4.3. TABU SEARCH

- ***Basic principle***

Tabu search is an adaptive procedure which guides a hill descending heuristic to continue exploration of the solution space without becoming confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged.

Tabu search makes intensive use of adaptive memory structures to guide the search, contrary to other:
(a) memoryless approaches (e.g. simulated anealing)
(b) approaches using rigid memory structures (branch-and-bound)

- ***Tabu list***

A tabu list is a set of forbidden solutions determined by historical information from the last `tabu_tenure` iterations of the search.

At each iteration, tabu search moves from the current solution ***S*** to that within its neighborhood which most improves the objective function.

Moves which lead to solutions in the tabu list (or with similar characteristics) are forbidden. The value of `tabu_tenure` determines the number of iterations along which a move remains forbidden.

Moves associated with more restrictive tabu activation rules should be forbidden for a smaller number of iterations (smaller values of `tabu_tenure`).

If there are no improving moves, tabu search chooses the move which least deteriorates the objective function and inserts the reverse move (or the current solution) in the tabu list.

Need for efficient data structures for the fast computation of move values.

- ***Aspiration criteria***

Aspiration is a criterion which determines that a move is admissible, despite of being in the tabu list.

Representation of the tabu list:
(a) forbidden solutions → too much space to store and time to check
(b) forbidden moves → less space to store and time to check, but forbid many other solutions which have not been visited.

Example: 0-1 integer programming (minimization)

$\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{\boldsymbol{n}})$
$\boldsymbol{N}(\boldsymbol{x}) = \{\boldsymbol{x'} \in \{0,1\}^{\boldsymbol{n}} : \exists$ exactly one index $\boldsymbol{k} \in \{1,2,...,\boldsymbol{n}\}$ such that
$\boldsymbol{x'}_{\boldsymbol{k}} = 1 - \boldsymbol{x}_{\boldsymbol{k}}$ and $\boldsymbol{x'}_{\boldsymbol{j}} = \boldsymbol{x}_{\boldsymbol{j}}$ for any $\boldsymbol{j} \neq \boldsymbol{k}\}$

Current solution: $\boldsymbol{S} = (0,1,\boldsymbol{0})$
Neighbor: $(0,1,\boldsymbol{1})$
Move: $\boldsymbol{x}_3 = 0 \rightarrow \boldsymbol{x'}_3 = 1$
Solution which should be forbidden: $\boldsymbol{S} = (0,1,\boldsymbol{0})$
Forbidden, reverse move: $\boldsymbol{x}_3 = 1 \rightarrow \boldsymbol{x'}_3 = 0$
Solutions which are actually forbidden, as a result of inserting this move in the tabu list: $(0,0,\boldsymbol{0})$, $(0,1,\boldsymbol{0})$, $(1,0,\boldsymbol{0})$, $(1,1,\boldsymbol{0})$

Let $\boldsymbol{a} = \boldsymbol{c}(\boldsymbol{S^*})$, where $\boldsymbol{S^*}$ is the best solution found.
Then, a possible (but trivial) aspiration criterion consists in accepting a forbidden move which leads to a solution $\boldsymbol{S'}$ such that $\boldsymbol{c}(\boldsymbol{S'}) < \boldsymbol{a}$, i.e., if it leads to a solution improving the best solution found so far (we may also take e.g. $\boldsymbol{a} = \boldsymbol{c}(\boldsymbol{S^*})$, with $\boldsymbol{S^*}$ now representing the best solution found when the search reached $\boldsymbol{S'}$ for the first time).

Aspiration criteria may overrule the tabu status of a move.

