DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 9

Genetic Algorithms

Marco Chiarandini

Outline

1. Task 2 – 2-opt for TSP
2. Metaheuristics
   Very Large Scale Neighborhoods
3. Population-based LS Methods

Outline

1. Task 2 – 2-opt for TSP
2. Metaheuristics
   Very Large Scale Neighborhoods
3. Population-based LS Methods

Numerical Results

Alg.	Err.	Size	sec.
DM63-2-281274-I-BB	8.99	657	171.50
DM63-2-260581-I-BB	8.99	657	146.96
DM63-2-281274-NN-BB	5.40	657	43.76
DM63-2-260581-NN-BB	5.05	657	35.75
Stuetzle-I-BB	8.99	657	4.16
Stuetzle-NN-BB	5.63	657	1.07
DM63-2-281274-I-FB	6.98	657	200.07
DM63-2-260581-I-FB	10.43	657	2.95
DM63-2-281274-NN-FB	6.44	657	66.78
DM63-2-260581-NN-FB	7.68	657	1.39
Stuetzle-I-FB	10.43	657	0.07
Stuetzle-NN-FB	9.56	657	0.08
DM63-2-281274-I-BFR	9.30	657	17.19
DM63-2-260581-I-BFR	8.03	657	153.99
DM63-2-281274-I-FFR	7.92	657	4.78
DM63-2-281274-NN-BFR	5.65	657	1.35
DM63-2-260581-NN-BFR	5.59	657	49.71
DM63-2-281274-NN-FFR	6.01	657	1.03
Stuetzle-I-BFRDLB	8.20	657	0.57
Stuetzle-I-FFRDLB	8.31	657	0.00
Stuetzle-NN-BFRDLB	5.81	657	0.15
Stuetzle-NN-FFRDLB	7.07	657	0.00

Outline

1. Task 2 – 2-opt for TSP
2. Metaheuristics
   Very Large Scale Neighborhoods
3. Population-based LS Methods

So far...

- beam search
- greedy randomized adaptive search procedure
- variable neighborhood search (and extensions)
- variable depth search (and Lin-Kernighan heuristic for TSP)
- ejection chains and dynasearch
- very large scale neigborhood search
- randomized iterative improvement (min-conflict heuristic, novelty)
- probabilistic iterative improvement (metropolis algorithm)
- simulated annealing (and noising method, threshold method, old bachelor acceptance)
- tabu search
- dynamic local search (guided local search)
- iterated local search
- iterated greedy
- multilevel refinement

Very Large Scale Neighborhoods

Key idea: use very large neighborhoods that can be searched very efficiently (preferably in polynomial time) or are searched heuristically

General framework to Variable Depth Search, Dynasearch and Ejection Chains

Three parts needed in a neighborhood search technique:

- Neighborhood structure: exponentially large neighborhoods
- Method for searching: polynomial time search algorithm
- Method for selecting: select the optimal solution in the neighborhood

VLSN allows to use the literature on polynomial time algorithms

Examples of VLSN Search:

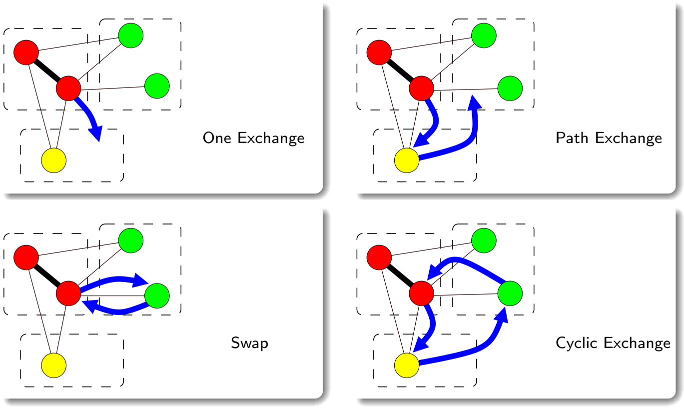
- based on Dynamic Programming or Network Flows
  - dynasearch and variants (ex. SMTWTP)
  - assignment based neighborhoods (ex. TSP)
  - cyclic exchange neighborhood (ex. VRP)
- based on polynomially solvable special cases of hard combinatorial optimization problems
  - Pyramidal tours
  - Halin Graphs

