DM811

Heuristics for Combinatorial Optimization

Lecture 9

Stochastic Local Search and Metaheuristics

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Min-Conflict Heuristic

(Already encountered)

procedure *MCH* (*P*, *maxSteps*)
  input: *CSP instance P, positive integer maxSteps*
  output: *solution of P or* "no solution found"
  a := randomly chosen assignment of the variables in P;
  for *step* := 1 **to** *maxSteps* **do**
    if a satisfies all constraints of P **then return** a **end**
    x := randomly selected variable from conflict set $K(a)$;
    v := randomly selected value from the domain of x such that
      setting x to v minimises the number of unsatisfied constraints;
    a := a with x set to v;
  end
  return "no solution found"
end *MCH*

Min-Conflict Heuristic
In Comet


```
import cotls;
int n = 16;
range Size = 1..n;
UniformDistribution distr(Size);

Solver<LS> m();
var{int} queen[Size](m,Size) := distr.get();
ConstraintSystem<LS> S(m);

S.post(alldifferent(queen));
S.post(alldifferent(all(i in Size) queen[i] + i));
S.post(alldifferent(all(i in Size) queen[i] - i));
m.close();

int it = 0;
while (S.violations() > 0 && it < 50 * n) {
  select(q in Size : S.violations(queen[q])>0) {
    selectMin(v in Size)(S.getAssignDelta(queen[q],v)) {
      queen[q] := v;
      cout<<"change: queen["<<q<<"] := "<<v<<" viol: "<<S.violations() <<
           endl;
    }
    it = it + 1;
  }
}
```


Randomized Iterative Impr.
aka, Stochastic Hill Climbing

Key idea: In each search step, with a fixed probability perform an uninformed random walk step instead of an iterative improvement step.

Randomized Iterative Improvement (RII):
determine initial candidate solution s
while termination condition is not satisfied **do**
  With probability $\texttt{wp}$:
    choose a neighbor s' of s uniformly at random
  Otherwise:
    choose a neighbor s' of s such that $f(s') < f(s)$ or,
      if no such s' exists, choose s' such that $f(s')$ is minimal
  $s := s'$

Example: Randomized Iterative Improvement for GCP

procedure *RIIGCP*(F, $\texttt{wp}$, $\texttt{maxSteps}$)
  input: *a graph* G *and* k, *probability* $\texttt{wp}$, *integer* $\texttt{maxSteps}$
  output: *a proper coloring* φ *for* G **or** $\emptyset$
  choose coloring φ of G uniformly at random;
  steps := 0;
  while not(φ is not proper) **and** (*steps* $<$ *maxSteps*) **do**
    with probability $\texttt{wp}$ **do**
      select v in V and c in Γ uniformly at random;
    otherwise
      select v in V^c and c in Γ uniformly at random from those that
        maximally decrease number of edge violations;
    change color of v in φ;
    steps := *steps*+1;
  end
  if φ is proper for G **then return** φ
  else return $\emptyset$
  end
end *RIIGCP*

Note:

- No need to terminate search when local minimum is encountered

  Instead: Impose limit on number of search steps or CPU time, from beginning of search or after last improvement.
- Probabilistic mechanism permits arbitrary long sequences of random walk steps

  Therefore: When run sufficiently long, RII is guaranteed to find (optimal) solution to any problem instance with arbitrarily high probability.

Min-Conflict + Random Walk


```
procedure WalkSAT (F, maxTries, maxSteps, slc)
    input: CNF formula F, positive integers maxTries and maxSteps,
        heuristic function slc
    output: model of F or 'no solution found'
    for try := 1 to maxTries do
        a := randomly chosen assignment of the variables in formula F;
        for step := 1 to maxSteps do
            if a satisfies F then return a end
            c := randomly selected clause unsatisfied under a;
            x := variable selected from c according to heuristic function slc;
            a := a with x flipped;
        end
    end
    return 'no solution found'
end WalkSAT
```

Example of slc heuristic: with prob. wp select a random move, with prob. $1 - wp$ select the best

Tabu Search

Key idea: Use aspects of search history (memory) to escape from local minima.

- Associate tabu attributes with candidate solutions or solution components.
- Forbid steps to search positions recently visited by underlying iterative best improvement procedure based on tabu attributes.