- ***Template: Short-term memory tabu search algorithm (minimization)***

Step 1. Generate an initial solution $\boldsymbol{S}_0$ and set $\boldsymbol{S}, \boldsymbol{S}^* \leftarrow \boldsymbol{S}_0$
Step 2. Set the tabu list $\boldsymbol{T} \leftarrow \varnothing$
Step 3. Set iteration counters $\boldsymbol{p}, \boldsymbol{q} \leftarrow 0$
Step 4. While $\boldsymbol{p} \neq \boldsymbol{p}_{\max}$ and $\boldsymbol{q} \neq \boldsymbol{q}_{\max}$ do:
(a) Select the best neighbor $\boldsymbol{S}' \in \boldsymbol{N}(\boldsymbol{S}) \setminus \boldsymbol{T}$
(b) Select the best neighbor $\boldsymbol{S}'' \in \boldsymbol{N}(\boldsymbol{S}) \cap \boldsymbol{T}$ among those satisfying the aspiration criterion
(c) If $\boldsymbol{c}(\boldsymbol{S}'') < \boldsymbol{c}(\boldsymbol{S}')$, then set $\boldsymbol{S}' \leftarrow \boldsymbol{S}''$
(d) If $\boldsymbol{c}(\boldsymbol{S}') < \boldsymbol{c}(\boldsymbol{S}^*)$, then set $\boldsymbol{S}^* \leftarrow \boldsymbol{S}'$ and $\boldsymbol{q} \leftarrow 0$
(e) If $\boldsymbol{c}(\boldsymbol{S}') > \boldsymbol{c}(\boldsymbol{S})$, then insert the reverse move $(\boldsymbol{S}', \boldsymbol{S})$ in the tabu list $\boldsymbol{T}$
(f) Set $\boldsymbol{S} \leftarrow \boldsymbol{S}'$, $\boldsymbol{p} \leftarrow \boldsymbol{p} + 1$ and $\boldsymbol{q} \leftarrow \boldsymbol{q} + 1$
Step 5. Output the best solution $\boldsymbol{S}^*$

- ***Intensification and diversification***

Strategies which are used, respectively to:

(a) Focus the search into more promising regions (based on the structure of elite solutions), modifying choice rules to stimulate combinations of good moves and elite solutions.

(b) Drive the search to unexplored regions of the solution space, making use of penalties and incentives.

- ***Some other components and features***

Candidate lists: used to reduce the number of moves investigated at each iteration, when the neighborhood size is very large or the computational cost of evaluating each solution is very high

(a) random sampling of the neighborhood
(b) moving fractions of the neighborhood
(c) first improving move
(d) master list of elite moves which are examined in successive iterations, before reconstruction of the list

Elite solutions: good solutions which may be used to guide intensification (local optima or threshold with respect to the best solution)

Frequency counters: take into account the number of times or the frequency in which some elements appear in the solution.

- ***Strategic oscillation***: ***interplay intensification - diversification***

Strategic oscillation operates by moving until hitting a feasibility boundary, where normally the method would stop. Instead of stopping, the neighborhood definition is extended or the evaluation criterion for selecting moves is modified, to permit the boundary to be crossed.

The approach then proceeds for a specified depth beyond the boundary, and then turns around. At this point, the boundary is again approached and crossed, this time from the opposite direction. The process of repeatedly approaching and crossing the boundary from different directions creates a form of oscillation which gives the method its name.

Example: multidimensional 0-1 knapsack problem

(a) Variables are changed from 0 to 1 until reaching the feasibility boundary.
(b) Then, continue into the infeasible region using the same type of changes, but with a modified evaluator.
(c) After a selected number of steps, direction is reversed by changing variables from 1 to 0.

