numbers 1, 2, or 3 represent the labels for the three cities).
- The TSP also belongs to a class of minimization problems for which the objective function L has many local minima.
- In practical cases, it is often enough to be able to choose from these a minimum which, even if not absolute, cannot be significantly improved upon. If one absolutely wants to get *the* global optimum for n very large, the only way is to let the computer run for a very, very long time
- In the language of combinatorial optimization, the problem is proved to be NP-hard. How to solve such problems? There is no universal method; when the number of variables n is large, computers are used to obtain a solution.

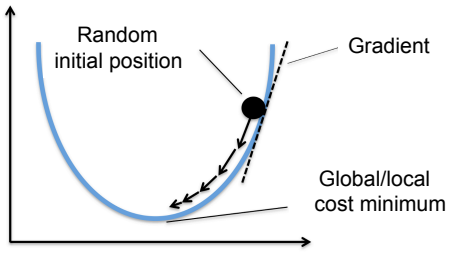
Gradient descent

10

- Gradient descent (steepest descent) is the simplest deterministic gradient-based iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point
- If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.
- Gradient descent is based on the observation that if the multi-variable cost function L(**x**) is defined and differentiable in a neighborhood of a point **x** then L(**x**) decreases fastest if one goes from **x** in the direction of the negative gradient of L at **x**, $-\nabla L(\mathbf{x})$. It follows that, if

$$\vec{x}_{n+1} = \vec{x}_n - \gamma \vec{\nabla} L(\vec{x}_n)$$

for $\gamma > 0$ small enough, then $L(\mathbf{x}_n) \geq L(\mathbf{x}_{n+1})$.

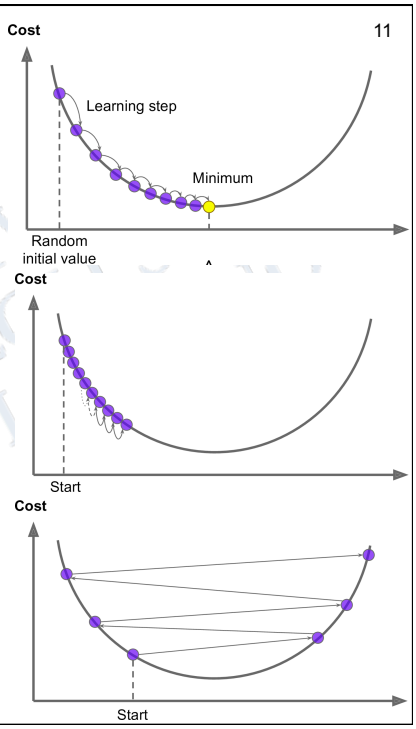


11

- With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of L, and considers the sequence $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, … such that

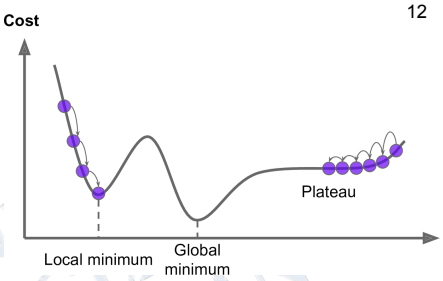
$$\vec{x}_{n+1} = \vec{x}_n - \gamma \vec{\nabla} L(\vec{x}_n) \qquad n \geq 0$$

- We obtain a monotonic sequence $L(\mathbf{x}_0) \geq L(\mathbf{x}_1) \geq L(\mathbf{x}_2) \geq \ldots$ so hopefully the sequence $\{x_n\}$ converges to the desired local minimum.
- An important parameter is the size of the steps, determined by the learning rate hyperparameter γ. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time
- On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before.



12

- Finally, not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult. The figure shows the two main challenges with gradient descent: if the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.



- Convergence to a local minimum can be guaranteed. When the function L is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.
- In many situation, however, the optimization is multi-objective and one can build a single cost/loss function by adding the whole set of objectives to obtain the sum-cost/loss function:

$$L(\vec{x}) = \sum_{i=1}^{n} L_i(\vec{x})$$

Stochastic gradient descent

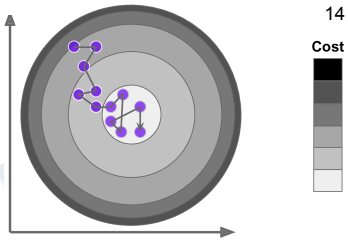
- When used to minimize the sum-cost/loss function, a standard (in this case usually called "batch"/group/set) gradient descent (BGD) method would perform the following iterations :

$$\vec{x}_{n+1} = \vec{x}_n - \gamma \vec{\nabla} L(\vec{x}_n) = \vec{x}_n - \gamma \sum_{i=1}^{I} \vec{\nabla} L_i(\vec{x}_n) \qquad n \geq 0$$

  where γ is the learning rate.

- In many cases, the summand functions have a simple form that enables inexpensive evaluations of the sum-cost function and the sum gradient. However, in other cases, evaluating the sum-gradient may require expensive evaluations of the gradients from all summand functions L_i .
- At the opposite extreme, stochastic gradient descent (SGD) just picks a random instance in the cost function set at every step and computes the gradients based only on that single instance. Obviously this makes the algorithm much faster since it has very little data to manipulate at every iteration.
- Moreover, when the cost function is very irregular, this can actually help the algorithm jump out of local minima, so SGD has a better chance of finding the global minimum than BGD does.