Tabu Search (TS):

determine initial candidate solution s
While *termination criterion* is not satisfied:
    determine set N' of non-tabu neighbors of s
    choose a best candidate solution s' in N'

    update tabu attributes based on s'
    $s := s'$

Example: Tabu Search for GCP – TabuCol

- **Search space:** set of all complete colorings of G.
- **Solution set:** proper colorings of G.
- **Neighborhood relation:** one-exchange.
- **Memory:** Associate tabu status (Boolean value) with each pair (v, c).
- **Initialization:** a construction heuristic
- **Search steps:**
  - pairs (v, c) are tabu if they have been changed in the last `tt` steps;
  - neighboring colorings are admissible if they can be reached by changing a non-tabu pair or have fewer unsatisfied edge constr. than the best coloring seen so far (*aspiration criterion*);
  - choose uniformly at random admissible coloring with minimal number of unsatisfied constraints.
- **Termination:** upon finding a proper coloring for G *or* after given bound on number of search steps has been reached *or* after a number of idle iterations

Note:

- Non-tabu search positions in $N(s)$ are called admissible neighbors of s.
- After a search step, the current search position or the solution components just added/removed from it are declared tabu for a fixed number of subsequent search steps (tabu tenure).
- Often, an additional aspiration criterion is used: this specifies conditions under which tabu status may be overridden (*e.g.*, if considered step leads to improvement in incumbent solution).
- Crucial for efficient implementation:
  - keep time complexity of search steps minimal by using special data structures, incremental updating and caching mechanism for evaluation function values;
  - efficient determination of tabu status: store for each variable x the number of the search step when its value was last changed it_x; x is tabu if $it - it_x < tt$, where it = current search step number.

Note: Performance of Tabu Search depends crucially on setting of tabu tenure `tt`:

- `tt` too low $\Rightarrow$ search stagnates due to inability to escape from local minima;
- `tt` too high $\Rightarrow$ search becomes ineffective due to overly restricted search path (admissible neighborhoods too small)

Advanced TS methods:

- **Robust Tabu Search** [Taillard, 1991]:
  repeatedly choose `tt` from given interval;
  also: force specific steps that have not been made for a long time.
- **Reactive Tabu Search** [Battiti and Tecchiolli, 1994]:
  dynamically adjust `tt` during search;
  also: use escape mechanism to overcome stagnation.

Metaheuristics
Population Based Metahe

Further improvements can be achieved by using *intermediate-term* or *long-term memory* to achieve additional *intensification* or *diversification*.

Examples:

- Occasionally backtrack to *elite candidate solutions*, *i.e.*, high-quality search positions encountered earlier in the search; when doing this, all associated tabu attributes are cleared.
- Freeze certain solution components and keep them fixed for long periods of the search.
- Occasionally force rarely used solution components to be introduced into current candidate solution.
- Extend evaluation function to capture frequency of use of candidate solutions or solution components.

Metaheuristics
Population Based Metahe

Tabu search algorithms algorithms are state of the art for solving many combinatorial problems, including:

- SAT and MAX-SAT
- CSP and MAX-CSP
- GCP
- many scheduling problems

Crucial factors in many applications:

- choice of neighborhood relation
- efficient evaluation of candidate solutions (caching and incremental updating mechanisms)

Min-Conflict + Tabu Search

Metaheuristics
Population Based Metahe

- After the value of a variable x is changed from v to v' with min-conflict heuristic, the variable/value pair (x_i, v) is declared tabu for the next tt steps
- $\mathtt{tt} = 2$ is often a good choice

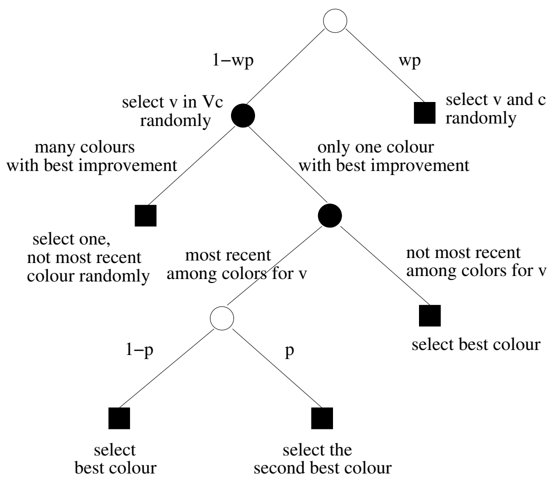
➨ Advantage: the neighborhood does not need to be searched exahustively

Min-Conflict + RW + TS

Another more involved hybrid:

Example on GCP

: decision tree for `step`



TS for GCP

Design choices:

- Neighborhood exploration:
  - no reduction
  - min-conflict heuristic
- Prohibition power for `move = <v,new_c,old_c>`
  - `<v,-,->`
  - `<v,-,old_c>`
  - `<v,new_c,old_c>`, `<v,old_c,new_c>`
- Tabu list dynamics:
  - Interval: $\texttt{tt} \in [t_b, t_b + w]$
  - Adaptive: $\texttt{tt} = \lfloor \alpha \cdot c_s \rfloor + \texttt{RandU}(0, t_b)$

Probabilistic Iterative Improv.

