GENETIC ALGORITHMS

- The slides follow closely the text and examples from :
- Z. Michalewicz – “Genetic Algorithms + Data Structures = Evolution Programs”, Springer Verlag
- M. Mitchell – “An Introduction to Genetic Algorithms”, MIT Press

A NON-CLASSICAL APPROACH

- Self-adaptation: optimisation as discovery (rather than invention)
- Less approximations
- More than one good solution
- Processing directly computer representations of solutions (not those of intermediary mathematical objects)
- Turning the combinatorial explosion to the user's benefit
- Meta-optimisation (of the algorithm)

WHAT ARE WE DEALING WITH ?

- “General-purpose” search methods
- Probabilistic algorithms
- Inspired from natural evolution
- Soft-computing paradigm
- Useful in
  - optimisation
  - machine learning
  - design of complex systems

LESS APROXIMATION

Classical -*hard computing*

Understand the problem

Set up a model (*)

Devise theoretical results

Design an algorithm (*)

Code it, run the program
→ solution (*)

Evolutionary -*soft computing*

Understand the problem

Set up a model (*)
representation/function

Design the EA

Code, run experiments
→ solutions (o)

OPTIMISATION

- Discrete or continuous variables
- Dynamic (moving) optimum
- Multicriterial
- Procedural view:
  - Step by step improvement
  - “Needle in the haystack”
- Criteria: accuracy and precision

STRATEGIES IN PROBLEM SOLVING (1)

- Deterministic approach
  - **Exact determinist algorithms**: exact solutions
    - Complexity hierarchy for <u>problems</u>
      - Knapsack, TSP, MaxClique – each NP-complete
  - **Heuristics**: approximate solutions
    - Time complexity vs. precision for <u>problems</u>
      - Knapsack polynomial for any precision
      - TSP polynomial down to 50% precision
      - MaxClique **still exponential** for any precision!

STRATEGIES IN PROBLEM SOLVING (2)

- PROBABILISTIC APPROACH
  - **Probabilistic algorithms**
    - approximate solutions
    - Monte Carlo, Las Vegas, …
    - Evolutionary Algorithms, Hillclimbing, Simulated Annealing, Tabu Search…
    - Time complexity vs. precision in a statistical setting
      - Genetic Algorithms have a polynomial scheme;
      - some Evolutionary Strategies have probability 1 to find the exact solution
      - …

DESIGN OF COMPLEX SYSTEMS

- **Top-down** approach – the procedure:
  1. Design the general view of the system, with successive levels;
  2. While there are unvisited levels do
     - Descend one level and design that level.
- **Bottom-up** approach - alternatives:
  1. **Analytic / synthetic** procedure:
     - design one component at a time
     - join the components together (level by level)
  2. **"Evolutionary"** procedure:
     - Design a frame for evolution
     - Design very basic features / components
     - Let the components evoluate (the system evolves itself)
- Examples: ANN, sorting networks etc.

COMPUTING (1)

- HARD COMPUTING
  - Rigid, “static” models: no adaptation, or adaptation driven from outside
  - The model for a problem is completely described beforehand, not to be changed as solving goes on
  - Calculus, Numerical Analysis (Newton method), …

COMPUTING (2)

- SOFT COMPUTING
  - **Self-adapting** models
  - Only the frame for autoadaptation is specified
  - Properties of the **instance** of the problem - fully exploited
  - Probabilistic modelling and reasoning
    - Evolutionary Computing
    - Artificial neural networks
    - Fuzzy sets

LOCAL SEARCH (MAXIMIZATION): ITERATED HILLCLIMBING

begin t←0;

repeat

local←FALSE;

select a candidate solution (bitstring) $\mathbf{v_c}$ at random; evaluate $\mathbf{v_c}$;

repeat generate the *length* strings Hamming-neighbors of $\mathbf{v_c}$ {or select some among them};

select $\mathbf{v_n}$, the one among these which gives the largest value for the objective function f;

if $f(\mathbf{v_c}) < f(\mathbf{v_n})$ **then** $\mathbf{v_c} \leftarrow \mathbf{v_n}$ **else** *local* ← TRUE

until *local*;

t←t+1;

until t=Max **end.**

LOCAL SEARCH (MAXIMIZATION): SIMULATED ANNEALING

begin

t←0; initialize temperature T;

select a current candidate solution (bitstring) $\mathbf{v_c}$ at random;

evaluate $\mathbf{v_c}$;

repeat

repeat select at random one of the *length* strings Hamming-neighbors of $\mathbf{v_c}$;

if $f(\mathbf{v_c}) < f(\mathbf{v_n})$ **then** $\mathbf{v_c} \leftarrow \mathbf{v_n}$ **else**

if *random* [0,1) < *exp* $((f(\mathbf{v_n}) - f(\mathbf{v_c}))/T)$ **then** $\mathbf{v_c} \leftarrow \mathbf{v_n}$

until *(termination condition);*

T←g(T,t); t←t+1 {g(T,t)<T, ∀t}

until (*halting-criterion*)

end.

EVOLUTIONISM

- Darwin (approx. 1850) – natural selection
  - “the Jenkins nightmare”: if combination is like mixing fluids, then … homogenous population – no selection / improvement.
- Mendel (1865): Genetics - discrete, not continuous
- de Vries, Correns, von Tschermak (1900) – “rediscover” and spread Genetics
- T. Morgan – chromosomes as sequences of genes
- Cetverikov, others (1920) – join natural selection and genetics: **evolution**.

GLOSSARY (1)

- Chromosome (individual) – a sequence of genes (…)
- Gene – atomic information in a chromosome
- Locus – position of a gene
- Allele – all posible values for a locus
- Mutation – elementary random change of a gene
- Crossover (recombination) – exchange of genetic information between *parents*
- Fitness function – actual function to optimise; *environment*

GLOSSARY (2)

- Population – set of candidate solutions
- Generation – one iteration of the evolutionary process
- Parent – chromosome which “creates” offspring via a genetic operator
- Descendant (offspring) – chromosome obtained after applying a genetic operator to one or more parents
- Selection – (random) process of choosing parents and / or survivors
- Genotype – representation at the individual level
- Phenotype – (theoretical) object represented through genotype
- Solution given by a GA – best candidate in the last generation

EVOLUTIONARY COMPUTING

- Evolution Strategies
- Evolutionary Programming
- Genetic Algorithms
  - Genetic Programming

EARLY IDEAS

- Evolution-inspired algorithms for optimisation: Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962), Red e.a. (1967). No follow-up in their respective lines.
- Computer simulations of evolution for controlled experiments: Baricellli (1957-1962), Fraser (1957), Martin e.a. (1960).

THE FIRST TWO MAJOR DIRECTIONS

- EVOLUTION STRATEGIES
  - A method to optimise real-valued parameters
  - Random mutation and selection of the fittest
  - Ingo Rechenberg (1965,1973); Schwefel (1975,1977)
- EVOLUTIONARY PROGRAMMING
  - Fogel, Owens, Walsh (1966)
  - Random mutations on the state-transition diagrams of finite state machines; selection of the fittest

A BREAKTHROUGH: GAs

- John Holland – 60's and 70's.
- Goal: not solving specific problems, but rather study adaptation
- "Adaptation in Natural and Artificial Systems" (1975)
- Abstraction of biological evolution
- Theoretical study (schema theorem).

NEW IDEAS

- Messy Genetic Algorithms (Goldberg)
- Genetic Programming (Koza)
- Co-evolution (Hillis)
- Memetic / cultural algorithms
- Micro-economic algorithms
- Statistical study of “fitness” between representation and fitness function, operators, etc.