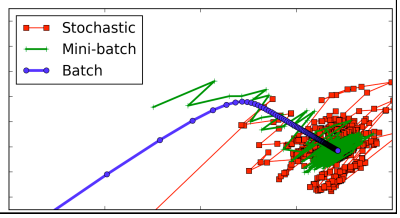
- On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than batch gradient descent: instead of gently decreasing until it reaches the minimum, the cost function with SGD will bounce up and down, decreasing only on average.



- To improve on this feature, at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch SGD computes the gradients on small random sets of instances called mini-batches.

$$\vec{x}_{n+1} = \vec{x}_n - \gamma \vec{\nabla} L(\vec{x}_n) = \vec{x}_n - \gamma \sum_{Random(i)}^{M<<I} \vec{\nabla} L_i(\vec{x}_n)$$

- The algorithm's progress in parameter space is less erratic than with SGD. As a result, Mini-batch SGD will end up walking around a bit closer to the minimum than SGD. But, on the other hand, it may be harder for it to escape from local minima



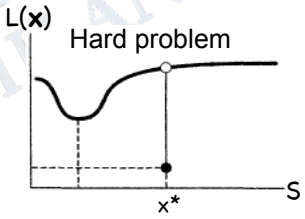
Stochastic search and optimization

- Given the difficulties in many real-world problems and the inherent uncertainty in information that may be available for carrying out the task, stochastic search and optimization methods have been playing a rapidly growing role.
- DEFINITION: Stochastic optimization methods are optimization algorithms which incorporate probabilistic elements, either in the problem data (the objective function, the constraints, etc.), or in the algorithm itself (through random parameter values, random choices, etc.), or in both. The concept contrasts with the deterministic optimization methods, where the values of the objective function are assumed to be exact, and the computation is completely determined by the values obtained so far.
- The two main problems of interest are:
  - Find the value(s) of $\mathbf{x} \in S = X^n$ that minimize a scalar-valued loss function $L(\mathbf{x})$
  - Find the value(s) of $\mathbf{x} \in S = X^n$ that solve the equation $g(\mathbf{x})=0$ for some function $g(\mathbf{x})$
- The second problem can be converted directly into an optimization problem by noting that a $\mathbf{x}$ such that $g(\mathbf{x})=0$ is equivalent to a $\mathbf{x}$ such that $L(\mathbf{x})=\|g(\mathbf{x})\|^2$ is minimized for any vector norm $\|\bullet\|$

15

Finding a global minimum?

- One of the major distinctions in optimization is between global and local optimization. In practice, however, a global solution may not be available and one must be satisfied with obtaining a local solution.
- For example, $L(\mathbf{x})$ may be shaped such that there is a clearly defined minimum point over a broad region of the domain S, while there is a very narrow spike at a distant point…
- This example illustrates that, in general, one cannot be guaranteed of ever obtaining a global solution. Without prior knowledge about the possible existence of such a point, no typical algorithm would be able to find this solution because there is nothing near $\mathbf{x}^*$ to indicate the presence of a minimum.

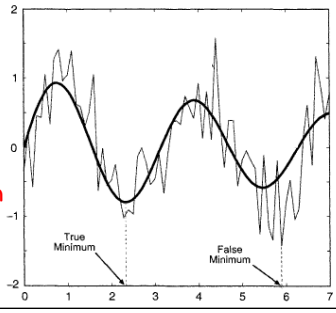


- Because of the inherent limitations of the vast majority of optimization algorithms, it is usually only possible to ensure that an algorithm will approach a local minimum with a finite amount of resources being put into the optimization process.

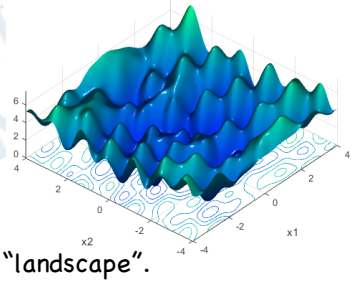
16

Stochastic data or procedures

- However, since the local minimum may still yield a significantly improved solution (relative to no formal optimization process at all), the local minimum may be a fully acceptable solution for the resources available (human time, money, computer time, etc.) to be spent on the optimization. However, we will also consider some algorithms (random search, simulated annealing, genetic algorithms, etc.) that are in principle able to find global solutions from among multiple local solutions.
- Stochastic search and optimization applies ineluctably where there is random noise in the measurement of L(**x**) or g(**x**)
- Partly-random input data arise in situation like simulation-based optimization where Monte Carlo simulations are run as estimates of an actual system, and problems where there is experimental error in the measurements of the criterion.
  Note that in the presence of noise the search for a global minimum could have no meaning at all!