- ***Template: Tabu search algorithm (minimization)***

Step 1. Generate an initial solution $\boldsymbol{S}_0$ and set $\boldsymbol{S}, \boldsymbol{S}^* \leftarrow \boldsymbol{S}_0$
Step 2. Set the tabu list $\boldsymbol{T} \leftarrow \varnothing$
Step 3. Set iteration counters $\boldsymbol{p}, \boldsymbol{q} \leftarrow 0$
Step 4. While $\boldsymbol{p} \neq \boldsymbol{p}_{\max}$ and $\boldsymbol{q} \neq \boldsymbol{q}_{\max}$ do:
(a) Select the best neighbor $\boldsymbol{S}' \in \boldsymbol{N}(\boldsymbol{S}) \setminus \boldsymbol{T}$
(b) Select the best neighbor $\boldsymbol{S}'' \in \boldsymbol{N}(\boldsymbol{S}) \cap \boldsymbol{T}$ among those satisfying the aspiration criterion
(c) If $\boldsymbol{c}(\boldsymbol{S}'') < \boldsymbol{c}(\boldsymbol{S}')$, then set $\boldsymbol{S}' \leftarrow \boldsymbol{S}''$
(d) If $\boldsymbol{c}(\boldsymbol{S}') < \boldsymbol{c}(\boldsymbol{S}^*)$, then set $\boldsymbol{S}^* \leftarrow \boldsymbol{S}'$, $\boldsymbol{q} \leftarrow 0$, and update the set of elite solutions.
(e) If $\boldsymbol{c}(\boldsymbol{S}') > \boldsymbol{c}(\boldsymbol{S})$, then insert the reverse move $(\boldsymbol{S}',\boldsymbol{S})$ in the tabu list $\boldsymbol{T}$
(f) Update the aspiration criterion.
(g) Set $\boldsymbol{S} \leftarrow \boldsymbol{S}'$, $\boldsymbol{p} \leftarrow \boldsymbol{p} + 1$ and $\boldsymbol{q} \leftarrow \boldsymbol{q} + 1$
Step 5. Intensification strategy (basic idea): re-apply steps 2 to 4, starting from every solution $\boldsymbol{S}$ in the set of elite solutions.
Step 6. Diversification strategy (basic idea): if diversification was not sufficiently explored yet, use frequency counts to penalize frequent solution elements, generate a new initial solution $\boldsymbol{S}_0$ based on original costs and penalties, set $\boldsymbol{S} \leftarrow \boldsymbol{S}_0$ and go back to step 2.
Step 7. Output the best solution $\boldsymbol{S}^*$

- ***Comments***

(a) Intensive use of adaptive memory structures
(b) Requires more engineering for its implementation (many parameters to be set), but can produce good solutions in relatively small computation times if well done.
(c) Good quality solutions: best solutions found for many problems
(d) Since tabu search has more features to escape from local optima, it does not necessarily need very good initial solutions.
(e) Robustness
(f) Large number and diversity of applications

- ***Parallelization***

(a) Neighborhood decomposition (synchronous master-slave)
  → Single partitioning
  → Multiple partitioning (load balance)

(b) Multiple independent search threads

(c) Multiple cooperative search threads with common memory
  → Combination of different initial solution algorithms
  → Combination of different search strategies
  (e.g. tabu lists, tabu tenures)

4.4. GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES (GRASP)

- ***Principle: combination of constructive methods with local search***

Iterative procedure with completely independent iterations

Each iteration has two phases:
(a) construction
(b) local search

- ***Template: GRASP algorithm (minimization)***

> ***Step 1.*** Set $\boldsymbol{c}(\boldsymbol{S}^*) \leftarrow \infty$
> ***Step 2.*** For $\boldsymbol{n}_{\max}$ iterations do:
> (a) Build a solution $\boldsymbol{S}$ using a greedy randomized algorithm
> (b) Apply a local search procedure starting at $\boldsymbol{S}$ and finishing with $\boldsymbol{S}'$
> (c) If $\boldsymbol{c}(\boldsymbol{S}')$ improves the current best solution $\boldsymbol{S}^*$, i.e. if $\boldsymbol{c}(\boldsymbol{S}') < \boldsymbol{c}(\boldsymbol{S}^*)$, then set $\boldsymbol{S}^* \leftarrow \boldsymbol{S}'$
> ***Step 3.*** Output the best solution $\boldsymbol{S}^*$

- ***Construction phase***

A feasible solution is built, one element at-a-time.

At each iteration of the construction phase:
(a) Using a greedy criterion, evaluate the benefit of including each element in the solution.
(b) Create a restricted candidate list, formed by those elements whose benefits are among the best.
(c) At random, select one element from the candidate list.
(d) Adapt problem information according with the element just included.

The choice of the next element to be added is determined by placing all elements in a list, ordered with respect to a greedy function.

Restriction on the candidate elements may be based on:
(a) maximum number of elements in the list
(b) value of the elements with respect to the pure greedy choice

An element from the restricted candidate list is chosen from among the best candidates in the list (not necessarily the top candidate):
(a) solution quality depends on the ***quality*** of the elements in the candidate list ***L***.
(b) solution diversity depends on the ***cardinality*** of the candidate list ***L***.