Key idea: Accept worsening steps with probability that depends on respective deterioration in evaluation function value: bigger deterioration $\cong$ smaller probability

Realization:

- Function $p(f, s)$: determines probability distribution over neighbors of s based on their values under evaluation function f.
- Let $\texttt{step}(s, s') := p(f, s, s')$.

Note:

- Behavior of PII crucially depends on choice of p.
- II and RII are special cases of PII.

Example: Metropolis PII for the TSP

- **Search space S:** set of all Hamiltonian cycles in given graph G.
- **Solution set:** same as S
- **Neighborhood relation $\mathcal{N}(s)$:** 2-edge-exchange
- **Initialization:** an Hamiltonian cycle uniformly at random.
- **Step function:** implemented as 2-stage process:
  1. select neighbor $s' \in N(s)$ uniformly at random;
  2. accept as new search position with probability:

$$p(T, s, s') := \begin{cases} 1 & \text{if } f(s') \leq f(s) \\ \exp \frac{f(s) - f(s')}{T} & \text{otherwise} \end{cases}$$

  (Metropolis condition), where *temperature* parameter T controls likelihood of accepting worsening steps.
- **Termination:** upon exceeding given bound on run-time.

Inspired by statistical mechanics in matter physics:

- candidate solutions $\cong$ states of physical system
- evaluation function $\cong$ thermodynamic energy
- globally optimal solutions $\cong$ ground states
- parameter $T \cong$ physical temperature

Note: In physical process (*e.g.*, annealing of metals), perfect ground states are achieved by very slow lowering of temperature.

Simulated Annealing

Key idea: Vary temperature parameter, *i.e.*, probability of accepting worsening moves, in Probabilistic Iterative Improvement according to annealing schedule (aka *cooling schedule*).

Simulated Annealing (SA):

determine initial candidate solution s
set initial temperature T according to annealing schedule
while termination condition is not satisfied: **do**
  while maintain same temperature T according to annealing schedule **do**
    probabilistically choose a neighbor s' of s using proposal mechanism
    if s' satisfies probabilistic acceptance criterion (depending on T) **then**
      $s := s'$
  update T according to annealing schedule

- 2-stage step function based on
  - proposal mechanism (often uniform random choice from $N(s)$)
  - acceptance criterion (often *Metropolis condition*)
- Annealing schedule
  (function mapping run-time t onto temperature $T(t)$):
  - initial temperature T_0
    (may depend on properties of given problem instance)
  - temperature update scheme
    (*e.g.*, linear cooling: $T_{i+1} = T_0(1 - i/I_{max})$,
    geometric cooling: $T_{i+1} = \alpha \cdot T_i$)
  - number of search steps to be performed at each temperature
    (often multiple of neighborhood size)
  - may be *static* or *dynamic*
  - seek to balance moderate execution time with asymptotic behavior properties
- Termination predicate: often based on *acceptance ratio*,
  i.e., ratio of proposed *vs* accepted steps *or* number of idle iterations

Example: Simulated Annealing for the TSP

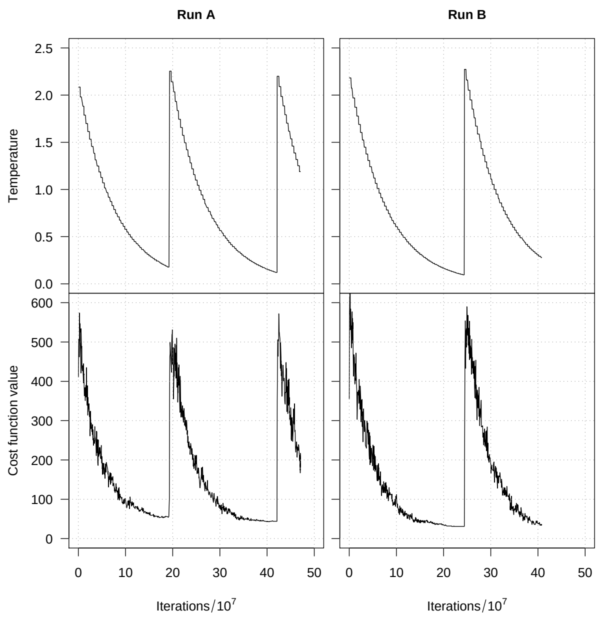
Extension of previous PII algorithm for the TSP, with

- *proposal mechanism:* uniform random choice from 2-exchange neighborhood;
- *acceptance criterion:* Metropolis condition (always accept improving steps, accept worsening steps with probability $\exp[(f(s) - f(s'))/T]$);
- *annealing schedule*: geometric cooling $T := 0.95 \cdot T$ with $n \cdot (n-1)$ steps at each temperature (n = number of vertices in given graph), T_0 chosen such that 97% of proposed steps are accepted;
- *termination:* when for five successive temperature values no improvement in solution quality and acceptance ratio $< 2\%$.