- In such cases, knowledge that the function values are contaminated by random "noise" leads naturally to algorithms that use statistical inference tools to estimate the "true" values of the function and/or make statistically optimal decisions about the next steps.
- On the other hand, even when the data is exact, it is sometimes beneficial to deliberately introduce randomness into the search process as a means of speeding convergence and making the algorithm less sensitive to modelling errors.
- Further, the injected randomness may provide the necessary impetus to move away from a local solution when searching for a global optimum. Think about a multidimensional space S full of local minima; any kind of deterministic optimization procedure would be unable to explore all this complex "landscape".



- Indeed, this randomization principle is known to be a simple and effective way to obtain algorithms with almost certain good performance uniformly across all data sets, for all sorts of problems.

Random search & No free lunch theorem

- Random search simply consists in picking up random potential solutions and evaluating them. The best solution over a number of samples is the result of random search.
- In general there is a competition between algorithm efficiency and algorithm robustness. In essence, algorithms that are designed to be very efficient on one type of problem tend to be not reliably transferred to problems of a different type.
- The *No Free Lunch theorem* (Wolpert & Macready, 1995) states that no search algorithm is better than a random search on the space of all possible problems. In other words, if a particular algorithm does better than a random search on a particular type of problem, it will not perform as well on another type of problem, so that all in all, its global performance on the space of all possible problems is equivalent to a random search
- Hence, there can never be a universally best search algorithm. One must consider the characteristics of the problem together with the goals of the search and the resources available (computing power, human analysis time, etc.) in choosing an approach.

19

Metaheuristics

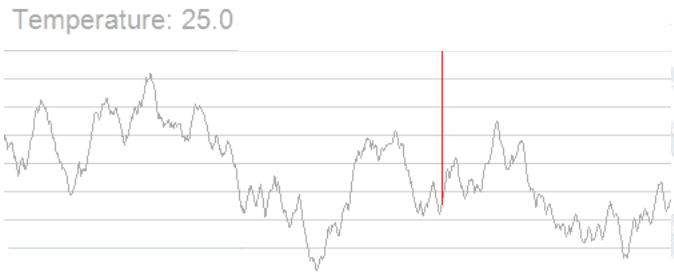
- Algorithms with stochastic components were often referred to as heuristic in the past, though the recent literature tends to refer to them as metaheuristics.
- We will follow the convention to call all modern nature-inspired algorithms metaheuristics. Loosely speaking, heuristic means to find or to discover by trial and error. Here meta- means beyond or higher level, and metaheuristics generally perform better than simple heuristics.
- Two major components of many metaheuristic algorithms are: intensification and diversification. Diversification means to generate diverse solutions so as to explore the search space on a global scale, while intensification means to focus the search in a local region knowing that a current good solution is found in this region. A good combination of these two major components will usually ensure that global optimality is achievable
- Examples of metaheuristic algorithms are: simulated annealing, parallel tempering, genetic algorithms, memetic algorithms, tabu search, ant colony optimization, particle swarm optimization etc. we will discuss simulated annealing, parallel tempering and genetic algorithms

20

Optimization: Simulated annealing

- At the heart of the method of simulated annealing is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal.
- At high T, the molecules of a liquid move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. The atoms are often able to line themselves up and form a pure crystal that is completely ordered over a distance up to billions of times the size of an individual atom in all directions.
- This crystal is the state of minimum energy for this system. The amazing fact is that, for slowly cooled systems, nature is able to find this minimum energy state. In fact, if a liquid is cooled quickly it does not reach this state but rather ends up in a polycrystalline or amorphous state having somewhat higher energy.
- So the essence of the process is slow cooling, allowing enough time for redistribution of the atoms as they lose mobility. This is the technical definition of annealing, and it is essential for ensuring that a low energy state will be achieved

- Any optimization problem can be 'embedded' in a statistical-mechanics/ annealing problem. The idea is to interpret the cost function L(x), $x \in X^n$, as the energy of a statistical-mechanics system and consider the Boltzmann distribution $p(x) = \exp[-\beta L(x)]/Z$
- In the low-temperature limit $\beta \to \infty$, the distribution becomes concentrated on the minima of L(x), and the original optimization setting is recovered.
- Since the Monte Carlo method provides a general technique for sampling from the Boltzmann distribution, one may wonder whether it can be used, in the $\beta \to \infty$ limit, as an optimization technique.
- The idea of simulated annealing consists in letting β vary with time. More precisely, one decides on an annealing schedule $\{(\beta_1, n_1); (\beta_2, n_2); \ldots (\beta_N, n_N)\}$, with inverse temperatures $\beta_i \in (0,\infty)$ and integers $n_i > 0$.
- The algorithm is initialized on a configuration $\mathbf{x}_0$ and executes n_1 Monte Carlo steps at temperature $1/\beta_1$, n_2 steps at temperature $1/\beta_2$, ..., and n_N steps at temperature β_N. The final configuration of each cycle i (with i = 1, ..., N-1) is used as the initial configuration of the next cycle.