⇒ Idea: turn a special case into a neighborhood

Note

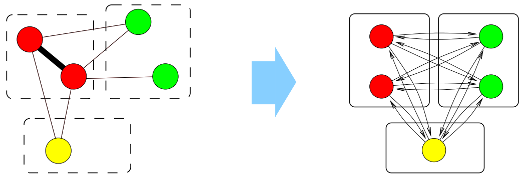
- they can provide a form of look ahead
- they can be effective but must be tested empirically

Iterative Improvement for GCP



Iterative Improvement for GCP

Exponential size but can be searched efficiently



Improvement Graph

A Subset Disjoint Negative Cost Cycle Problem in the Improvement Graph can be solved by dynamic programming in $\mathcal{O}(|V|^2 2^k |D'|)$.
Yet, heuristics rules can be adopted to reduce the complexity to $\mathcal{O}(|V'|^2)$ (Dumitrescu, 2002).

Iterative Improvement for GCP

Cyclic exchanges

- negative cost cycles can be detected rather *easily*
- further speed-up through Lemma from Lin-Kernighan
  If a sequence of edge costs has negative sum, then there is a cyclic permutation of these edges such that every partial sum is negative.

Path exchanges

- requires the introduction of dummy nodes in the definition of the improvement graph
- the same subset disjoint negative cycle method can then be used to retrieve either cylces or paths.

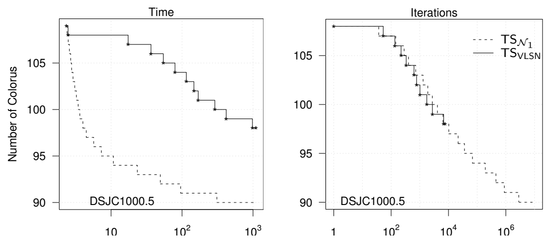
Iterative Improvement for GCP

Num. vertices	Num. distinct colorings	One exchange	Path and cyclic exchanges exhaustive	Path and cyclic exchanges truncated
3	7 (2)	0	0	0
4	63 (6)	1	0	1
5	756 (21)	10	0	9
6	14113 (112)	83	4	52
7	421555 (853)	532	15	260
8	22965511 (11117)	348	11	134
9	2461096985 (261080)	134	1	54

Tabu Search with VLSN for GCP

Tabu search algorithm

- integrates above iterative improvement algorithm
- tabu attributes are color assignments to nodes; not considered in the VLSN part



Outline

1. Task 2 – 2-opt for TSP

2. Metaheuristics
   Very Large Scale Neighborhoods

3. Population-based LS Methods

Population-based LS Methods

LS methods discussed so far manipulate one candidate solution of given problem instance in each search step.

Straightforward extension: Use *population* (*i.e.*, set) of candidate solutions instead.

Note:

- The use of populations provides a generic way to achieve search diversification.
- Population-based LS methods can fit into the general definition of the first Lectures by treating sets of candidate solutions as search positions.

Genetic Algorithms

Key idea: Iteratively apply *genetic operators mutation*, *recombination*, *selection* to a population of candidate solutions.

Inspired by simple model of biological evolution:

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

aka *Evolutionary Computing Algorithms* or *Evolutionary Algorithms (EA)* while distinct from *Evolutionary Programming* and *Evolution strategy*

Evolutionary Algorithm (EA):

determine initial population sp

While *termination criterion* is not satisfied:

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 perturbative search after initialization, mutation and recombination.

⇒ *Memetic Algorithms* (aka *Genetic Local Search*)

Evolutionary Algorithm (EA):
Memetic Algorithm (MA):

determine initial population sp

perform *subsidiary perturbative search* on sp

termination criterion is not satisfied:

generate set spr of new candidate solutions by recombination

perform *subsidiary perturbative search* on spr

generate set spm of new candidate solutions from spr and sp by mutation

perform *subsidiary perturbative search* on spm

select new population sp from candidate solutions in sp, spr, and spm

Solution representation

- neat separation between solution encode or representation (*genotype*) from actual variables (*phenotype*)
- genotype set $\mathcal{A}^l$ made of strings of length l whose elements are symbols from an alphabet $\mathcal{A}$
  $c : \mathcal{A}^l \rightarrow \mathcal{X}$
  - the elements of strings are the *genes*
  - the values of elements can take are the *alleles*
- the search space is then $\mathcal{S} \subseteq \mathcal{A}^l$,
- if the strings are member of a population they are called *chromosomes* and their recombination *crossover*
- strings are evaluated by $f(c(s))$ which gives them a *fitness*