Diversification relies on controlled randomization: different solutions are built at each GRASP iteration.

Greedy construction leads to solutions close to local optima, speeding up the local search.

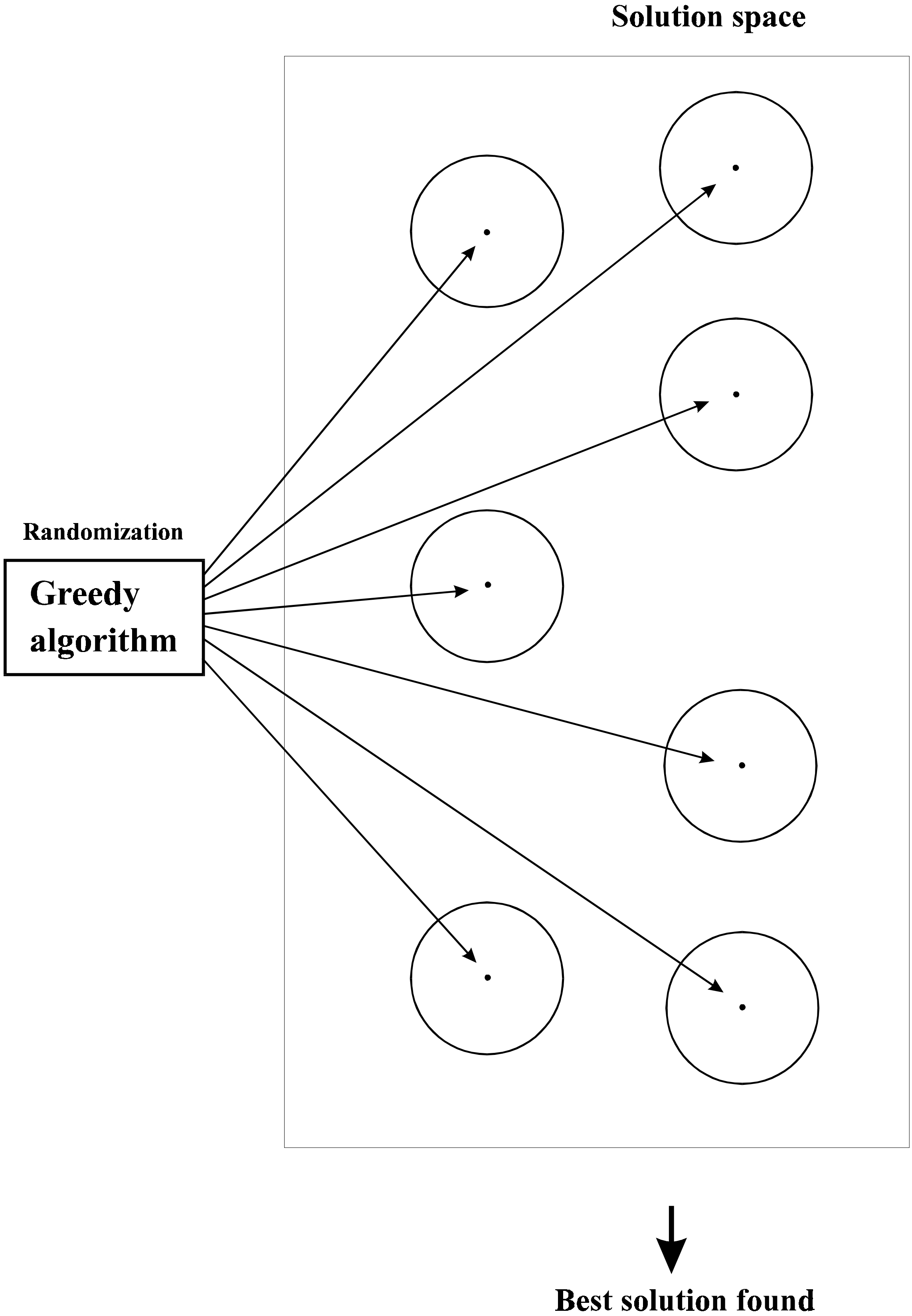
- ***Local search phase***

Construction phase not necessarily stops at a local optimum:
almost always local search can improve the solution found.