- Mathematically, such a process is a time-dependent Markov chain.
- Within a specific temperature t=1/β being in the configuration **x** with "energy" L(**x**) one generates a new configuration **x'** with energy L(**x'**) which replaces the original configuration with probability:

$$P = \begin{cases} e^{-\beta(L(\vec{x}')-L(\vec{x}))} & if\ L(\vec{x}') > L(\vec{x}) \\ 1 & otherwise \end{cases}$$

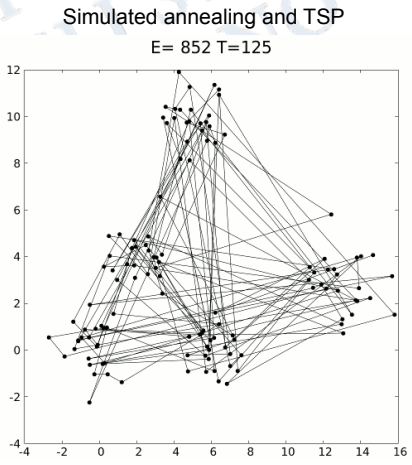
where β is the fictitious inverse temperature.

this can be easily sampled with a Metropolis algorithm.

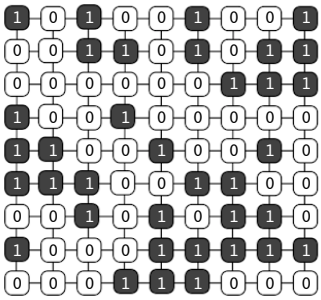
23

24

- At any given temperature such an (ergodic) Monte-Carlo process samples the configurations according to their thermodynamic probability. Thus, at high temperature moves with or against the gradient are accepted with almost equal probability. At low temperature only downhill moves are accepted.
- In simulated annealing one thus starts with high temperature simulation and gradually cools the system to zero temperature. If ergodicity is not lost during the cooling schedule, the simulation will stop in the global minimum of the energy landscape.
- Usually, due to the finiteness of the simulation time, simulated annealing allows us to obtain configurations close to the ground states, but not true ground states. This is however sufficient for most applications.



Spin models as a computational tool



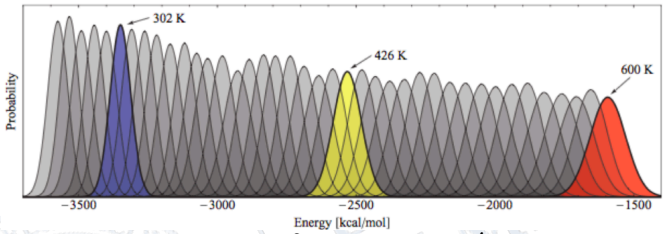
- Previously introduced spin models can also turn out to become a useful computational tool
- Consider a particular spin configuration of an Ising model; given such a configuration, like any other, we can think about it as the ground state of specific class of (presently unknown/unexpressed) Hamiltonians
- We can also associate a 1 to each up spin and a zero to each down spin
- From the point of view of information theory, considering spin as a binary alphabet, the given configuration could represent everything, from a number to one or more phrases.
- If this number (or anything else) is the solution of a problem we can try to solve our problem in two steps:

1. Solve an embedding problem, i.e. find one Hamiltonian for which this configuration is a ground state (not necessarily an easy task!)
2. Annealing: let your spin model to evolve toward its ground state (for example reducing its "temperature") and … read the solution!

Parallel tempering

- Low temperature simulations on rugged potential energy surfaces can be trapped for long times in similar metastable configurations because the energy barriers to structurally potentially competing different conformations can be very high.
- The parallel (or simulated) tempering technique was introduced to overcome the difficulties in the evaluation of thermodynamic observables for models with very rugged potential energy surfaces (PES) and applied previously in several protein folding studies. As an optimization procedure, it resembles the simulated annealing technique
- In parallel tempering we consider n systems. In each of these systems we perform a simulation in the canonical (NVT) ensemble, but each system is at a different temperature.
- Systems with a sufficiently high temperature pass over the potential. The low-temperature systems, on the other hand, mainly probe the local energy minima. The idea of parallel tempering is to include MC trial moves that attempt to "swap" systems that belong to different thermodynamic states, e.g., to swap a high-temperature configuration with a low temperature configuration.

- If the temperature difference between the two systems is very large, such a swap has a very low probability of being accepted instead of making attempts to swap between a low and a high temperature, we swap between ensembles with a small temperature difference.



- Let us call the temperature of system i, $T_i=1/\beta_i k_B$, and the systems are numbered according to an increasing temperature scale, $T_1< T_2< \ldots < T_n$, . We define an extended ensemble that is the combination of all n subsystems.
- The partition function of this extended ensemble is the product of all individual NVT_i ensembles:

$$Z_{PT} = \prod_{i=1}^{n} Z_{NVT_i} = \prod_{i=1}^{n} \frac{1}{\Lambda_i^{3N} N!} \int d\vec{x}_i \, e^{-\beta_i L(\vec{x}_i)}$$

27

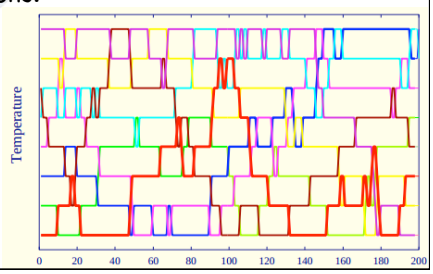
28

- To sample this extended ensemble it is in principle sufficient to perform NVT_i simulations of all individual ensembles. But we can also introduce a Monte Carlo move, which consists of a swap between two ensembles. The acceptance rule of a swap between ensembles i and j follows from the condition of detailed balance.
- It turns out to be (it should be accepted both for i and j):