EVOLUTIONARY COMPUTING - SHARED FEATURES

- maintain a population of representations of candidate solutions
- which are improved over successive generations
- by means of mutation(, crossover,…)
- and selection based on individual merit
- assigned to individuals through evaluation against fitness function (“environment”)

DATA STRUCTURES

- Genetic Algorithms
  - standard: bitstring representations (integers, real numbers, permutations etc.);
  - others: bidimensional string structures (graphs, matrices etc.); neighborhood-based (“island”); “geographically”-distributed etc.
  - GP: trees (programs)
- Evolution Strategies: floating point representations (real numbers)
- Evolution Algorithms: finite state machines

THEORETICAL MODELS

- Schema model (Holland)
  - Scheme;
  - Building-blocks hypothesis
  - Implicit parallelism
- Markov-chain models (→linked processes)
- Vose models (statistical mechanics)
- Bayesian models (Muehlenbein, …)

THE ELEMENTS OF A GENETIC ALGORITHM

- A population of *pop_size* individuals chromosomes
  - An initialisation procedure
- A representation (genotype) of candidate solutions (phenotype)
- An evaluation (fitness) function
- A selection scheme
- Genetic operators (mutation, crossover, …)
- Parameters (size, rates, scalability, halting condition, …)

META-OPTIMISATION

- Design of a GA for a given problem:
  - choose a representation (out of a few possible);
  - choose a fitness function (various models; scalable);
  - choose a selection scheme (around 10 possible);
  - choose operators (or invent them);
  - set the parameter values (tens/hundreds).
- A moderate evaluation: 5*10*10*10*100≈500.000: hundreds of thousands of GAs for a given problem.
- Most of them probably perform poorly, but few of them usually outperform other kinds of algorithms
- Finding the “good” GA for a problem is an optimisation problem in itself

GENERAL SCHEME OF A GA

begin

- $t \leftarrow 0$
- Initialise P(t)
- Evaluate P(t)
- **While not** (halting condition) **do begin**
  - $t \leftarrow t+1$
  - Select P(t) from P(t-1)
  - Apply operators to P(t)
  - Evaluate P(t)
- **end**

end

Except possibly for the Evaluation step, the scheme is polynomial!

This scheme is valid for (most) evolutionary algorithms.

THREE SAMPLE GAs

1. Numerical optimisation
   - Representing numbers
   - The fitness function is the studied function
2. Combinatorial optimisation
   - Representing graphs / permutations
   - The objective function becomes fitness
3. Machine learning
   - Representing strategies
   - Optimisation is the overall gain.

A SIMPLE NUMERICAL EXAMPLE (1)

- Find up to 6 digits $x_0 \in [-1; +2]$ to maximize $f(x) = x\sin(10\pi x) + 1$.
- **Representation.**

  $[-1; +2]$ to be divided into $3 \cdot 10^6$ equal subintervals. 22 bits are required:

  $2097152 = 2^{21} < 3 \cdot 10^6 < 2^{22} = 4194304$
- *Decoding*.

  $(b_{21}\, b_{20} \ldots b_0) \rightarrow \sum b_i \cdot 2^i = x' \rightarrow x = -1 + x' \cdot (3/(2^{22} - 1))$

A SIMPLE NUMERICAL EXAMPLE (2)

- (000000000000000000000) → -1.
- (111111111111111111111) → +2.
- v_1=(100010111011010100011 1)→x_1= +0.637197
- v_2=(000000111000000000010000)→x_2= -0.958973
- v_3=(111000000**<u>0</u>**111111000101)→x_3= +1.627888
- **Evaluation**. eval(v_1) = f(x_1) = 1.586245;

eval(v_2)=f(x_2) = 0.078878; eval(v_3)=f(x_3) = 2.250650

- **Mutation**. Possible improvement with *no arithmetic calculation*!
- v'_3 = (111000000**<u>1</u>**111111000101) →

x'_3 =+1.630818 → eval(v_3') = f(x_3') = 2.343555 > f(x_3) = eval(v_3) !

A SIMPLE NUMERICAL EXAMPLE (3)

- **Crossover**.
- Suppose v_2 , v_3 were selected as parents.
- v_2 = (00000 | 01110000000010000)
- v_3 = (11100 | 00000111111000101)
- Offspring:
- v'_2=(00000|00000111111000101)→x'_2=-0.99811
- v'_3=(11100|01110000000010000)→x'_3=+1.66602
- eval(v'_2) = f(x'_2) = 0.940865 > eval(v_2)
- eval(v'_3) = f(x'_3) = 2.459245 > eval(v_3) > eval(v_2)

A SIMPLE NUMERICAL EXAMPLE (4)

- **(Main) parameters**.
- *pop_size* = 50; p_c = 0.25; p_m = 0.01.
- **Experiments**.
- Solution after 150 generations:
- v_{max} = (111100110100010000101)
- x_{max} = 1.850773; eval(v_{max}) = f(x_{max})= 2.850227
- Evolution of f($x_{max\_so\_far}$) over generations:

1	10	40	100
1.441942	2.250363	2.345087	2.849246

A COMBINATORIAL EXAMPLE

- **Travelling Salesperson Problem**. Given the cost of travelling between every two cities, find the least expensive itinerary which passes exactly once through each city and returns to the starting one.
- **An evolutionary approach**.
  - Representation: integer vector for **permutations**.
  - Evaluation: sum of costs of successive edges in the tour.
  - Mutation: switch two genes.
  - Crossover (OX): choose a subsequence from a parent, for the rest preserve the relative order from the other one

P: (1 2 3 **4 5 6 7** 8 9 10 11 12) (7 3 1 **11 4 12 5** 2 10 9 6 8)

O: (3 1 11 **4 5 6 7** 12 2 10 9 8) (1 2 3 **11 4 12 5** 6 7 8 9 10)

ITERATED PRISONER'S DILEMMA (1)

- Gain table (one iteration):

Player1	Player2	Points1	Points2
Defect	Defect	1	1
Defect	Cooperate	5	0
Cooperate	Defect	0	5
Cooperate	Cooperate	3	3

- Maximize the number of points

  = 5 - # years_ in_prison.

- Always defect? Overall in many iterations, worse than "always cooperate".
- Example: arms race.

ITERATED PRISONER'S DILEMMA (2)

- Axelrod (1987).
- Deterministic strategies, based on previous three moves: $4^3 = 64$ different histories.
- Strategy: what move to make against each possible history: 64 bits (instead of "C/D").
- 6 extra bits for the history of the actual iterated game.
- 70 bits represent a strategy and a chromosome: 2^{70} strategies!

ITERATED PRISONER'S DILEMMA (3)

- Random initialisation of the population;
- One generation:
  - Evaluate each player (chromosome) against other players on the actual games;
  - Select chromosomes for applying operators (average score→breed once;

    one SD above average→breed twice;

    one SD below average→no breeding);
  - Mutation and crossover applied (random pairs).

ITERATED PRISONER'S DILEMMA (4)

- TIT FOR TAT won two tournaments.
  - cooperate;
  - while (not halt) do what the oponent did in the previous step.
- Experiments – first round: *fixed environment*.
  - *pop_size* = 20; *max_gen* = 50; *no_runs* = 40;
  - evaluation of each chromosome: average of iterated games against 8 best human strategies;
  - results: ≈ TIT FOR TAT **or better** in the static *fitness landscape* – the 8 strategies)!
  - TIT FOR TAT is general; GA solutions were adapted to the specific environment.
  - browsing only 20*50=1000 strategies out of the 2^{70}!

ITERATED PRISONER'S DILEMMA (5)

- Experiments – second round: *changing environment*.
  - each individual evaluated against all 20;
  - environment changes as individuals evoluate;
  - fitness: average score of the individual;
  - early generations: uncooperative strategies;
  - after 10-20 geenrations: more and more cooperation in the TIT FOR TAT manner: reciprocation!
- Further ideas:
  - no crossover (Axelrod 1987);
  - expandable memory (Lindgren 1992).

HOW DO GENETIC ALGORITHMS WORK?

- Implementation and use of a Genetic Algorithm
  1. Define its elements
     - Representation
     - Fitness function
     - Selection mechanism
     - Genetic operators
     - Parameters
  2. Design the experiment (includes optimising the GA)
     - Stand-alone
     - Comparison to non-GA algorithms
  3. Actually perform the experiment
  4. Interpret the results

WHAT ABOUT THE PROBLEM ?

1. State the problem as optimisation of a real function (clustering?)
2. A step-by-step solution improvement strategy possible
   - Otherwise, create a frame for this (e.g., #_of_minterms)
3. Uni-criterial optimisation. Otherwise:
   - build one global criterion (linear – timetable –/ non-linear) or
   - work in a Pareto setting
4. Optimisation as maximisation
   - Otherwise, use **-f** instead of **f**
5. Optimisation of positive functions
   - Otherwise, translate the function by a constant
6. Restrictions:
   - Encoded; penalties; repair.

SATISIFIABILITY PROBLEM

- Straightforward to encode solutions (a string of k bits is a truth assignment for k variables)
- Straightforward evaluation (the truth value of the resulting expression)
- But with only two possible values of the fitness function, impossible to “learn” / improve step by step.
- Fitness: number of terms which are evaluated as “true” – step-by-step improvement possible.