Be careful to:
(a) make an appropriate neighborhood choice,
(b) use efficient data structures to speed up local search, and
(c) have a good initial solution.

Starting from a good initial solution (close to a local optimum), such as those produced by a greedy (randomized) algorithm, leads to fast local search.

Figure 5: GRASP as a sampling technique



- ***Solution quality***

GRASP may be seen as a repetitive sampling technique on the solution space:

(a) Each GRASP iteration acts as sampling a solution from an unknown distribution.

(b) Average and variance of the sample depends on the restrictions imposed for the creation of the restricted candidate list (RCL).

$|RCL| = 1 \rightarrow$ all solutions found are the same (pure greedy algorithm)
mean = value of the greedy solution
variance = 0

$|RCL|$ increases $\rightarrow$ solution quality decreases (bad random choices)
mean < value of the greedy solution
variance increases

Best solution found may be better than the greedy solution value, outperforming the mean value.

- ***Example: MAX SAT, 1391 variables, 3026 clauses***

100,000 GRASP iterations

$\lvert RCL \rvert$	Worst value	Occurrences	Average	Best value	Occurences
1	3024	100,000	3024.00	3024	100,000
2	3022	151	3023.94	3024	93,796
4	3021	75	3023.79	3025	2
8	3019	1	3023.53	3026	5
16	3019	16	3023.27	3026	25
32	3017	1	3023.14	3026	58
64	3017	4	3023.00	3026	163
128	3017	4	3022.91	3026	281
256	3016	1	3022.89	3026	689

$\lvert \mathrm{RCL} \rvert$	3019	3020	3021	3022	3023	3024	3025	3026
1						100,000		
2				151	6,053	93,796		
4			75	1,676	17,744	80,503	2	
8	1	50	750	6,566	31,257	61,336	35	5
16	16	282	2,485	13,274	38,329	45,547	42	25

- ***Comments***

(a) Very easy to implement, given a greedy heuristic and a local search procedure

(b) Greedy heuristics can be conceived for almost any combinatorial optimization problem

(c) Very few parameters to be set:
  - restrictiveness of the candidate list (size or quality)
  - number of iterations

(d) Contrary to tabu search, GRASP needs and benefits from good initial solutions:
  - TS has mechanisms such as diversification and intensification to attain a good solution even if the initial solution is bad; GRASP relies only on randomization from one iteration to the other and each iteration benefits from the quality of the initial solution.

(e) Use of filters to speedup the local search:
  - Apply local search only to the best of a sequence of solutions built by the construction phase.
  - Apply local search only to solutions built by the construction phase which satisfy some threshold acceptance criterion.

(f) Drawback: memoryless technique which does not make use of information collected during the search.

(g) Despite of its simplicity, many successful applications

- ***Some applications***

(a) Set covering: Steiner triple systems, flexible manufacturing systems, airline scheduling
(b) Production planning and scheduling: computer-aided process planning, just-in-time schedule of parallel machines, single machine scheduling with penalties, single machine scheduling with sequence-dependent costs
(c) Location problems: $\boldsymbol{p}$-hub, quadratic assignment, positioning containers in trains
(d) Graphs: maximum independent set, coloring, 2-partitioning, graph planarization
(e) Power transmission system expansion planning
(f) Vehicle routing
(g) Satisfiability problem

- ***Parallelization***

Very simple and direct parallelization: perform $\boldsymbol{n}_{\max}/\boldsymbol{p}$ iterations at each of $\boldsymbol{p}$ processors and collect the best solution found.

4.5. VARIABLE (DYNAMIC) NEIGHBORHOOD SEARCH

- ***Basic principle***

Apply local search, sampling some neighbors of the current solution until finding a local optimum.
Then, make use of an enlarged neighborhood.