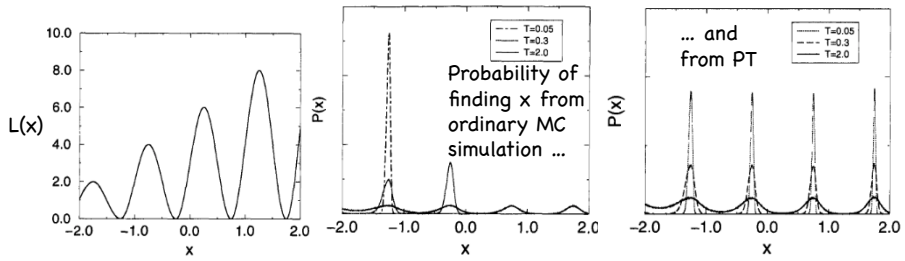
$$P = \min\left(1, e^{-\{(\beta_j-\beta_i)[L(\vec{x}_i)-L(\vec{x}_j)]\}}\right)$$

where β_i and $L(\mathbf{x}_i)$ are the inverse temperatures, energy and configuration of the i^{th} replic

- Since we know the total "energy" of a configuration anyway, these swap moves are very inexpensive since they do not involve additional calculations.
- The temperature scale for the highest and lowest temperatures is determined by the requirement to efficiently explore the configurational space and to accurately resolve local minima, respectively.



- Once parallel tempering is used for a ordinary simulation, the exchange mechanism improves the configurational averaging of the low-temperature simulations, because the exchange with high-temperature simulations provides a mechanism to overcome the high energy barriers between low-lying metastable configurations.



- Applied as an optimization technique, however the simulation associated with the lowest temperature will typically yield the estimate for the global optimum, while all others are required to generate different configurations.
- The computational effort of the method rises linearly with the number of temperatures.

29

Introduction to Genetic Algorithms

30

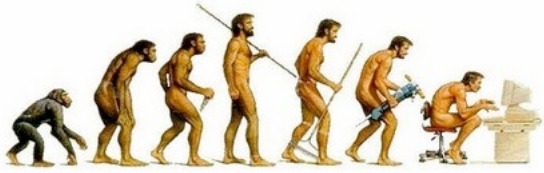
- Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his collaborators in the 1960s and the 1970s. Their goal originally was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which natural adaptation might be imported into computer systems.
- Holland's 1975 book *Adaptation in Natural and Artificial Systems* presented GAS as an abstraction of biological evolution. Gas soon became a general framework to solve optimization problems based on the mechanics of natural selection and natural genetics
- Why use evolution as an inspiration for solving computational optimization problems?
- As we have seen, many computational problems require searching through a huge number of possibilities for solutions. Such search problems can often benefit from an effective use of parallelism, in which many different possibilities are explored simultaneously in an efficient way.

The 2007 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer program to create the best radiation pattern.

The appeal of evolution

- Biological evolution is an appealing source of inspiration for addressing these optimization problems. Evolution is, in effect, a method of searching among an enormous number of possibilities for "solutions".
- In biology the enormous set of possibilities is the set of possible genetic sequences, and the desired "solutions" are highly fit organisms that are well able to survive and reproduce in their environments.
- Evolution can also be seen as a method for designing innovative solutions to complex problems. Seen in this light, the mechanisms of evolution can inspire computational search/optimization methods.
- Of course the fitness of a biological organism depends on many factors, for example, how well it can adapt to the environment and how well it can compete with or cooperate with the other organisms around it. The fitness criteria continually change as creatures evolve, so evolution is searching a constantly changing set of possibilities.
- Furthermore, natural evolution is a massively parallel search method at work!: rather than work on one species at a time, evolution tests and changes millions of species in parallel.

31



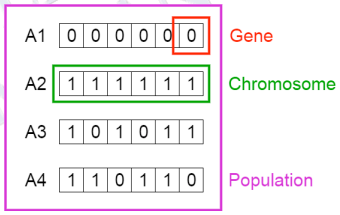
- The "rules" of evolution are remarkably simple: species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations. Yet these simple rules are thought to be responsible, in large part, for the extraordinary variety and complexity we see in the biosphere.
- Holland was the first to attempt to put computational evolution on a firm theoretical footing (see Holland 1975). This theoretical foundation, based on the notion of "schemas" was the basis of almost all subsequent theoretical work on genetic algorithms.
- GAs attractively contain the two major components of metaheuristic algorithms which are: intensification and diversification. What is needed to obtain these components is both computational parallelism and an intelligent strategy for choosing the next set of sequences to evaluate.

32

Elements of Genetic Algorithms

- In the original Holland's idea GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a sort of "natural selection" together with the genetics-inspired operators of crossover, mutation, and maybe others.

- Each chromosome consists of "genes" (e.g., bits), each gene being an instance of a particular "allele" (e.g., 0 or 1).



- The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more descendants than the less fit ones.
- Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single-chromosome organisms; mutation randomly changes the allele values of some locations in the chromosome.