A GA FRAME FOR NUMERICAL OPTIMISATION

- f: $\mathbf{R}^k \rightarrow \mathbf{R}$, with restrictions $x_i \in [a_i ; b_i]$, $f > \mathbf{0}$.
- Optimisation using d=6 decimal positions.
- D_i is to be cut into $(b_i - a_i) \cdot 10^6 \leq 2^{m_i} - 1$. ($m_i$ minimal)
- Each x_i would then be represented by a substring of m_i bits.
- Decoding: $x_i = a_i + decimal(d_{m_i-1} \ldots d_1 d_0) \cdot \dfrac{b_i - a_i}{2^{m_i} - 1}$
- A candidate solution is represented by a bitstring of length

$$m = \sum_{i=1}^{k} m_i .$$

ELEMENTS OF A GA FOR NUMERICAL OPTIMISATION (1)

- **Initialization**:
  - population v_1 , v_2 , …, v_{pop_size} .
  - random bitwise or
  - heuristic (initiate with some found solutions).
- **Evaluation**:
  - decode the bitstring **v** into a vector **x** of numbers
  - calculate **eval(v)=f**(**x**) .
- **Halting condition**: a maximum number of generations (iterations) may be enforced.

ELEMENTS OF A NUMERICAL OPTIMISATION GA: **SELECTION** FOR SURVIVAL (1)

- Probability distribution based on fitness values
- Best known procedure is "roulette wheel":
  - Slots proportional to fitness.
  - Calculate (steps 1., 3. ,4. , for i=1..*pop_size*)
    1. Individual fitnesses: $\mathbf{eval}(v_i)$
    2. Total fitness: $F = \sum_{i=1}^{pop\_size} \mathbf{eval}(v_i)$
    3. Individual selection probabilities: $p_i = \mathbf{eval}(v_i) / F$
    4. Cumulative selection probabilities: $q_i = \sum_{j=1}^{i} p_j$

       $q_0 = 0$.

ELEMENTS OF A NUMERICAL OPTIMISATION GA: **SELECTION** FOR SURVIVAL (2)

- Idea: create an intermediate population
  - Spin the roulette wheel *pop_size* times. Each time:
    - select **for survival** one chromosome from the current population
    - put it into the intermediate population.
- Implementation:
  - *generate* a random (float) number r in [0 , 1];
  - **if** ($q_{i-1} < r \leq q_i$) **then** *select* $\mathbf{v}_i$.
- In principle, best chromosomes get more and more copies; average stay even; worst die off.

OPERATORS FOR A NUMERICAL OPTIMISATION GA: **CROSSOVER** (1)

- p_c – the probability of crossover – gives the expected number of chromosomes which undergo crossover: $p_c \cdot$*pop_size*.
- **Selection for crossover** (→parents).
  - In the intermediate population

for (each chromosome) ***do***

generate a random (float) number *r* in [0 , 1];

if $r < p_c$ ***then*** *select* the current chromosome.

mate the **parents** randomly.

OPERATORS FOR A NUMERICAL OPTIMISATION GA: **CROSSOVER** (2)

- ***for*** (each paired parents) ***do***
  - *generate* a random number $pos \in \{1,..,m-1\}$
  - *replace* the two parents by their offspring.

$$P1 = (a_1\ a_2\ \ldots a_{pos}\ a_{pos+1}\ \ldots\ a_m)$$
$$P2 = (b_1\ b_2\ \ldots b_{pos}\ b_{pos+1}\ \ldots\ b_m)$$

$$O1 = (a_1\ a_2\ \ldots a_{pos}\ b_{pos+1}\ \ldots\ b_m)$$
$$O2 = (b_1\ b_2\ \ldots b_{pos}\ a_{pos+1}\ \ldots\ a_m)$$

- Uniform distribution?

OPERATORS FOR A NUMERICAL OPTIMISATION GA: **MUTATION**

- p_m – probability of mutation – gives the *expected* number of mutated bits: $p_m \cdot m \cdot pop\_size$
- Each bit in the population has the same chance to undergo mutation.

for (each chromosome after crossover) ***do***

  for (each bit within the chromosome) ***do***

    generate a random number *r* in [0 , 1];

    if (*r* < p_m**)** ***then*** *mutate* the current bit.

A DETAILED EXAMPLE (Michalewicz)

max **f**

$\mathbf{f}(x_1,x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$,

with restrictions: $-3.0 \le x_1 \le 12.1$, $4.1 \le x_2 \le 5.8$,

with 4 decimal places for each variable.

- **Representation**:
  - $2^{17} < 151000 < 2^{18}$; $2^{14} < 17000 < 2^{15}$: 33 bits.
  - **v** = (010001001011010000.111110010100010)
  - 010001001011010000 → 1.0524
  - 111110010100010 → 5.7553
  - **eval**(**v**) = **f**(1.0524, 5.7553) = 20.2526.
- Set: *pop_size* = 20; $p_c = 0.25$; $p_m = 0.01$.

FIRST GENERATION (1)

$\begin{cases} \text{Random initialisation} \\ \text{Evaluation (best\_so\_far: } v_{15}\text{; worst: } v_2\text{)} \end{cases}$

- $F=\sum \mathbf{eval}(v_i)=387.7768$
- Field of probabilities (p_i, i=1..*pop_size*).
- Cumulative probabilities (q_i, i=0..*pop_size*).
- Spinning the roulette wheel *pop_size* times.
- New population (*same generation*).

FIRST GENERATION - CROSSOVER

- **Selection for crossover**.
  - Some chromosomes selected ($p_c \cdot$ *pop_size* ?).
  - ***If*** (odd number) ***then*** randomly *decide* {drop / add}.
  - Randomly couple together selected parents.
- **Example – one pair**. Generate $pos \in \{1..32\}$. $pos = 9$.

v_2'= (100011000.10110100111100000111001 0)
v_{11}'= (111011101.10111000010001111101111 0)

v_2''= (100011000. 10111000010001111101111 0)
v_{11}''=(111011101. 10110100111100000111001 0)

FIRST GENERATION - MUTATION

- **Selection for mutation**.
  - 660 times generate a random number in [0;1]
  - Distribution? (uniform / targeted)
  - Expected number of mutated bits: $p_m \cdot m \cdot pop\_size = 6.6$

r	Bit position (population)	Chromos. number	Bit position (chromos.)
0.000213	112	4	13
0.009945	349	11	19
0.008809	418	13	22
0.005425	429	13	33
0.002836	602	19	8

A DETAILED EXAMPLE - CONCLUSIONS

- Best chromosome in the final, 1000th generation:

  v_{11}=(110101100000010010001100010110000)

  eval (v_{11}) = f (9.6236 , 4.4278) = 35.4779.
- 396th generation: **eval** (*best_so_far*) = 38.8275!
- Store *best_so_far* as the current solution given by the GA (not *best_of_generation*).
- Reason: stochastic errors of sampling (pseudo-random numbers, finite populations, finite number of generations).

THE NEED FOR THEORETICAL MODELS

- What laws describe the macroscopic behaviour of Genetic Algorithms?
- How do low-level operators (selection, crossover, mutation) result in the macroscopic behaviour of Genetic Algorithms?
- What makes a problem suited for GAs?
- What makes a problem unsuited for GAs?
- Which performance criteria are relevant?
- See "*Foundations of Genetic Algorithms*" series (started in 1991).

EXISTING THEORETICAL MODELS

- **Schema Theorem**:
  - "The two-armed bandit problem" (Holland 1975);
  - "Royal Roads" (Forrest, Holland, Mitchell, 1992)
- **Exact mathematical models** of simple GAs (Vose 1991, Goldberg 1987, Davis 1991, Horn 1993, Whitley 1993)
- **Statistical-Mechanics** approaches (Prügel-Bennett 1994)

THE **SCHEMA** APPROACH

- "Schema" ("schemata") is a formalization of the informal notion of "building blocks".
- A schema is **a set of bitstrings** (hyperplane) which is described by **a template** made up of ones, zeroes and asterisks (wild cards).
- Example: **m**=10.
  - H_1={(0011000111), (0011000110)}; S_1=(001100011*)
  - H_2={(0011000111), (0001000111), (0011001111), (0001001111)}; S_2=(00*100*111)

SCHEMAS

- A schema with *a* wildcards describes a hyperplane containing 2^a strings.
- A string of length *m* is matched by (is represented, is an **instance** of) 2^m schemata.
  - m=2. **01** is an instance of: **01**, **0***, ***1**, ****.
- For all the 2^m strings of length *m* there exist exactly 3^m schemata.
- Between 2^m and $pop\_size \cdot 2^m$ of them may be represented in one generation.
- When is the union / difference of two schemas a schema?