Improvements:

- neighborhood pruning (*e.g.*, candidate lists for TSP)
- greedy initialization (*e.g.*, by using NNH for the TSP)
- *low temperature starts* (to prevent good initial candidate solutions from being too easily destroyed by worsening steps)

Profiling



Iterated Local Search

Key Idea: Use two types of LS steps:

- *subsidiary local search* steps for reaching local optima as efficiently as possible (intensification)
- perturbation steps for effectively escaping from local optima (diversification).

Also: Use acceptance criterion to control diversification *vs* intensification behavior.

Iterated Local Search (ILS):
determine initial candidate solution s
perform subsidiary local search on s
while termination criterion is not satisfied **do**
  $r := s$
  perform perturbation on s
  perform subsidiary local search on s
  based on acceptance criterion,
    keep s or revert to $s := r$

Note:

- *Subsidiary local search* results in a local minimum.
- ILS trajectories can be seen as walks in the space of local minima of the given evaluation function.
- *Perturbation phase* and *acceptance criterion* may use aspects of *search history* (*i.e.*, limited memory).
- In a high-performance ILS algorithm, *subsidiary local search*, *perturbation mechanism* and *acceptance criterion* need to complement each other well.

Subsidiary local search:

- More effective subsidiary local search procedures lead to better ILS performance.
  Example: 2-opt *vs* 3-opt *vs* LK for TSP.
- Often, subsidiary local search = iterative improvement, but more sophisticated LS methods can be used. (*e.g.*, Tabu Search).

Perturbation mechanism:

- Needs to be chosen such that its effect *cannot* be easily undone by subsequent local search phase.
  (Often achieved by search steps larger neighborhood.)
  Example: local search = 3-opt, perturbation = 4-exchange steps in ILS for TSP.
- A perturbation phase may consist of one or more perturbation steps.
- Weak perturbation ⇒ short subsequent local search phase; but: risk of revisiting current local minimum.
- Strong perturbation ⇒ more effective escape from local minima; but: may have similar drawbacks as random restart.
- Advanced ILS algorithms may change nature and/or strength of perturbation adaptively during search.

Acceptance criteria:

- Always accept the best of the two candidate solutions

  ⇒ ILS performs Iterative Improvement in the space of local optima reached by subsidiary local search.

- Always accept the most recent of the two candidate solutions

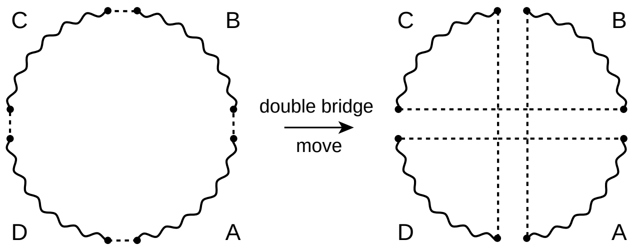
  ⇒ ILS performs random walk in the space of local optima reached by subsidiary local search.

- Intermediate behavior: select between the two candidate solutions based on the *Metropolis criterion* (*e.g.*, used in *Large Step Markov Chains* [Martin et al., 1991].

- Advanced acceptance criteria take into account search history, *e.g.*, by occasionally reverting to *incumbent solution*.

Example: Iterated Local Search for the TSP (1)

- **Given:** TSP instance G.
- **Search space:** Hamiltonian cycles in G.
- **Subsidiary local search:** Lin-Kernighan variable depth search algorithm
- **Perturbation mechanism:**
  'double-bridge move' = particular 4-exchange step:



- **Acceptance criterion:** Always return the best of the two given candidate round trips.

Variable Neighborhood Search

Variable Neighborhood Search is a method based on the systematic change of the neighborhood during the search.

Central observations

- a local minimum w.r.t. one neighborhood function is not necessarily locally minimal w.r.t. another neighborhood function
- a global optimum is locally optimal w.r.t. all neighborhood functions

- Principle: change the neighborhood during the search
- Several adaptations of this central principle
  - (Basic) Variable Neighborhood Descent (VND)
  - Variable Neighborhood Search (VNS)
  - Reduced Variable Neighborhood Search (RVNS)
  - Variable Neighborhood Decomposition Search (VNDS)
  - Skewed Variable Neighborhood Search (SVNS)
- Notation
  - $\mathcal{N}_k$, $k = 1, 2, \ldots, k_m$ is a set of neighborhood functions
  - $N_k(s)$ is the set of solutions in the k-th neighborhood of s

How to generate the various neighborhood functions?