Examples of enlarged neighborhoods:
(a) Traveling salesman problem: change from $\boldsymbol{k}$-opt to ($\boldsymbol{k}$+1)-opt
(b) 0-1 integer programming: increase from $\boldsymbol{k}$ to $\boldsymbol{k}$+1 the number of variables which are complemented

- ***Template: Variable (dynamic) neighborhood search algorithm (minimization)***

Step 1. Let $\boldsymbol{S}_0$ be an initial feasible solution, set $\boldsymbol{k} \leftarrow 1$ and $\boldsymbol{S} \leftarrow \boldsymbol{S}_0$
Step 2. While (termination condition is not satisfied) do:
Step 3.1. While there exists $\boldsymbol{S'} \in \boldsymbol{N^k}(\boldsymbol{S})$ such that $\boldsymbol{c}(\boldsymbol{S'}) < \boldsymbol{c}(\boldsymbol{S})$ do:
(a) Set $\boldsymbol{S} \leftarrow \boldsymbol{S'}$
Step 3.2. Set $\boldsymbol{k} \leftarrow \boldsymbol{k} + 1$ (enlarged neighborhood $\boldsymbol{N^{k+1}}(\boldsymbol{S})$)
Step 3. Set $\boldsymbol{S^*} \leftarrow \boldsymbol{S}$ and output the best solution $\boldsymbol{S^*}$

- ***Comments***

(a) Complementary with respect to GRASP:
VNS applies randomization during local search, while GRASP applies randomization during the construction phase.
(b) Good results in recent applications.

4.6. LARGE STEP OPTIMIZATION

- ***Basic principle***

Apply local search until finding a local optimum.
Then, diversify the search taking a larger step and obtaining a neighbor from a much larger neighborhood.

Traveling salesman problem: perform local search using Lin & Kernighan, then apply a 4-opt move.

- ***Template: Large step optimization (minimization)***

> ***Step 1.*** Let S_0 be an initial feasible solution and set $S \leftarrow S_0$
> ***Step 2.*** While (termination condition is not satisfied) do:
> ***Step 3.1.*** While there exists $S' \in N(S)$ such that $c(S') < c(S)$ do:
> (a) Set $S \leftarrow S'$
> ***Step 3.2.*** Diversify the search applying a larger step and moving towards a structuraly different new solution.
> ***Step 3.*** Set $S^* \leftarrow S$ and output the best solution S^*

- ***Comments***

(a) Similar to VNS: based on diversifying the search through a larger step, leading to a structuraly different solution, instead of enlarging the neighborhood step by step.
(b) Good results in recent applications.

4.7. GENETIC ALGORITHMS

- ***Basic principles***

Simulation of the development of new and better populations among different species during evolution, just as their biological counterparts.

Genetic algortithms use information of a population of individuals (solutions) during the search for better solutions, and not only information from a single individual.

Computation of sub-optimal solutions by letting a set of random solutions undergo a sequence of unary and binary transformations guided by a selection scheme biased towards high-quality solutions.

- ***Solution encoding***

Solutions coded as finite length strings

solution vector	↔	chromosome	(0110010101)
↓		↓	
elements	↔	genes	0-1

- ***Critical parts***

(a) Representation of the solutions
(b) Generation of the initial population
(c) Selection of individuals in an old population that will be allowed to affect the individuals of a new population. In terms of evolution: selection of suitable parents for the new population
(d) Genetic operators, such as crossover and mutation, i.e. how to recombine the genetic heritage from their parents in the previous generation

- ***Main operators***

(a) ***Evaluation*** of the fitness (objective function) of each individual
(b) ***Selection*** of well adapted individuals of the population
(c) ***Crossover*** (recombination) of selected individuals
(d) ***Mutation*** of some individuals

Example of crossover:

$$\begin{array}{lcl} \boldsymbol{S_1} = (1,0,1,\boldsymbol{0},\boldsymbol{1},\boldsymbol{1}) & & \boldsymbol{S'_1} = (1,0,1,\boldsymbol{0},\boldsymbol{0},\boldsymbol{1}) \\ & \rightarrow & \\ \boldsymbol{S_2} = (0,1,1,\boldsymbol{0},\boldsymbol{0},\boldsymbol{1}) & & \boldsymbol{S'_2} = (0,1,1,\boldsymbol{0},\boldsymbol{1},\boldsymbol{1}) \end{array}$$