CHARACTERISTICS OF SCHEMATA (1)

1. Defined bits of a given schema S:

all non-wildcard positions of S.

2. Order of a schema S, o(S):

number of defined bits in S

- how specific is S?
- $o(S_2) = 8$.

3. Defining length of a schema S, δ(S):

distance between its outermost <u>defined</u> bits.

- how compact is the information in the schema?
- $\delta(S_2) = 10-1 = 9$.
- if S has only one defined bit, then $\delta(S)=0$.

CHARACTERISTICS OF SCHEMATA (2)

4. Average ("static") fitness of a schema S in a Genetic Algorithm: the arithmetic mean of the evaluations of all instances str_i of S:

$$\boldsymbol{eval}(\mathrm{S}) = (1/2^r)\cdot\sum_i \boldsymbol{eval}(str_i)$$

- r – the number of wildcards in S.

5. Average fitness of a schema S **in generation** n **of a specific Genetic Algorithm**:

$$\boldsymbol{eval}(\mathrm{S},n) = (1/\mathrm{k})\cdot\sum_j \boldsymbol{eval}(str_j)$$

- k – the number of instances of S in the population P(n).

EVALUATION OF SCHEMA

- Schemas **are not** explicitly represented / evaluated by the GA.
- Estimates of schema average fitnesses **are not** calculated / stored by the GA.
- However, <u>the number of instances</u> of given schemas in successive generations increases or decreases, **as if they were**! (is this a **genotype** for the **phenotype** "estimate"?)
  - With random initialization, in the first generation, on the average, half of the population will be instances of the schema S=(1*…*) and the rest - instances of the schema T=(0*…*). All evaluations of chromosomes can be considered as estimating **eval**(S) and **eval**(T).

SCHEMA DYNAMICS

- Notation: "$x \in H$" for "x is an instance of H"
- Let **H** be a schema with
  - $\eta(\mathbf{H},\mathbf{t})$ instances in generation **t**;
  - **eval(H,t)** - the observed average fitness**.**

$$\mathbf{E}(\eta(\mathbf{H},\mathbf{t}+1)) = ?$$

(what is the *expected number* of instances of **H** in generation **t**+1 , under the following assumptions:

  - "roulette wheel" selection procedure
  - one-point crossover
  - standard mutation).

THE EFFECT OF SELECTION

- One chromosome x will have in the next generation an expected number of f(x) / $\bar{f}(t)$ offspring.
  - f(x) = **eval** (x); $\bar{f}(t)$ - average fitness in gen. **t**

$$E(\eta(\mathbf{H},\mathbf{t}+1)) = \sum_{x\in\mathbf{H}} \mathrm{eval}(x)/\bar{f}(t) = \eta(\mathbf{H},\mathbf{t})\cdot \mathrm{eval}(\mathbf{H},\mathbf{t})/\bar{f}(t),$$

since $\mathrm{eval}(\mathbf{H},\mathbf{t}) = (\sum_{x\in H} f(x)) / \eta(H,t)$.

- The GA does not calculate explicitly **eval**(**H**,**t**), but this quantity decides the number of instances of **H** in subsequent generations.

THE EFFECT OF CROSSOVER

- Effect: destroy or create instances of **H**.
- Consider the disruptive effect only → lower bound for $\mathbf{E}(\eta(\mathbf{H},\mathbf{t}+1))$.
- Suppose an instance of **H** is a parent.
- The lower bound for the probability $S_c(\mathbf{H})$ that **H** will survive after crossover:

$$S_c(\mathbf{H}) \geq 1 - p_c \cdot (\delta(\mathbf{H})/(m-1))$$

- *The probability of survival under crossover is higher for more compact ("shorter") schemas.*

THE EFFECT OF MUTATION

- Again, consider the destructive effect only.
- The probability of survival under mutation:

$$S_m(\mathbf{H}) = (1-p_m)^{o(\mathbf{H})}$$

(the probability that no defined bits will be mutated).

- *The probability of survival under mutation is higher for lower-order schemas.*

SCHEMA THEOREM

$$E(\eta(H, t+1)) \geq \frac{eval(H,t)}{\bar{f}(t)} \cdot \eta(H,t) \cdot (1 - p_c \cdot \frac{\delta(H)}{m-1}) \cdot (1 - p_m)^{o(H)}$$

- Describes the minimal growth of a schema from one generation to the next.
- One way to intepret it:
  - "Short,
  - low-order,
  - *constantly* above average schemas
  - receive exponentially increasing numbers of instances in successive generations of a GA"
  - (because the factor of growth is eval(H,t)/$\bar{f}(t)$).

OPERATORS AS "BUILDERS"

- The Schema Theorem gives a lower bound because it neglects the "creativity" of operators.
- It is however believed that **crossover** is a major source of GA power.
  - recombines instances of good schemas to create instances of (oftenly enough) at least that good schemas
- "**Building Block Hypothesis**": the *supposition* that this is how GAs work.
- **Mutation** provides *diversity* even when the population tends to converge
  - if in the population a bit becomes definitely 0, then only the mutation gives a chance to still try 1.

IMPLICIT PARALLELISM

- In one generation, a Genetic Algorithm:
  - estimates the average fitnesses of **all schemas** which have instances in that generation;
  - increases / decreases the representation of **these** schemas in the next generation accordingly.
- **Implicit evaluation of a large number of schemas** using only *pop_size* chromosomes.
- This is **implicit parallelism** (Holland 1975).
- Different from *inherent parallelism* (GAs lend themselves to parallel implementations).

SELECTION IN ACTION

- Selection **biases** the sampling procedure towards instances of schemas whose average fitness is estimated to be above average.
- The estimation is more and more accurate, as the “age” of the population increases.
- There are however counterexamples (“deceiving problems”).

ADAPTATION REVISITED

- Holland: an adaptive system should **identify, test** and **incorporate** structural properties which are likely to give better performance in some environment.
- **Exploration vs. Exploitation**:
  - search for new, useful adaptations vs.
  - use and propagation of adaptations.
- Neglecting one of them may lead to:
  - overadaptation (inflexibility to novelty) “stuck”
  - underadaptation (few/no gained properties).
- GA should have a proper balance between them.

THE TWO-ARMED BANDIT PROBLEM

- A model in statistical decision theory, adaptive control (Bellman 1961).
- Holland (1975) used it to study how the Genetic Algorithm allocates samples to schemas.

- A gambler has N coins to play a two-armed slot machine. The goal is to maximize the N-times payoff.
  - the two arms are labeled A_1 and A_2;
  - their mean payoffs per trial are μ_1 and μ_2;
  - which are stable over the experiment (stationary, independent processes);
  - the variances are σ_1 and σ_2 respectively;
  - all μ_1, μ_2, σ_1, σ_2 are unknown to the player;
  - she can only estimate them by playing coins in each arm.

TWO-ARMED BANDIT USEFUL

- How should coins be allocated to each arm, given:
  - the observed payoffs
  - the respective estimates of the average payoffs μ_1, μ_2
  - the estimates of their respective variances σ_1, σ_2 ?
- Resource allocation under uncertainty.
- Not for **guessing** the better arm, but rather for **maximizing** the payoff (on-line!)
- It is a combined goal, which could be losely described as: "find the solution using a minimum of resources".
- Holland: 3^m schemas → 3^m arms. Strategy: allocate exponentially more trials to the "better" arms.
- Maximization of on-line performance of GAs.

A SOLUTION TO THE TWO- ARMED BANDIT PROBLEM

- Assume $\mu_1 \geq \mu_2$:
  - A_1 **has** higher average payoff than A_2.
- After N trials, let:
  - $A_H(N, N\text{-}n)$ be the arm with **observed** higher payoff;
  - $A_L(N, n)$ be the arm with **observed** lower payoff;
  - n be the number of trials allocated to $A_L(N, n)$;
  - N-n – the number of trials allocated to $A_H(N, N\text{-}n)$.
- *Find n^* to maximize expected profits over N trials.*
- Ideally: $N \cdot \mu_1$ (all trials allocated to the true best).
- A "loss" is a trial allocated to the true worse arm.