- for many problems different neighborhood functions (local searches) exist / are in use
- change parameters of existing local search algorithms
- use k-exchange neighborhoods; these can be naturally extended
- many neighborhood functions are associated with distance measures; in this case increase the distance

Basic Variable Neighborhood Descent

Procedure BVND
input : $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$, and an initial solution s
output: a local optimum s for $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
  $s' \leftarrow$ FindBestNeighbor(s,$\mathcal{N}_k$)
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
    $(k \leftarrow 1)$
  else
    $k \leftarrow k + 1$
until $k = k_{max}$;

Variable Neighborhood Descent

Procedure VND
input : $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$, and an initial solution s
output: a local optimum s for $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
  $s' \leftarrow$ IterativeImprovement(s,$\mathcal{N}_k$)
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
until $k = k_{max}$;

- Final solution is locally optimal w.r.t. all neighborhoods
- First improvement may be applied instead of best improvement
- Typically, order neighborhoods from smallest to largest
- If iterative improvement algorithms II_k, $k = 1, \ldots, k_{max}$ are available as black-box procedures:
  - order black-boxes
  - apply them in the given order
  - possibly iterate starting from the first one
  - order chosen by: *solution quality* and *speed*

Example

VND for single-machine total weighted tardiness problem

- Candidate solutions are permutations of job indexes
- Two neighborhoods: swap and insert
- Influence of different starting heuristics also considered

initial	swap		insert		swap+insert		insert+swap	
solution	Δavg	$t$avg	Δavg	$t$avg	Δavg	$t$avg	Δavg	$t$avg
EDD	0.62	0.140	1.19	0.64	0.24	0.20	0.47	0.67
MDD	0.65	0.078	1.31	0.77	0.40	0.14	0.44	0.79

Δavg deviation from best-known solutions, averaged over 100 instances

Basic Variable Neighborhood Search


```
Procedure BVNS
input  : N_k, k = 1,2,...,k_max, and an initial solution s
output: a local optimum s for N_k, k = 1,2,...,k_max
repeat
    k ← 1
    repeat
        s' ← RandomPicking(s,N_k)
        s'' ← IterativeImprovement(s',N_k)
        if f(s'') < f(s) then
            s ← s''
            k ← 1
        else
            k ← k + 1
    until k = k_max ;
until Termination Condition ;
```


To decide:

- which neighborhoods
- how many
- which order
- which change strategy

- Extended version: parameters k_{min} and k_{step}; set $k \leftarrow k_{min}$ and increase by k_{step} if no better solution is found (achieves diversification)

Extensions (1)

Reduced Variable Neighborhood Search (RVNS)

- same as VNS except that no IterativeImprovement procedure is applied
- only explores different neighborhoods randomly
- can be faster than standard local search algorithms for reaching good quality solutions

Extensions (2)

Variable Neighborhood Decomposition Search (VNDS)

- same as in VNS but in IterativeImprovement all solution components are kept fixed except k randomly chosen
- IterativeImprovement is applied on the k unfixed components

- IterativeImprovement can be substituted by exhaustive search up to a maximum size b (parameter) of the problem

Extensions (3)

Skewed Variable Neighborhood Search (SVNS)

- Derived from VNS
- Accept $s \leftarrow s''$ when s'' is worse
  - according to some probability
  - skewed VNS: accept if

$$g(s'') - \alpha \cdot d(s, s'') < g(s)$$

$d(s, s'')$ measure the distance between solutions
(underlying idea: avoiding degeneration to multi-start)

Guided Local Search

- **Key Idea:** Modify the evaluation function whenever a local optimum is encountered.
- Associate weights (penalties) with solution components; these determine impact of components on evaluation function value.
- Perform Iterative Improvement; when in local minimum, increase penalties of some solution components until improving steps become available.

Guided Local Search (GLS):
determine *initial candidate solution* s
initialize penalties
while *termination criterion* is not satisfied **do**
  compute modified evaluation function g' from g
    based on penalties
  perform subsidiary local search on s
    using evaluation function g'
  update penalties based on s

Guided Local Search (continued)

- **Modified evaluation function:**

$$g'(\pi, s) := g(\pi, s) + \sum_{i \in SC(\pi', s)} \texttt{penalty}(i),$$

where $SC(\pi', s)$ is the set of solution components of problem instance π' used in candidate solution s.

- **Penalty initialization:** For all i: $\texttt{penalty}(i) := 0$.
- **Penalty update** in local minimum s: Typically involves *penalty increase* of some or all solution components of s; often also occasional *penalty decrease* or *penalty smoothing*.
- **Subsidiary local search:** Often *Iterative Improvement*.