⇒ binary representation is appealing but not always good (see subset selection problems)

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 ⇒ disproved by the *No Free Lunch Theorems*
- building block hypothesis

Initial Population

- Which size? Trade-off
- Minimum size: at least one allele is guarantee to be be present at each locus. Ex: if binary:
  $$P_2^* = (1-(0.5)^{M-1})^l$$
  for $l = 5$ it is sufficient $M = 17$.
- *Often:* independent, uninformed random picking from given search space.
- Attempt to cover at best the search space, eg, Latin hypercube.
- *But:* can also use multiple runs of construction heuristic.

Selection

Main idea: selection should be related to fitness

- Roulette-wheel method and the stochastic universal selection version: the probability of selecting any candidate solution s is proportional to its fitness value, $f(c(s))$.
- Rank based and selection pressure
- Tournament selection: a set of τ chromosomes is chosen and compared and the best chromosome chosen.

Recombination (Crossover)

- Binary or linear 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)
- Non-linear representations
  - (Permutations) Partially mapped crossover
  - (Permutations) mask based
- More commonly ad hoc crossovers are used as this appear to be a crucial feature of success
- 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

Example: One-point binary crossover operator

Given two parent candidate solutions $x_1 x_2 \ldots x_n$ and $y_1 y_2 \ldots y_n$:

1. choose index i from set $\{2, \ldots, n\}$ uniformly at random;
2. define offspring as $x_1 \ldots x_{i-1} y_i \ldots y_n$ and $y_1 \ldots y_{i-1} x_i \ldots x_n$.

cut

0	1	1	0	1	1	1	0	Parent 1
1	0	0	0	1	0	1	0	Parent 2
0	1	1	0	1	0	1	0	Offspring 1
1	0	0	0	1	1	1	0	Offspring 2

Mutation

- *Goal:* Introduce relatively small perturbations in candidate solutions in current population + offspring obtained from *recombination*.
- Typically, perturbations are applied stochastically and independently to each candidate solution; amount of perturbation is controlled by *mutation rate*.
- Mutation rate controls the application of bit-wise mutations. 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.
- The role of mutation (as compared to recombination) in high-performance evolutionary algorithms has been often underestimated

Subsidiary perturbative 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 obtained from *recombination*, *mutation* (+ *subsidiary perturbative search*). (λ, μ) $(\lambda + \mu)$
- *Goal:* Obtain population of high-quality solutions while maintaining *population diversity*.
- Selection is based on evaluation function (*fitness*) of candidate solutions such that better candidate solutions have a higher chance of 'surviving' the selection process.
- It is often beneficial to use *elitist selection strategies*, which ensure that the best candidate solutions are always selected.
- Most commonly used: steady state in which only one new chromosome is generated at each iteration
- Diversity is checked and duplicates avoided

Example: A memetic algorithm for TSP

- *Search space:* set of Hamiltonian cycles
  Note: tours can be represented as permutations of vertex indexes.
- **Initialization:** by randomized greedy heuristic (partial tour of $n/4$ vertices constructed randomly).
- **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 perturbative 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.

Types of evolutionary algorithms

- *Genetic Algorithms (GAs)* [Holland, 1975; Goldberg, 1989]:
  - have been applied to a very broad range of (mostly discrete) combinatorial problems;
  - often encode candidate solutions as bit strings of fixed length, which is now known to be disadvantageous for combinatorial problems such as the TSP.
- *Evolution Strategies* [Rechenberg, 1973; Schwefel, 1981]:
  - 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*.
- *Evolutionary Programming* [Fogel *et al.*, 1966]:
  - similar to Evolution Strategies (developed independently), but typically does not make use of *recombination* and uses *stochastic selection* based on *tournament mechanisms*.
  - often seek to adapt the program to the problem rather than the solutions