- ***Template: Basic genetic algorithm (minimization)***

Step 1. Set $\boldsymbol{t} \leftarrow 0$ and initialize population $\boldsymbol{P}(\boldsymbol{t})$
Step 2. While (termination condition is not satisfied) do:
(a) Evaluate the fitness of each individual in $\boldsymbol{P}(\boldsymbol{t})$
(b) Set $\boldsymbol{t} \leftarrow \boldsymbol{t} + 1$
(c) ***New_P***($\boldsymbol{t}$) $\leftarrow$ selected elements from $\boldsymbol{P}(\boldsymbol{t} - 1)$
(d) ***Cross_P***($\boldsymbol{t}$) $\leftarrow$ recombination of elements from ***New_P***($\boldsymbol{t}$)
(e) ***Mutation_P***($\boldsymbol{t}$) $\leftarrow$ mutation of elements from ***Cross_P***($\boldsymbol{t}$)
Step 3. Output the best solution $\boldsymbol{S^*}$ among all those in the population

- ***Typical termination conditions***

(a) Maximum number of iterations
(b) The population is dominated by a few individuals
(c) The best objective function of the population is constant over a given number of generations

- ***Comments***

(a) Critical parts: solution representation (encoding), operators
(b) Not competitive for combinatorial optimization problems so far (good results: set covering)
(c) Common ideas with respect to scatter search

- ***Parallelization***

Several possible levels of parallelism:
(a) Phase level parallelism: e.g. compute in parallel the fitness function for all individuals in the population.
(b) Population level parallelism: divide the original population into many subpopulations. The subpopulations are guided to evolve and periodically some of the best individuals are exchanged between subpopulations.
(c) Process level paralellism: many genetic algorithms executed in parallel, but now they can be different and independent of each other.

The two highest levels (b) and (c) of paralellism are the basis for parallel genetic algorithms:
(a) Centralized: basic calculations or phases of the sequential GA are performed in many slave processors, whereas the master maintains the actual population (master-slave or farming model, e.g. the master sends one or more individuals to each slave for fitness evaluation and collects information from them).
(b) Distributed: direct application of the population level parallelism (island model, since the population is divided into many subpopulations, each of them living in its own "island").
A host is needed to control the exchange of individuals between subpopulations, as well as the execution termination.
(c) Network model: derived from the distributed model, by increasing the number of subpopulations such that there is only one individual per island. In addition, the individuals are connected together by one of many possible interconnection networks (e.g. 2-dimensional grid, hypercube, ring). Each individual is living in its own site and occasionally interacts with some of its neighbors, i.e., with some probability an individual

mates randomly with one of its neighbors and may be replaced by the resulting offspring (also called diffusion model, because good individuals are effectively diffused around the topology of individuals during execution).

4.8. ASYNCHRONOUS TEAMS (A-TEAMS)[1]

[1] Esta seção foi preparada por Marcus Vinicius S. Poggi de Aragão, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rua Marquês de São Vicente 225, Rio de Janeiro 22453-900, Brazil.

4.9. ANT COLONIES

- ***Principle:***

Simulation of the behavior of a set of agents that cooperate to solve an optimization problem by means of very simple communications

Ants seem to be able to find their way (from the nest to a food source and back, or around an obstacle) with relative ease. Enthomology studies discovered that in many cases this capacity is the result of the interplay via chemical communication between ants (through a substance called ***pheromone***) and an emergent phenomena caused by the presence of many ants.

- ***Example: Traveling salesman problem***

Each ant acts as follows:

(a) At every step the ant chooses, using a probabilistic rule, a city to move among those not yet visited.

(b) The probability of choosing edge (i,j) is directly proportional to the amount τ_{ij} of pheromone in the edge (i,j) and to the inverse of the edge length c_{ij}.

(c) The ant remembers each city already visited.

(d) After a tour has been completed, the ant lays a positive trail $\Delta\tau_{ij}$ (some amount of pheromone) on each edge (i,j) of the tour ($\Delta\tau_{ij} = 0$ for edges not in the tour).