Potential problem:

Solution components required for (optimal) solution may also be present in many local minima.

Possible solutions:

A: Occasional decreases/smoothing of penalties.

B: Only increase penalties of solution components that are least likely to occur in (optimal) solutions.

Implementation of **B**:

Only increase penalties of solution components i with maximal utility [Voudouris and Tsang, 1995]:

$$\texttt{util}(s', i) := \frac{g_i(\pi, s')}{1 + \texttt{penalty}(i)}$$

Example: Guided Local Search (GLS) for the TSP

[Voudouris and Tsang 1995; 1999]

- **Given:** TSP instance G
- **Search space:** Hamiltonian cycles in G with n vertices;
- **Neighborhood:** 2-edge-exchange;
- **Solution components** edges of G; $g_e(G, p) := w(e)$;
- **Penalty initialization:** Set all edge penalties to zero.
- **Subsidiary local search:** Iterative First Improvement.
- **Penalty update:** Increment penalties for all edges with maximal utility by

$$\lambda := 0.3 \cdot \frac{w(s_{\text{2-opt}})}{n}$$

Lagrangian Method

- Change the objective function bringing constraints g_i into it

$$L(\vec{s}, \vec{\lambda}) = f(\vec{s}) + \sum_i \lambda_i g_i(\vec{s})$$

- λ_i are continous variables called Lagrangian Multipliers
- $L(\vec{s}^*, \lambda) \leq L(\vec{s}^*, \vec{\lambda}^*) \leq L(\vec{s}, \vec{\lambda}^*)$
- Alternate optimizations in $\vec{s}$ and in $\vec{\lambda}$

Outline

Evolutionary Algorithms

Key idea (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival

Iteratively apply genetic operators mutation, recombination, selection to a population of candidate solutions.

- Mutation introduces random variation in the genetic material of individuals (unary operator)
- Recombination of genetic material during reproduction produces offspring that combines features inherited from both parents (N-ary operator)
- Differences in evolutionary fitness lead selection of genetic traits ('survival of the fittest').

Terminology

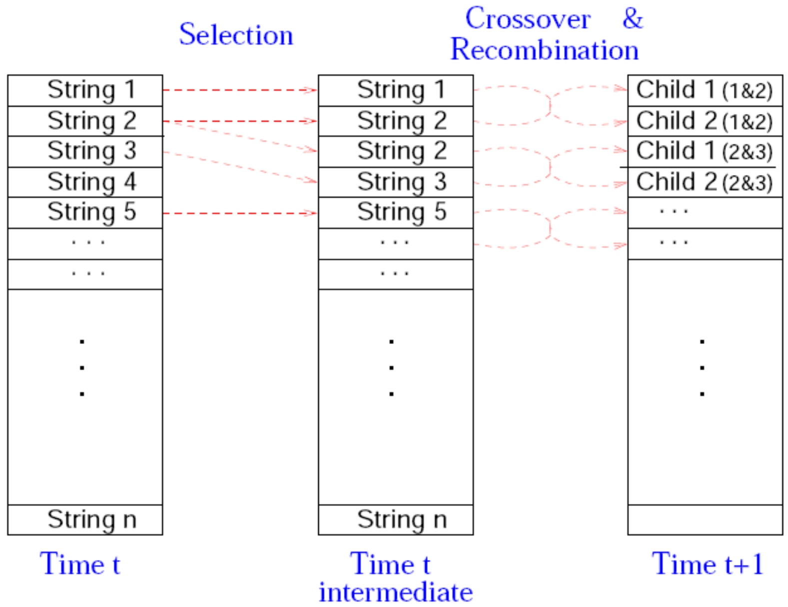
Individual	$\Longleftrightarrow$	Solution to a problem
Genotype space	$\Longleftrightarrow$	Set of all possible individuals determined by the solution encoding
Phenotype space	$\Longleftrightarrow$	Set of all possible individuals determined by the genotypes (ie, the variable–value themselves)
Population	$\Longleftrightarrow$	Set of candidate solutions
Chromosome	$\Longleftrightarrow$	Representation for a solution (*e.g.*, set of parameters)
Fitness	$\Longleftrightarrow$	Quality of a solution
Gene and Allele	$\Longleftrightarrow$	Part and value of the representation of a solution (*e.g.*, parameter or degree of freedom)
Crossover Mutation	$\Longleftrightarrow$	Search Operators
Natural Selection	$\Longleftrightarrow$	Promoting the reuse of good solutions