SOURCES OF PROFIT LOSS

1. $A_L(N, n) = A_1$
   - Over the N-n trials given to $A_H(N, N\text{-}n)$, the gambler lost expected profits of $(N\text{-}n)\cdot(\mu_1 - \mu_2)$.
     - $A_H(N, n)$ may change over time, but this is irrelevant
2. $A_L(N, n) = A_2$
   - The loss in expected profits comes now from the n trials given to $A_L(N, n)$; it is exactly $n\cdot(\mu_1 - \mu_2)$
3. Let q be the probability of case 1.:

   $q = Prob(A_L(N, n) = A_1)$

ANALYSIS OF RESOURCE ALLOCATION FOR 2AB (1)

- Let L(N-n,n) be the losses over N trials:

  $L(N\text{-}n,n) = q\cdot(N\text{-}n)\cdot(\mu_1 - \mu_2) + (1\text{-}q)\cdot n\cdot(\mu_1 - \mu_2)$

- Find n^* which minimizes L(N-n,n).

$$\frac{dL}{dn} = (\mu_1 - \mu_2)\cdot(1 - 2q + (N - 2n)\frac{dq}{dn}) = 0$$

But $q = Prob(\frac{S_2^{N-n}}{N-n} > \frac{S_1^n}{n}) = Prob(\frac{S_1^n}{n} - \frac{S_2^{N-n}}{N-n} < 0)$,

where the two S's are the sums of the payoffs of all allocated trials to A_1, respectively A_2.

ANALYSIS OF RESOURCE ALLOCATION FOR 2AB (2)

- The two S's are actually random variables and so is there difference.
- What is (an approximation of) the area given by the part of the distribution which is less than zero?
  - Holland : the central limit theorem (normality);
  - later on, Frantz: the theory of large deviations!
- Frantz: the optimal allocation, n^*, of trials to the worse arm A_2 is approximated by (c_i are constants): $$n^* \approx c_1 \cdot \ln\left(\frac{c_2 \cdot N^2}{\ln(c_3 \cdot N^2)}\right)$$

ANALYSIS OF RESOURCE ALLOCATION FOR 2AB (3)

- We get:

$$N - n^* \approx e^{n^*/2c_1} \cdot \sqrt{\frac{\ln(c_3 N^2)}{c_2}} - n^*$$

- As n^* increases: $N - n^* \approx e^{cn^*}$
- **The optimal allocation of trials to the observed better arm, $N-n^*$, should increase exponentially with the number of trials allocated to the observed worse arm.**

INTERPRETATION OF THE SOLUTION

- 2AB illustrates perfectly the **“exploration” vs. “exploitation”** problem faced by any adaptive system.
- GA: there are 3^m rather than 2 arms.
- The schema theorem states that, under its assumptions, the **GA embodies a version of the optimal 2AB strategy** above:
- *The population increases exponentially w.r.t. the trials allocated to worse observed schemas.*

THE GENETIC ALGORITHM VS. THE OPTIMAL 2AB STRATEGY

- Grefenstette e.a. discuss the fitness function:

$$f(x) = \begin{cases} 2 \text{ if } x \in 111*\ldots* \\ 1 \text{ if } x \in 0***\ldots* \\ 0 \text{ otherwise} \end{cases}$$

- **eval**(1*…*) = ½ (3 instances out of 4 give 0, one gives 2)
- **eval**(0*…*) = 1
- **eval**(1*…*, n) → **eval**(111*…*, n) ≈ 2 after some generations
- Instances of 111*…* - also instances of 1*…* - will be strongly selected, much more often than instances of 0*…*
- High variance of **eval**(1*…*) ("2" vs. "0")!

GA VS. 2AB: THE DIFFERENCE

- 2AB: the two random variables describing each arm are **independent**.
- GA: different "arms" (schemas) **interact**
  - the observed payoff of schema 1*…* is strongly influenced by the observed payoff of 111*…*
  - additional instances of 1*…* bring no additional information about the payoff of 1*…* , since they are likely to be **all** instances of 111*…*
- The GA does not sample schemas independently for estimating their true payoffs.

HOW CLOSE IS THE GA TO THE 2AB ?

- Actually, the GA is not playing a 3^m–armed bandit with all schemas competing as arms.
- The GA plays a **2^k-armed bandit** for each schema of order k (Holland).
- The GA allocates exponentially more samples to the best observed sub-schema of a schema than to the next best, etc.
- The GA's "strategy" will be (nearly) optimal if the fitnesses of schemas in the population are reasonably uniform distributed (Grefenstette example above for d*…*).

GA: THE EVOLUTION OF 2^K–ARMED BANDIT

- k (from "2^k–armed bandit") is largely believed to increase as the GA proceeds towards more aged generations:

  from low-order schemas to higher and higher-order schemas.

- However, at each generation selection introduces new biases in the way schemas are sampled (see Grefenstette example).
- This is why the *static average fitnesses* of schemas are not necessarily correlated with their *observed average fitness*.

ON-LINE PERFORMANCE

- 2AB strategy is fit for **adaptation**, or equivalently, for **on-line performance**.
- E.g.: real-time control problems.
  - automatic control of machineries
  - learning to navigate in an unknown environment
  - predicting financial markets
- To **optimise** (find the best solution) vs. to **satisfice** (find a good enough solution).
  - approximated solution, but found very quickly.
  - **Satisficing** means **maximizing**, for a given amount of time (samples), the amount of information about the sub-spaces likely to provide good solutions if sampled in the future: **GA**.
  - **Optimising** may need GA (≈global) + Hill climber (≈local).

DECEIVING A GENETIC ALGORITHM (1)

- Bethke (1980): if low-order schemata contain misleading information about higher-order schemata, then the GA will hardly find the optimum.
- Example:
  - call the schema with highest static fitness in a hyperplan “a winner”;
  - suppose schemas with all defined bits ‘1’ are winners, except for 11…1;
  - let 00…0 be a winner (hence, an optimum);
  - this is a “fully deceptive” (Goldberg, 1987) function: lower-order schemata give misleading information as to where the optimum is likely to be found.

DECEIVING A GENETIC ALGORITHM (2)

- There are *degrees of deception* (some schemas give correct information, others don't).
- Bethke used "Walsh transforms" (analogous to Fourier transforms) to design such functions
- Is deception **that** important?
  - no, if GAs are *satisficers* rather than *optimisers;*
  - no, if you look for alternative representations (which one is best for a GA to solve a problem?)
- Grefenstette gives examples of deceptive functions which are either easy or difficult for GA to optimise!
  - so, *deception* is not clearly important!
- There is a need to take into consideration other, more dynamic features of GAs. E.g., biases from selection!

SCHEMA APPROACH: DRAWBACKS

- Explains how convergence happens, **if it happens**.
- How does a non-represented schema appear in a future generation?
- Which new schemas are more likely to be discovered?
- How is the estimate of the average fitness built from the average fitness in one (successive) generation(s) ?
- What about situations where the observed fitness of a schema and its actual fitness *are* ***not*** *correlated*?
- What about the speed of convergence?
- Schemas are fit for the likely building blocks of *one-point crossover*. Other structures deal better with other operators.
- One-step process only; unreallistic assumptions for longer-time predictions.

ROYAL ROAD FUNCTIONS

- Stephanie Forrest, John Holland, Melanie Mitchell – 1992→
- Schema Theorem: positive effects of selection, negative aspects of operators.
- RRF illustrate the *constructive* power of crossover.
- Schema Theorem: suggested Building Blocks
  - short, low order, highly-fit schemata;
  - allow for the presence of intermediate "stepping stones"
    - intermediate order, higher fitness schemas
    - obtained from lower-order schemas
    - which can be combined in even higher-fitness schemas.

ROYAL ROAD R_1

- A function with all those properties.
- R_1 is built from a list of schemas s_i.
- Each schema is given with a coefficient c_i.

$$\mathbf{R_1(x) = \sum_i c_i \cdot \delta_i(x),}$$

- where $\delta_i(x) =$ **if** $(x \in s_i)$ **then** 1 **else** 0.

R_1 has a building-block structure;

the building-block hypothesis should lay out a "royal road"

for the GA to reach the optimal string.

STRUCTURE FOR A ROYAL ROAD FUNCTION

- s_1 = 11111111**********…**********, c_1=8
- s_2 = ********11111111********…**********, c_2=8
- s_3 = **********11111111…**********, c_3=8
- s_4 = **********…**********, c_4=8
- s_5 = **********…**********, c_5=8
- s_6 = **********…**********, c_6=8
- s_7 = **********…11111111********, c_7=8
- s_8 = **********…********11111111, c_8=8
- s_{opt}= 11111111111111111111111…11111111111111.
- In this case, **c_i = order (s_i)**.