33

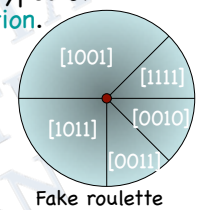
Population, Genes and Alleles

34

- Genetic algorithms are different from more normal optimization and search procedures in four ways:
  - GAs in general work with a coding of the parameter set, not the parameters themselves
  - GAs search from a population of "points", not a single point
  - GAs use direct information, not derivatives or other auxiliary knowledge
  - GAs use probabilistic transition rules, not deterministic rules
- Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some specific (in general finite) alphabet
- The population is the ensemble of the chromosomes existing at a given time. Chromosomes (a possible solution) could be:
  - Bit string [0101...1100] - Real numbers [43.2 -33.1 ... 0.0 89.2]
  - Program elements (genetic/evolutionary programming)
  - ...any data structure
- Genes: elements of a chromosome
  Bit string: the chromosome [1001101] has 7 genes
- Alleles: possible values of a gene
  Bit string: 2 values (0,1)

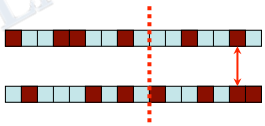
Operators and definitions

- The simplest form of genetic algorithm involves three types of operators: selection, crossover (single point), and mutation.
- Selection: this operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce (Fitness: the measure of goodness of a solution in an optimization problem).



Fake roulette wheel selection

- Crossover: when two individuals have been selected, both parents pass their chromosomes onto their offspring. The two chromosomes come together and swap genetic material. In binary GAs crossover is performed by swapping a part of binary strings between two solutions at a randomly chosen cross-site with some probability.



- Mutation: conversion of genes from one to another. In Binary GAs mutation is performed by converting some random bit of a binary string into its complementary bit (i.e. a 1 to a 0 or vice versa) with some probability. Mutation will help prevent the population from stagnating. It adds diversity.
- Note that crossover and mutation destroy old solutions.

35

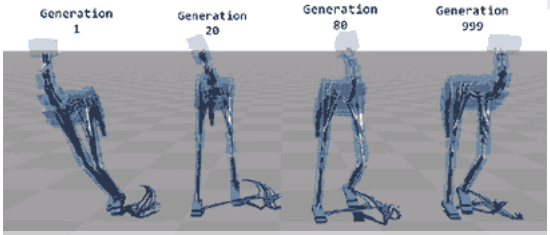
A simple genetic algorithm

- Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n l-bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness f(x) of each chromosome x in the population.
3. Repeat the following steps until n offspring have been created:
   A. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.
   B. With probability P_c crossover the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.
   C. Mutate the two offspring at each locus with probability P_m, and place the resulting chromosomes in the new population.
4. Replace the current population with the new population.
5. Repeat: Go to step 2.

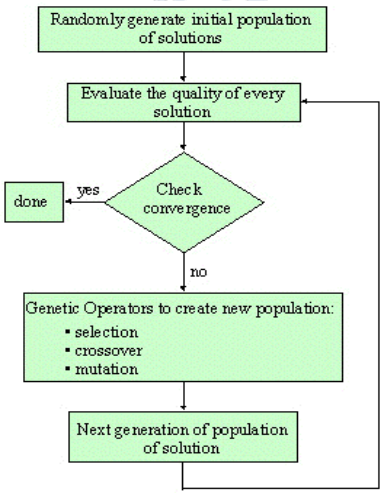
36

- Each iteration of this process is called a generation. The entire set of generations is called a run. At the end of a run there are often one or more highly fit chromosomes in the population.
- The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as the size of the population and the probabilities of crossover and mutation, and the success of the algorithm often depends greatly on these details.



Flexible Muscle-Based Locomotion for Bipedal Creatures
In ACM Transactions on Graphics, Vol. 32, T. Geijtenbeek et al.

Flow chart



37

For example one can have:

- Solutions Generational GA: entire populations replaced with each iteration
- Steady-state GA: a few members replaced each generation
- Elitism: Some elite (good) solutions are carried onto the next generation without being destroyed.

GAs are especially useful when

- The search space is large, complex or the knowledge about it is scarce or it is difficult to encode to narrow the search space.
- Traditional search methods fail.

Moreover GAs:

- support multi-objective optimization
- give always an answer; and this answer gets better with time
- are inherently parallel

The traditional theory of Gas assumes in fact that, at a very general level of description, GAs work by discovering, emphasizing, and recombining good "building blocks" of solutions in a highly parallel fashion.

38

The "building block": schema

- The idea here is that good solutions tend to be made up of good building blocks—combinations of bit values that confer higher fitness on the strings in which they are present.
- Holland ('75) introduced the notion of schema to formalize the informal notion of "building blocks."
- DEFINITION: a schema is an equivalence class of chromosomes
- A schema is a set of bit strings that can be described by a template made up of ones, zeros, and asterisks, the asterisks (*) representing wild cards (or "don't cares").
- For example, the schema H=[1****1] represents the set of all 6-bit strings that begin and end with 1. (here I use Goldberg's notation in which H stands for "hyperplane." H is used to denote schemas because schemas define hyperplanes–"planes" of various dimensions in the l-dimensional space of length-l bit strings.)
- The strings that fit this template (e.g., [100111] and [110011]) are said to be instances of H (e.g. [1****1])

39

Instances, order and defining length

40

- DEFINITION: a schemas is of order n iff n is the number of genes different from an asterisks (*); formally n=o(H)
- DEFINITION: the defining length of a schemas H is the maximum distance in H between two defined genes; formally d(H)
- Thus, in the example above, the schema H=[1****1] is said to have two defined bits (non-asterisks) or, equivalently, to be of order 2. Its defining length (the distance between its outermost defined bits) is 5.