Original Streams

- Evolutionary Programming [Fogel et al. 1966]:
  - mainly used in continuous optimization
  - typically does not make use of recombination and uses stochastic selection based on tournament mechanisms.
  - often seeks to adapt the program to the problem rather than the solutions
- Evolution Strategies [Rechenberg, 1973; Schwefel, 1981]:
  - similar to Evolution Programming (developed independently)
  - originally developed for (continuous) numerical optimization problems;
  - operate on more natural representations of candidate solutions;
  - use self-adaptation of perturbation strength achieved by mutation;
  - typically use elitist deterministic selection.
- Genetic Algorithms (GAs) [Holland, 1975; Goldberg, 1989]:
  - mostly for discrete optimization;
  - often encode candidate solutions as bit strings of fixed length, (which is now known to be disadvantageous for combinatorial problems such as the TSP).

Evolutionary Algorithm (EA):
determine initial population sp

while termination criterion is not satisfied: **do**
- generate set spr of new candidate solutions by recombination
- generate set spm of new candidate solutions from spr and sp by mutation
- select new population sp from candidate solutions in sp, spr, and spm



Problem: Pure evolutionary algorithms often lack capability of sufficient search intensification.

Solution: Apply subsidiary local search after initialization, mutation and recombination.

Memetic Algorithms [Dawkins, 1997, Moscato, 1989]

- transmission of memes, mimicking cultural evolution which is supposed to be direct and Lamarckian
- (aka Genetic/Evolutionary Local Search, or Hybrid Evolutionary Algorithms if more involved local search including other metaheuristics, eg, tabu search)

Memetic Algorithm (MA):
determine initial population sp
perform subsidiary local search on sp
while termination criterion is not satisfied: **do**
- generate set spr of new candidate solutions by recombination
- perform subsidiary local search on spr
- generate set spm of new candidate solutions from spr and sp by mutation
- perform subsidiary local search on spm
- select new population sp from candidate solutions in sp, spr, and spm

Solution representation

Separation between solution encode/representation (genotype) from actual solution (phenotype)

Example

- genotype set made of strings of length l whose elements are symbols from an alphabet $\mathcal{A} \Rightarrow$ set of all individuals $\mathcal{A}^l$
  - the elements of strings are the genes
  - the values that each element can take are the alleles
- the search space is $\mathcal{X} \subseteq \mathcal{A}^l$,
- if the strings are member of a population they are called chromosomes and their recombination crossover
- an expression maps individual to solutions (phenotypes) $c : \mathcal{A}^l \mapsto \mathcal{S}$
- strings are evaluated by $f(c(x)) = g(x)$ which gives them a fitness

Example

1001010 1101100 0111010 1010010 1000010

0101110 0111101 0110110 1101000 1010101

10010 101101100011101 010100101000010

01011 100111101011011 011010001010101

Which Produces the Offspring

010111011011000111010111010001010101

100101001111010110110101001010000010

Note: binary representation is appealing but not always good (in constrained problems binary crossovers might not be good)

Conjectures on the goodness of EA

schema: subset of $\mathcal{A}^l$ where strings have a set of variables fixed.
Ex.: 1 * * 1

- exploit intrinsic parallelism of schemata
- Schema Theorem:

$$E[N(S, t+1)] \geq \frac{F(S,t)}{\bar{F}(S)} N(s,t)[1 - \epsilon(S,t)]$$

- a method for solving all problems $\Rightarrow$ disproved by No Free Lunch Theorems
- building block hypothesis

Initial Population

- Which size? Trade-off
- Minimum size: connectivity by recombination is achieved if at least one instance of every allele is guaranteed to be present at each gene. Ex: if binary:

$$P_2^* = (1 - (0.5)^{M-1})^l$$

  for $l = 50$, it is sufficient $M = 17$ to guarantee $P_2^* > 99.9\%$.
- Generation: often, independent, uninformed random picking from given search space.
- Attempt to cover at best the search space, eg, Latin hypercube, Quasi-random (low-discrepancy) methods (Quasi-Monte Carlo method).
- But: can also use multiple runs of construction heuristic.

Selection

Main idea: selection should be related to fitness

- Fitness proportionate selection (Roulette-wheel method)

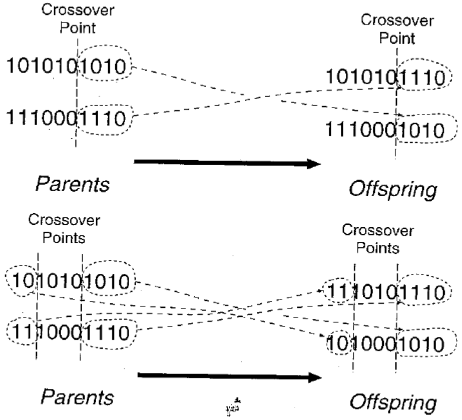
$$p_i = \frac{f_i}{\sum_j f_j}$$

- Tournament selection: a set of chromosomes is chosen and compared and the best chromosomes chosen.
- Rank based and selection pressure
- Fitness sharing (aka niching): probability of selection proportional to the number of other individuals in the same region of the search space.