A GENETIC ALGORITHM FOR R_1

- A standard GA with the following specific settings:
- *pop_size* = 128; random initialization; p_c=0.7; p_m=0.005.
- σ-selection (σ - the standard deviation of fitnesses in the population).
  - expected number of offspring per individual:

$$1 + (f(x_i) - F / pop\_size) / \sigma$$

  - topping at 1.5 (most often: 0, 1, 2), which
    - slows down the convergence by
    - preventing premature convergence
    - for R_1, some individuals are much better fitted than most individuals.
- The GA has been compared with 3 hill climbers.

PREMATURE CONVERGENCE

- The population converged
  - most chromosomes are copies of one genotype
- but not to a representation of an optimal solution
  - rather, to a local optimum
- It’s not necessarily “in too few generations”
- it’s “to a wrong target”

HILL CLIMBERS (1)

- Steepest ascent hill climbing (**SAHC**).
  1. choose a string at random (*current_hilltop*);
  2. generate all its neighbours at Hamming distance 1;
  3. first fitness-improving neighbour becomes new *current_hilltop* (ties decided at random);
  4. if (no fitness increase) then {save *current_hilltop*; go to 1} else go to 2;
  5. when *max_no_of_function_evaluations* is reached, return highest hilltop found.

HILL CLIMBERS (2)

- Next-ascent hill climbing (**NAHC**). m loci modulo m.

  1. choose a string at random (*current_hilltop*); set j=1;
  2. generate at most m neighbours at Hamming distance 1, starting from locus j;
  3. if (a fitness increase happens at locus k) then

     {set the new *current_hilltop*; j=(k+1)%m; go to 2;}
  4. save *current_hilltop* and go to 1;
  5. when *max_no_of_function_evaluations* is reached, return highest hilltop found.

HILL CLIMBERS (3)

- Random mutation hill climbing (**RMHC**).

1. choose a string at random (*current_hilltop*);
2. choose a locus at random to flip;
3. if (fitness at least that good) then

   set the new *current_hilltop*;
4. go to step 2 until *max_no_of_function _evaluations* is reached;
5. return *current_hilltop*.

EXPERIMENTAL RESULTS

- 200 runs of each algorithm, each time with a different random-number seed. Max_no_evaluations = 256,000.
- An algorithm was allowed to run until optimum was found; the number of function evaluations was recorded.

Number of evaluations over 200 runs	**GA (σ/sqrt(200))**	**SAHC**	**NAHC**	**RMHC (σ/sqrt(200))**
Mean	**61,334 (2,304)**	**⊥ >256,000**	**⊥ >256,000**	**6,179 (186)**
Median	**54208**	**⊥ >256,000**	**⊥ >256,000**	**5,775**

THE PROBLEM

- A landscape designed to provide “royal roads” for the GA to converge

  actually leads to a better performance of a random procedure!

- Under what conditions would a GA outperform other search algorithms (hill climbers)?

ANALYSIS OF RMHC (1)

- Feller 1968 – "An Introduction to Probability Theory and Its Applications"
- Suppose the Royal Road function uses N adjacent blocks of K ones (for R_1: K = 8, N = 8).
- E(K,N) – the expected number of function evaluations before RMHC finds the optimum.
- The time needed to find a second block is larger than the time needed to find the first block (mutations wasted to flip bits inside the first block).
- Non-wasted mutations: $(K \cdot N - K) / K \cdot N$ (outside the first block).
- Useful mutations p% of the time → 100/p times as long to reach the same performance as if no wasted mutations.

ANALYSIS OF RMHC (2)

- $E(K,2) = E(K,1) + E(K,1) \cdot (K \cdot N / (K \cdot N - K))$

  $= E(K,1) \cdot N / (N-1)$
- $E(K,3) = E(K,2) \cdot N / (N-2)$ …
- $E(K,N) = E(K,N-1) \cdot N / (N-(N-1))$
- $E(K,N) = E(K,1) \cdot N \cdot (1+1/2+1/3+\ldots+1/N)$

  (actually, worse than that: E(K,N) depends on worst-time block, not necessarily the first one).
- $E(K,N) \approx E(K,1) \cdot N \cdot (\ln N + \gamma)$

  γ is Euler's constant.

ANALYSIS OF RMHC (3)

- E(K,1) can be studied via a Markov chain analysis (mutation by mutation).
- Result: $E(K,1) \rightarrow 2^K$ (decreasing) as $K \rightarrow \infty$.
- E(8,1) = 301.2
- Overall: $\mathbf{E(K,N) \approx 2^K \cdot N \cdot (\ln N + \gamma)}$
- E(8,8) = 6,549
- Experimental result over 200 runs: 6,179

THE GA: HITCHHICKING

- One reason for the bad performance of the GA compared to RMHC: hitchhiking.
- Once an instance of a good schema is found, zeros in other positions than the discovered block(s) – defined positions of the schema – spread in the population **at high fitnesses.**
- Ones in wild-card positions become more difficult to find, especially those close to the defined positions of the schema (crossover disruption of "hitchhikers" is less likely there).

EFFECTS OF HITCHHICKING (1)

- “Spurious correlation” (Belew 1992, Whitley 1991, Schaffer 1991).
- The implicit parallelism of the GA is limited by restricting, at certain loci, the number of schemas sampled: **non-independent sampling**.
- The effectiveness of crossover is limited by early convergence to **wrong, but highly fitted** schemas.
- The effect exists in real population genetics (and not only there)!

EFFECTS OF HITCHHICKING (2)

- A good schema S not present in the first generation will be disfavored in subsequent generations by:
  - neighbor schemas already present in the population
  - mostly if both neighboring blocks are already found
    - in different schemas or
    - in one schema
- Overlapping should be defined in terms of “aura” rather than “position”.

AN IDEALIZED GENETIC ALGORITHM

- There is no independent sampling in either GA ("biased sampling" – Grefenstette) or RMHC:
  - GA: samples in the s_3 hyperplan were not independent of those in the s_2 or s_4 hyperplans;
  - RMHC: each string differed from the previous one only in one bit.
- Nevertheless, the RMHC explores the search space in a systematic, bit-by-bit manner, never loosing something already found.

STATIC BUILDING BLOCK HYPOTHESIS

If, in the GA

- there could be an independent sampling in each hyperplan and
- the best schema in each hyperplan were selected

then crossover should quickly combine the best schemas in different hyperplans into one string.

- How would an idealized GA, working according to the SBBH, look like (Mitchell, Holland, Forrest, 1994)?

AN IDEALIZED GENETIC ALGORITHM

- IGA does not have a population – one string at a time
- Input: desired schemas (*idealized*…).

repeat {**generate** strings at random}**until** (the generated string contains at least one desired schema) ;

store that string;

While (not halting condition) **do** {

generate strings at random (uniform probability);

whenever a string containing one or more *yet-undiscovered schemas* is found, **crossover** it with the stored string;

replace the stored string by the offspring containing all desired schemas discovered so far;
}

FEATURES OF IDA

- Strings are chosen completely independently, so schemas (hyperplans) are sampled independently
- Selection is deterministic and driven by the memory of desired / found schemas.
- Crossover is used.
- **Actually, desired schemas are not known beforehand.**
- However, IDA gives a **lower bound** on the time (number of function evaluations) needed by any GA to find the optimal string for R_1.

ANALYSIS OF IDA (1)

- Again, N blocks of K ones each.
- N=1 : one desired schema, H.
- p: the probability of finding an instance of H at random. $p = 1/2^K$.
- $q = 1 - p$.
- $p_1(t)$: the probability to find H before time t.
- $p_1(t) = 1 - q^t$
- $p_N(t) = (1 - q^t)^N$ (all N schemas found by time t)

ANALYSIS OF IDA (2)

- $p_N(t)$ is the probability to find all N schemas at time t at the latest.
- We are interested in the **expected time** to find all N schemas, E_N.
- Let $\boldsymbol{P}_N(t)$ be the probability to find them all **at** time N.
- $\boldsymbol{P}_N(t) = \boldsymbol{P}_N(t) - \boldsymbol{P}_N(t-1) = (1 - q^t)^N - (1 - q^{t-1})^N$

$$E_N = \sum_{t=1}^{\infty} t \cdot \boldsymbol{P}_N(t) = \sum_{t=1}^{\infty} t \cdot ((1-q^t)^N - (1-q^{t-1})^N)$$

ANALYSIS OF IDA (3)

$$((1-q^t)^N-(1-q^{t-1})^N)=$$

$$=[C_N^1(1/q-1)q^t]-[C_N^2(1/q^2-1)q^{2t}]+\ldots+(-1)^{N-1}[C_N^N(1/q^N-1)q^{Nt}]$$

- In order to sum (1..∞) such expressions multiplied by *t*, we can consider the infinite sum of first terms, the infinite sum of second terms etc.:

$$C_N^1(1/q-1)\sum_{t=1}^{\infty} t\cdot q^t = C_N^1(1/q-1)\cdot q(q+2q^2+3q^3+\ldots)=$$

$$C_N^1(1/q-1)\cdot q\cdot\frac{d}{dq}(q+q^2+\ldots)=C_N^1(1/q-1)\cdot q\cdot\frac{d}{dq}\left(\frac{q}{1-q}\right)=C_N^1\frac{1}{1-q}$$

- The sum over the nth term is $C_N^n\dfrac{1}{1-q^n}$.