- A central belief of traditional GA theory is that schemas are (implicitly) the building blocks that the GA processes effectively under the operators of selection, mutation, and single-point crossover.
- How does the GA process schemas? In the case of a binary alphabet, any given bit string of length l is an instance of 2^l different schemas.

How do GA work?

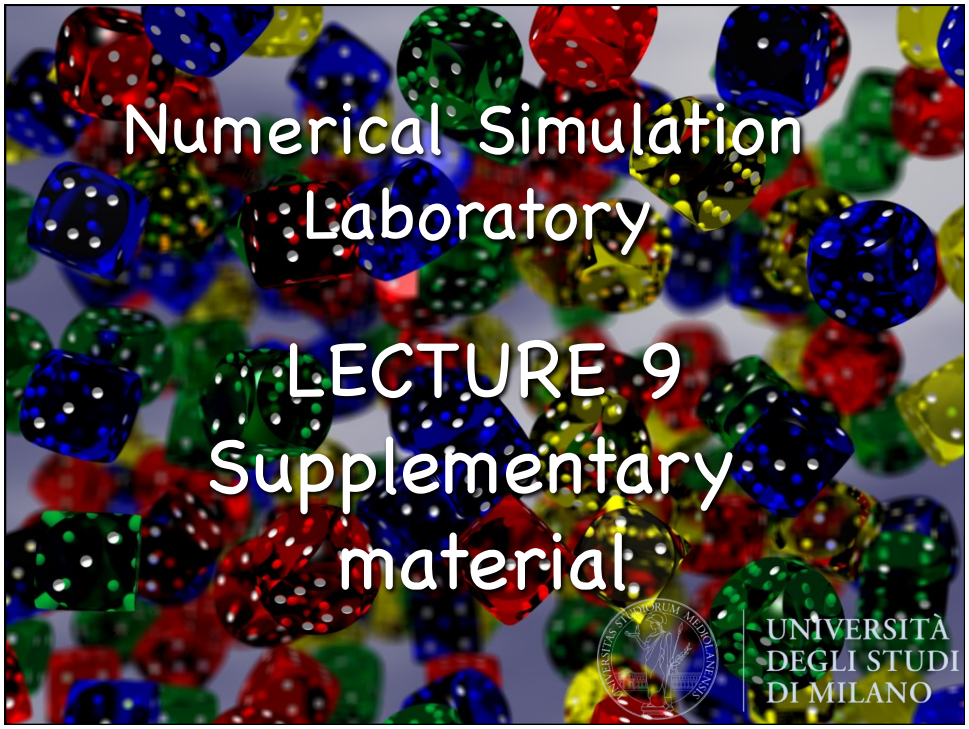
- For example, the string [11] is an instance of [**] (all four possible bit strings of length 2), [*1] , [1*] , and [11] (the schema that contains only one string, 11).
- Thus, any given population of n strings contains instances of between 2^l (maximal homogeneity) and $n \times 2^l$ (maximal diversity) different schemas. If all the strings are identical, then there are instances of exactly 2^l different schemas; otherwise, the number is less than or equal to $n \times 2^l$.
- This means that, at a given generation, while the GA is explicitly evaluating the fitnesses of the n strings in the population, it is actually implicitly estimating the average fitness of a much larger number of schemas
- The Schema Theorem (Holland '75, see supplementary material) says that "short, low-order schemas whose average fitness remains above the mean will receive exponentially increasing numbers of samples (i.e., instances evaluated) over time". The Schema Theorem is a lower bound, since it deals only with the destructive effects of crossover and mutation.

41

Lecture 9: Suggested books

- M.P. David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*
- J.C. Spall, *Introduction to Stochastic Search and Optimization*, Wiley (2003)
- A.K. Hartmann & H. Rieger, *Optimization Algorithms in Physics*, *Wiley* (2002)
- A.Brabazon, M. O'Neill, S. McGarraghy, *Natural Computing Algorithms*, Springer (2015)

42



The Schema Theorem

- Just as schemas are not explicitly represented or evaluated by the GA, the estimates of schema average fitnesses are not calculated or stored explicitly by the GA. However, as will be seen below, the GA's behaviour, in terms of the increase and decrease in numbers of instances of given schemas in the population, can be described as though it actually were calculating and storing these averages.
- We can calculate the approximate dynamics of this increase and decrease in schema instances as follows:
  - ◇ Let H be a schema with at least one instance x_i present in the population at time t, which retains N individuals.
  - ◇ Let N(H,t) (≤N) be the number of instances of H at time t, ...
  - ◇ and let F(H,t) be the observed average fitness of H at time t (i.e., the average fitness of instances of H in the population at time t):

$$F(H,t) = \frac{\sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{N(H,t)}$$

  being f the fitness function.
  - ◇ We want to calculate E[N(H,t+1)] , the expected number of instances of H at time t+1.