(e) Account for evaporation of pheromone at every edge (i,j), considering a evaporation factor β, with $\beta < 1$: $\tau_{ij} \leftarrow \beta.\tau_{ij} + \Delta\tau_{ij}$.

Since the initial values of τ_{ij} are constant for every edge, initially the ants are free to move almost randomly.

If we now consider M ants moving on the graph simultaneously, τ_{ij} is the way by which ants communicate: ants choose with higher probability edges with higher amounts of pheromone. Therefore, ants laying trail on a set of edges make them more "desirable" for other ants.

- ***Ant colony algorithm for TSP:***

The ant colony algorithm may be stated as the concurrent superposition of M single ant procedures.

Step 1. Let Q and τ_0 be constants, and set $c(S^*) \leftarrow \infty$
Step 2. Set $\Delta\tau_{ij} \leftarrow 0$ and $\tau_{ij} \leftarrow \tau_0$ for every edge (i,j)
Step 3. For each ant $k = 1,2,...,M$
(a) Select the initial city from which ant k departs
(b) Obtain a tour T^k for each ant k according to the procedure followed by each individual ant
(c) Let L^k be the length of tour T^k
(d) If $L^k < c(S^*)$, then set $S^* \leftarrow T^k$
(e) Compute the amount of trail left by ant k as
$\Delta\tau_{ij}(k) = Q / L^k$, if edge (i,j) belongs to tour T^k,
$\Delta\tau_{ij}(k) = 0$ otherwise
(f) Set $\Delta\tau_{ij} \leftarrow \Delta\tau_{ij} + \Delta\tau_{ij}(k)$
Step 4. Set $\tau_{ij} \leftarrow \beta.\tau_{ij} + \Delta\tau_{ij}$
Step 5. If the best tour S^* tour has not changed for the last global $n_{\max}$ iterations, then stop with S^* as the best solution found, otherwise go back to step 2.

Each ant, if it did not consider trail effects, would move with a local, probabilistic greedy rule, which most likely would lead to a bad final solution (nearest insertion rule: first edges are good ones, but choices for the final steps are constrained by early steps).

If we now consider the effect of the simultaneous presence of many ants, then each one contributes to a part of the trail distribution. Good sets of edges will be followed by many ants and therefore will receive a great amount of trail.

- ***Comments***

(a) Toy problems solved so far (TSP, QAP, graph coloring).
(b) Behavior of the computational model is not very like that of the natural model of ant colonies (e.g., in real life ants leave trail during their movement, not after reaching their final destination).
(c) Ants communicate by exchanging a particular kind of information, i.e. the amount of trail (pheromone), without no direct communication between them, and only local information is used to take decisions: the model is suitable to parallelization, based on the use of a small amount of global information.
(d) Like GAs, ant colonies are also based on a population of agents, not only on a single agent (solution).

5. APPLICATIONS AND GENERAL REMARKS

- ***Some applications:***

  - Optimal Location and Sizing of Off-Shore Platforms (1988)
  - Cell Formation in Group Technology (1993)
  - Circuit Partitioning for Pseudo-Exhaustive Logical Test (1994)
  - Multiprocessor Scheduling under Precedence Constraints (1995)
  - Query Optimization in Distributed Relational Databases (1995)
  - Graph Planarization (1996)
  - Phylogenetic Trees under Parsimony (1997)
  - Traffic Assignment in TDMA Systems (1997)

- ***General remarks:***

  - Basic principles:
    - efficient data structures
    - preprocessing
    - good constructs for initial solutions
    - reinitialization procedures
    - solution improvement by local search
    - controlled randomization
    - search diversification
    - intensification into promising regions

  - Power of improved local search strategies to find better solutions

  - New classes of heuristics: ***metaheuristics***
    - multi-start
    - simulated annealing
    - tabu search
    - greedy randomized adaptive search procedures
    - variable neighborhood search
    - large step optimization
    - genetic algorithms
    - asynchronous teams
    - ant colonies

- Illustration of some common features and ingredients:
  - local search
  - moves and neighborhoods
  - diversification
    (large step, randomization, frequency counters, extended neighborhood, mutation)