ANALYSIS OF IDA (4)

- Substituting 1-*p* for *q* and assuming that K is large enough to give $q^n = (1-p)^n \approx 1 - n \cdot p$, we get:

$$E_n \approx \frac{1}{p} \cdot \left(\frac{C_N^1}{1} - \frac{C_N^2}{2} + \frac{C_N^3}{3} - \ldots + (-1)^{n-1} \frac{C_N^N}{N} \right)$$

- For N=8, K=8, we obtain $E_N \approx 696$
- The experiments of Mitchell e.a. gave exactly this value over 200 runs of the IDA (σ=19.7).

ANALYSIS OF IDA VS. RMHC

- Binomial theorem + integration of $(1+x)^N$ give :

$$\sum_{n=1}^{N} C_N^n \cdot \frac{x^n}{n} = \sum_{n=1}^{N} \frac{1}{n} \cdot ((1+x)^{n-1} - 1)$$

- This leads to:

$$E_N \approx -\frac{1}{p} \sum_{n=1}^{N} C_N^n \cdot \frac{(-1)^n}{n} = \frac{1}{p} \sum_{n=1}^{N} \frac{1}{n} \approx \frac{1}{p} \cdot (\ln N + \gamma)$$

- Hence, for the GA: $\mathbf{E_N = O\,(2^K \cdot \ln(N))}$
- For RMHC : $\mathbf{E(K,N) = O\,(2^K \cdot N \cdot \ln(N))}$

WHAT MAKES IDA N TIMES FASTER THAN RMHC ?

- **Implicit parallelism:**
  - IDA perfectly implements it: new samples are given independently to each schema region;
  - RMHC: each new string gives a new sample to only one schema region.
- Independent sampling :
  - allows multiple schemas to be found in a new string
  - avoids wasted samples (mutations of correct bits).
- For IDA, the same comparison is valid against any HC method based on flipping single or few bits.

HOW MUCH OF IDA CAN BE BUILT IN AN ACTUAL GA ?

1. **Independent sampling:**
   1. the population size has to be large enough;
   2. mutation rate sufficiently large
   3. selection pressure lowered (the relative fitness of non-overlapping desirable schemas has to be small enough)
2. **Storing desired schemas**
   1. however, selection pressure high enough for desirable schemas to *survive*
3. **Instantaneous crossover**
   1. crossover rate has to be high enough to allow for *quick recombination* of two desired schemas
4. **Speedup over RMHC**
   1. the length of strings has to be large enough to have a significant *factor* **N**.

- How to balance non-compatible features (1.2 vs.2., 1.3 vs. 2.1)?

IMPLEMENTATION OF GENETIC ALGORITHMS

- Huge number of choices.
- Little theoretical guidance as to which ones are better in specific cases.
- Bitstring representation, fitness proportionate selection, simple operators may not be (and are not!) **the** choice in every particular case.
- There are as many GAs as there are GA projects

WHEN SHOULD A GA BE USED?

- Many successes, but there are failures too.
- A comparatively good GA is likely to exist if:

1. the search space is large (otherwise: exhaustive search);
2. the function to optimise is
   - "non-standard" and / or noisy (otherwise: a one-candidate-at-a-time approach – for example, Hillclimbing)
   - multi-modal (otherwise: gradient-ascent)
   - less understood (otherwise: heuristics – see the TSP)
3. a sufficiently good solution, not necessary the global optimum, is required (otherwise: exact algorithms).

DATA STRUCTURES (1) BINARY ENCODINGS

- The representation of candidate solutions is believed to be **the** central factor of success / failure of a GA.
- Widespread option: fixed length, fixed order bitstrings.
- Advantages:
  - more schemas for the same amount of information
  - better developed theory and parameter-setting techniques (e.g., operator rates).
- Disadvantages: unnatural, uneasy to use for many problems (e.g.: evolving rules; optimising weights for artificial neural networks).

DATA STRUCTURES (2)

- *Diploid* encodings (recessive genes).
- *Many-character* and *real-valued* encodings are more natural and easy to handle. For some problems, better performance with larger encoding alphabets.
- *Tree encodings* led to Genetic Programming (Koza). The search space is open-ended: any size.
- *Multi-dimensional* encodings (clustering).
- *Non-homogenous* encodings (e.g., instructions to build a solution – timetable).
- Sometimes, decodings performed with problem-specific heuristics (clustering).
- Davis (1991): choose the natural encoding, then devise the GA!

EVOLUTION OF ENCODING

- If less understood problems are fitted for using GAs, how could one know the "natural" encoding ahead of time?
- *The linkage problem*: which are the important loci for useful schemata? (in order to prevent them from being disrupted by crossover)
- Why not adapting the encoding itself?

ENCODING-ADAPTING TECHNIQUES

- Adaptation via **length evolution** (with variable length chromosomes)
- Inversion (representation-adapting operator)
- Identifying crossover “hot spots” (where to cut for best-fitted offspring?)
- Messy Genetic Algorithms (incomplete / contradictory representations).

ANOTHER VIEW ON ADAPTING THE ENCODING (1)

- **Deception**: GA misled by short, low-order schemata.
  - good schemas (observed fitness, order, defining length) which are however bad from the viewpoint of static fitness.
- Important for understanding the way GAs work
- Not that important at the hands-on level
  - all known examples have been designed on purpose
  - starting from a "royal road" for deception to occur
  - and NOT from mathematical objects which are actual candidate solutions for a given problem
  - and for which there always exist ***alternative representations*** (better than the deceptive one).

ANOTHER VIEW ON ADAPTING THE ENCODING (2)

- Approaches to deception:
  - Alternative representations
    - only artificial examples lack alternative representations.
  - Inversion
    - an operator focused against certain deceptive situations.
  - Messy Genetic Algorithms
    - a Genetic Algorithm with a different approach for representation and a modified procedural scheme.

INVERSION (1)

- Holland (1975).
- An operator to handle the linkage problem in fixed-length strings.
- It is a reordering operator inspired from real genetics.
- The interpretation of an allele does not depend on its position.

INVERSION (2)

- Acts for order similar to mutation for bits
- (00011010001) →

  ((1,0) (2,0) (3,0) (4,1) (5,1) (6,0) (7,1) (8,0) (9,0) (10,0) (11,1))
- If *inversion points* are randomly generated to be after loci 3 and 7, then after inversion the chromosome becomes:

  ((1,0) (2,0) (3,0)|(7,1) (6,0) (5,1) (4,1)|(8,0) (9,0) (10,0) (11,1))
- Schemas like (10*********01) can be preserved after crossover, if successful inversion finds the building block ((1,1) (2,0) (13,1) (12,0) (11,*)…(3,*))

INVERSION (3)

- Main drawback: crossing over two parents may result in meaningless offspring.
- This is because permutations are involved.
- Solution: master/slave approach.
  - one parent is chosen to be the master;
  - the other one is temporarily reordered to the same permutation as the master.
- Few good results, no proof (either by systematic experiments or by theoretical results) that inversion is beneficial.

CO-EVOLVING CROSSOVER "HOT SPOTS"

- Schaffer, Morishima: a dual approach to inversion.
- Find places where crossover is allowed to occur.
- In each chromosome the croosover points are marked (say, with "!"):

(1 0 0 1! 1 1 1! 1) → (1 0 0 1! 0 0! 1! 0)

(0 0 0 0 0 0! 0 0) → (0 0 0 0 1 1 0 1)

- Mutation may change 0's and 1's, but also it may erase a "!" or insert a new "!".
- Evaluation does not take into consideration the !'s.