44

- Assume that the probability for a string x_i (a chromosome) to be selected is equal to

$$P(x_i) = \frac{f(x_i)}{\sum_j f(x_j)}$$

  i.e. it is equal to the ratio among the fitness of x_i and the sum of the fitnesses of the population at time t.
- Then, assuming x_i is in the population at time t, and x_i is an instance of H, and (for now) ignoring the effects of crossover and mutation, we have that the expected number of instances of H at time t+1 is

$$E[N(H,t+1)] = N \times \sum_{j=1}^{N} P(x_j)\delta_{(x_j \in H)} = N \times \frac{\sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{\sum_{j=1}^{N} f(x_j)} =$$

$$N \times N(H,t) \frac{\frac{1}{N(H,t)} \sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{\sum_{j=1}^{N} f(x_j)} = N(H,t) \frac{F(H,t)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)} \qquad (*)$$

- Thus even though the GA does not calculate F(H,t) explicitly, the increases or decreases of schema instances in the population depend on this quantity: schemas with a greater average fitness will posses a greater number of instances as the generations evolve

45

- By assuming that $F(H,t) = [\Sigma_j f(x_j)/N](1+c) > \Sigma_j f(x_j)/N$ it follows that

$$E[N(H,t+1)] = N(H,t) \frac{\left[\frac{1}{N}\sum_{j=1}^{N} f(x_j)\right](1+c)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)} = N(H,t)(1+c)$$

- Then starting from t=0 and assuming c a constant we obtain:

$$E[N(H,t+1)] = N(H,t)(1+c)^t$$

  which is a *geometric progression*, the discrete analogous of the exponential form
- Thus, the selection operator assigns an increasing (decreasing) number of instances to schemas with high (low) idoneity following an exponential law.
- Crossover and mutation can both destroy and create instances of H. For now let us include only the destructive effects of crossover and mutation, those that decrease the number of instances of H.
- Including these effects, we modify the right side of the previous equation to give a lower bound on E[N(H,t+1)].

46

- Let P_c be the probability that single-point crossover will be applied to a string, and suppose that an instance of schema H is picked to be a parent.
- Schema H is said to "survive" under single-point crossover if one of the offspring is also an instance of schema H. We can give a lower bound on the probability $S_c(H)$ that H will survive to a single-point crossover:

$$S_c(H) \geq 1 - P_c \times \left(\frac{d(H)}{l-1} \right)$$

where d(H) is the defining length of H and l is the length of bit strings in the search space.

- That is, crossovers occurring within the defining length of H can destroy H (i.e., can produce offspring that are not instances of H), so we multiply the fraction of the string that H occupies by the crossover probability to obtain an upper bound on the probability that it will be destroyed. (The value is an upper bound because some crossovers inside a schema's defined positions will not destroy it, e.g., if two identical strings cross with each other.)
- Subtracting this value from 1 gives a lower bound on the probability of survival $S_c(H)$. In short, the probability of survival under crossover is higher for shorter schemas.

47

- The disruptive effects of mutation can be quantified as follows: Let P_m be the probability of any bit being mutated. Then $S_m(H)$, the probability that schema H will survive under mutation of an instance of H, is equal to $(1-P_m)^{o(H)}$, where o(H) is the order of H (i.e., the number of defined bits in H).
- That is, for each bit, the probability that the bit will not be mutated is $1-P_m$, so the probability that no defined bits of schema H will be mutated is this quantity multiplied by itself o(H) times. In short, the probability of survival under mutation is higher for lower-order schemas.
- These disruptive effects can be used to amend equation (*) :

$$E[N(H,t+1)] \geq N(H,t) \frac{F(H,t)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)} \left(1 - P_c \frac{d(H)}{l-1}\right) (1-P_m)^{o(H)}$$

- This is known as the Schema Theorem (Holland '75). It describes the growth of a schema from one generation to the next. The Schema Theorem is often interpreted as implying that short, low-order schemas whose average fitness remains above the mean will receive exponentially increasing numbers of samples (i.e., instances evaluated) over time.

48

- The Schema Theorem is a lower bound, since it deals only with the destructive effects of crossover and mutation.
- However, crossover is believed to be a major source of the GA's power, with the ability to recombine instances of good schemas to form instances of equally good or better higher-order schemas; this is known as the Building Block Hypothesis (Goldberg '89).
- In evaluating a population of n strings, the GA is implicitly estimating the average fitnesses of all schemas that are present in the population, and increasing or decreasing their representation according to the Schema Theorem.
- This simultaneous implicit evaluation of large numbers of schemas in a population of n strings is known as implicit paralelism (Holland '75).
- The effect of selection is to gradually bias the sampling procedure toward instances of schemas whose fitness is estimated to be above average.
- Mutation is what prevents the loss of diversity at a given bit position.
- In the end note that GAs can be seen as Markov processes!

49