Recombination (Crossover)

- Binary or assignment representations
  - one-point, two-point, m-point (preference to positional bias w.r.t. distributional bias)
  - uniform cross over (through a mask controlled by a Bernoulli parameter p)
- Permutations
  - Partially mapped crossover (PMX)
  - Mask based crossover
  - Order crossover (OX)
  - Cycle crossover (CX)
- Sets
  - greedy partition crossover (GPX)
- Real vectors
  - arithmetic crossovers
  - k-point crossover

Example: crossovers for binary representations



- Crossovers appear to be a crucial feature of success
- Therefore, more commonly: ad hoc crossovers
- Two off-springs are generally generated
- Crossover rate controls the application of the crossover. May be adaptive: high at the start and low when convergence

Mutation

- Goal: Introduce relatively small perturbations in candidate solutions in current population + offsprings obtained from recombination
- Typically, perturbations are applied stochastically and independently to each candidate solution
- Mutation rate controls the application of bit-wise mutations. It may be adaptive: low at the start and high when convergence
- Possible implementation through Poisson variable which determines the m genes which are likely to change allele.
- Can also use subsidiary selection function to determine subset of candidate solutions to which mutation is applied.
- With real vector representation: Gaussian mutation

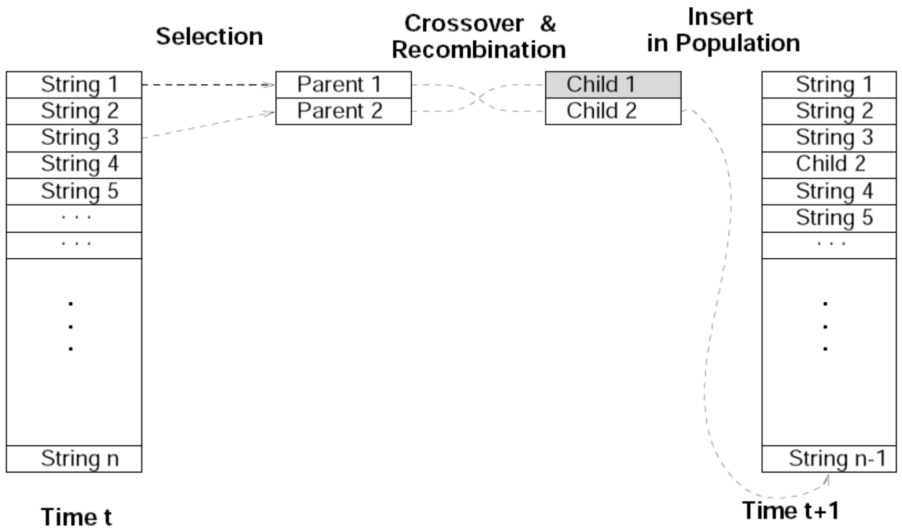
Subsidiary local search

- Often useful and necessary for obtaining high-quality candidate solutions.
- Typically consists of selecting some or all individuals in the given population and applying an iterative improvement procedure to each element of this set independently.

New Population

- Determines population for next cycle (generation) of the algorithm by selecting individual candidate solutions from
  - current population +
  - new candidate solutions from recombination, mutation (and subsidiary local search).
- Generational Replacement (λ, μ): $\lambda \leftarrow \mu$
- Elitist strategy $(\lambda + \mu)$ the best candidates are always selected
- Steady state (most common) only a small number of least fit individuals is replaced
- Goal: Obtain population of high-quality solutions while maintaining population diversity.

  Survival of the fittest and maintenance of diversity (duplicates avoided)



Example

A memetic algorithm for TSP

- **Search space:** set of Hamiltonian cycles
  Tours represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly before completing with greedy).
- **Recombination:** greedy recombination operator GX applied to $n/2$ pairs of tours chosen randomly:
  1) copy common edges (param. p_e)
  2) add new short edges (param. p_n)
  3) copy edges from parents ordered by increasing length (param. p_c)
  4) complete using randomized greedy.
- **Subsidiary local search:** LK variant.
- **Mutation:** apply double-bridge to tours chosen uniformly at random.
- **Selection:** Selects the μ best tours from current population of $\mu + \lambda$ tours (=simple elitist selection mechanism).
- **Restart operator:** whenever average bond distance in the population falls below 10.