MESSY GENETIC ALGORITHMS

- Goldberg, Deb – 1989.
- Human genome: did not start with strings of length $5.9*10^9$, but rather from simple forms.
- **Representation**:
  - bits tagged with locus
  - underspecification (schemata);
  - overspecification (diploidy);
  - {(1,0),(2,0),(4,1),(4,0)} → 00*1
  - {(3,1),(3,0),(3,1),(4,0),(4,1),(3,1)} → **10

THE MESSY GENETIC ALGORITHM (2)

- **Evaluation:**
  - **overspecified genes: left to right;**
  - **underspecified genes:**
    - **estimation of static fitness (randomly generated chromosomes representing the scheme);**
    - **template: local optimum found before running the GA**
- **Phases:**
  - **primordial phase (exploration);**
  - **juxtapositional phase (exploitation).**

THE MESSY GENETIC ALGORITHM (3)

- Primordial phase:
  - guess k – the shortest useful schemas order;
  - enumerate all these schemas.
  - for k=3, m=8, this enumeration is:
    {(1,0),(2,0),(3,0)}; {(1,0),(2,0),(3,1)};…
    {(1,1),(2,1),(3,1)}; {(1,0),(2,0),(4,0)};
    {(1,0),(2,0),(4,1)}; … {(6,1),(7,1),(8,1)}.
  - apply selection:
    - copies in proportion to fitness;
    - delete half of the population at regular intervals.

THE MESSY GENETIC ALGORITHM (4)

- Juxtapositional phase:
  - fixed population size;
  - selection continues;
  - two operators:
    - **cut** : one messy chromosome is cut to give birth to two messy chromosomes
    - **splice** : two messy chromosomes are spliced into one.
- Test function: m=30; eval(chr) = $\sum \text{eval}(b_{3i+1} b_{3i+2} b_{3(i+1)})$
- 000→28; 001→26; 010→22; 011→0; 100→14; 101→0; 110→0; 111→30.

THE MESSY GENETIC ALGORITHM (5)

- Problems:
  - k must be guessed – no a priori knowledge;
  - combinatorial explosion: the number of schemas to be generated is $2^k C_m^k$.
  - population size grows exponentially with k (is k always small?)
- "probabilistically complete intitialization": initial chromosomes of length between k and m (implicit parallelism helps).

SAMPLING MECHANISM

1. Fitness-proportionate with “roulette wheel”
2. Stochastic Universal Sampling
3. Sigma Scaling
4. Elitism
5. Boltzmann Selection
6. Rank-based selection
7. Tournament Selection
8. Steady-State selection

SOCHASTIC UNIVERSAL SAMPLING (1)

- “Roulette Wheel” selection does not actually result in the expected number of copies
  - relatively small populations
  - there is a non-zero probability to have all offspring alocated to the worse individual!
- Baker (1987): spin the roulette only once, not *pop_size* times, but with *pop_size* equally spaced pointers on it.

STOCHASTIC UNIVERSAL SAMPLING (2)

```
ptr = Rand();
for (sum = i = 0; i < pop_size; i++)
  for (sum += Expected_Value(i,t); sum > ptr; ptr++)
      Select(i);
```

- Each individual i is guaranteed to be selected, at generation t:

  at least $\lfloor$Expected_Value(i,t)$\rfloor$ times and

  at most $\lceil$Expected_Value(i,t)$\rceil$ times.

SIGMA SELECTION

- The rate of evolution depends on the variance of fitnesses in the population. Can it be smoothed?
  - too large variances lead to premature convergence
  - small variances lead to a nearly random algorithm
- Forrest (1985): making variance less influential.
- E_V(i,t) = **if** ($\sigma(t) \neq 0$)

  then $(1+(f(i)-f\_med(t))/2\sigma(t))$ **else** 1.0
- Expected_Value(i,t) =

  if (E_V(i,t)>=0) **then** E_V(i,t) **else** const1

  (e.g., const1 = 0.1).

ELITISM

- Ken DeJong (1975).
- A possible addition to any selection mechanism.
- Retain some number k of the best individuals at each generation.
- k is a parameter of the algorithm.
- Often, it significantly improves the GA performance.

BOLTZMANN SELECTION

- Unlike what happens under Sigma scaling, one does not need a constant selection pressure over the run of a GA.
- Rather, different rates are needed at different moments.
- Boltzmann selection: a continuously varying **temperature T** controls the selection pressure.
- Starting temperature – **high** → selection pressure **low**.
- Subsequently temperature lowers → selection pressure increases
- <u>Typical implementation</u>: each individual is assigned an expected value: $\mathbf{Exp\_Val(i,t) = e^{f(i)/T} / «e^{f(i)/T}»_t}$,

  where $«\cdot»_t$ means "average over generation t".
- As T decreases, differences in Exp_Val increase *(experimental exercise)*.

RANK SELECTION

- Purpose: to prevent premature convergence.
- Baker (1985):
  - individuals ranked according to fitness
  - expected values depend on rank, not on fitnesses
- No fitness-scaling is necessary.
- Rank selection **decreases the selection pressure** if the fitness variance is high; **the opposite** happens if the fitness variance is low.

RANK SELECTION: LINEAR RANKING

Rank individuals (increasing order of fitness):

1, 2, …, *pop_size;*

Choose the expected value $Max \geq 0$ for rank *pop_size*;

Set $Min = 2 - Max$ (expected value for rank 1);

Set: Exp_Val(i,t) =

$$= Min + (Max - Min)(\text{rank}(i,t) - 1)/(pop\_size - 1)$$

- **Exercise**: given the restriction that, for fixed t, $\sum \text{Exp\_Val}(i,t) = N$, prove that $1 \leq Max \leq 2$ and that *Min* has to be equal to 2-*Max*.

RANK SELECTION: A ROULETTE WHEEL

$$\begin{cases} \text{choose } q; \\ \text{for each rank } i\text{, set the probability to select} \\ \quad \text{chromosome } i \text{ prob}(i)=q(1-q)^{i-1}. \end{cases}$$

- i=1 → best chromosome.
- Example. *pop_size*=50, q=0.04:

prob(1)=0.04; prob(2)=0.0384; prob(3)=0.036864; etc.

$$\sum_{i=1}^{pop\_size} \text{prob}(i) = \sum_{i=1}^{pop\_size} q \cdot (1-q)^i \approx 1.$$

TOURNAMENT SELECTION

- Similar to rank selection in terms of selection pressure.

  Two individuals are chosen at random from the population;

  A random number $r \in [0;1]$ is generated;

  if ($r < k$) **then** (the fitter of the two individuals is selected) **else** (the less fit is selected);

  The two are returned to the sampled population

- k is a parameter (e.g., k=0.9).
- Deb and Goldberg analysed this selection mechanism (1991)

STEADY STATE SELECTION

- **Generational GAs**: new generation consists only of offspring.
- *No*, *few* or *more* parents may survive unchanged.
- **Generational gap**: the fraction of new individuals in the new generation (DeJong).
- Steady state selection: **only a few individuals are replaced in each generation**.
- Several of the least fit individuals are replaced by offspring of the fittest ones.
- Useful in evolving rule-based systems (classifier systems – Holland 1986)
- Analysed by DeJong and Sarma (1993).

SELECTION MECHANISMS – A BRIEF COMPARISON

- Fitness proportionate selection mechanism is used traditionally (as Holland's original proposal and because of its use in the schema theorem).
- Alternative selection mechanisms have been shown to improve convergence in many cases.
- Fitness-proportionate methods require two passes through each generation (one for mean fitness, one for expected values);
- Rank selection requires sorting the population – time consuming.
- Tournament selection is computationally more efficient and amenable to parallelisation.

SELECTION MECHANISM: IDEAS FOR TAXONOMY

- Dynamics of the field of probabilities:
  - *dynamic selections* (expected value of any chromosome varies over generations: classical);
  - *static selections* (fixed expected values: ranking).
- Survival probabilities:
  - extinctive – survival probabilities may be zero
    - left-extinctive – best chromosomes ← 0;
    - right-extinctive – worst chromosomes ← 0.
  - non-extinctive – all survival probabilities non-zero.
- Elitist / non-elitist

THE ISLAND MODEL

- The population is made of *subsets of chromosomes*.
- These subsets evoluate separately (selection and operators applied only inside each subset of the population).
- At times, subsets exchange chromosomes with a certain probability.
- Advantage: more evolution histories in one run.