Local Beam Search

The local beam search algorithm:

- keeps track of k states rather than just one
- begins with k randomly generated states
- at each step, all the successors of k states

The Idea

In a local beam search useful information is passed among the k parallel search threads. For example, if one state generates several good successors and the other $k-1$ states all generate bad successors, then the effect is that

Tabu Search

Key Features

- Always move to the best *available* neighborhood solution, even if it is worse than the current solution
- Maintain a list of solution points that must be avoided (not allowed) or a list of move features that are not allowed:
  - This is the Tabu List.
- Update the Tabu List based on some memory structure (short-term memory):
  - Remove Tabu moves after some time period has elapsed (Tenure).
- Allow for exceptions

Memory Structures

The memory structures used in Tabu Search can be divided into three categories:

Short-term The list of solutions recently considered. If a potential solution appears on this list, it cannot be revisited until it reaches an expiration point (Tenure).

Intermediate-term A list of rules intended to bias the search towards promising areas of the search space.

Long-term Rules that promote diversity in the search process (i.e. regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Aspiration Criteria

A criteria which allows a Tabu move to be accepted under certain conditions.
Most common aspiration criterion: If the move finds a new best solution, then accept the move even if the move is Tabu.

Tenure

The Tabu Tenure is the number of iterations that a move stays in the Tabu List

too small risk of cycling

too large may restrict the search too much

Intensification

Search parameters can be locally modified in order to perform intensification and/or diversification
Intensification – usually applied when no configurations with a quality comparable to that of stored elite configuration(s), have been found in the

Stopping Criteria

Potential Stopping Criteria:

- Number of iterations
- Number of iterations without improvement
-
-

Evolutionary Algorithms

Evolutionary Algorithms [Wikipedia]

An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm.

An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the

Chromosome

Chromosome – a package of DNA with part of all of the genetic materials of an organism

Artificial Chromosome

In Evolutionary Algorithms, an artificial chromosome is a genetic representation of the task to be solved.

Typically:

$$1 \text{ individual} = 1 \text{ chromosome} = 1 \text{ solution}$$

Also called a genotype.

Representation

Individuals / chromosomes can be represented as a string of values. Typically:

- Binary

Binary									
0	1	0	0	1	1	1	0	1	1

- Integer

Integer									
2	1	11	2	3	78				

- Floating-point

Float									

Well-Suited Chromosome: Features

- It must allow the accessibility of all admissible points in the search space.
- Design the chromosome in such a way that it covers only the search space and no additional areas so that there is no redundancy or only as little

Genotype vs Phenotype

Genotype – Organism's full hereditary information, even if not expressed. Directly inherited from parents

Genes vs Alleles

Genotype – Organism's full hereditary information, even if not expressed. Directly inherited from parents

Population

The set of solutions (individuals / chromosomes / genotypes) is called a population.

Individual Selection Mechanisms

Crossover Mechanisms

Potential Mutation

Mutation / Probability of Mutation

- Each component (bit, etc.) of every individual / chromosome is modified with *mutation probability* m_p

Algorithm 1.1 Genetic Algorithm Pseudocode

```
1: function GENETIC-ALGORITHM($population$, $fitness$) returns an individual
2:     repeat
3:         $weights \leftarrow$ WEIGHTED-BY($population, fitness$)
4:         $population2 \leftarrow$ empty list
5:         for $i = 1$ to SIZE($population$) do
6:             $parent1$, $parent2 \leftarrow$ WEIGHTED-RANDOM-CHOICES($population, weights, 2$)
7:             $child \leftarrow$ REPRODUCE($parent1$, $parent2$)
8:             if small random probability then
9:                 $child \leftarrow$ MUTATE($child$)
10:            end if
11:            add $child$ to $population2$
12:        end for
13:        $population \leftarrow population2$
14:    until some individual is fit enough, or enough time has elapsed
15:    return the best individual in $population$, according to $fitness$
16: end function
```

Genetic Algorithms

Design Issues

- Choosing basic implementation issues:
  - representation
  -
  -
  -
- Termination criteria

- Performance, scalability
- Solution is only as good as the evaluation function (often the hardest part)

Benefits

- Easy to understand and implement
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always has an answer
  - Answers get better with time
- Inherently parallel $\rightarrow$
- Variety of ways to improve performance as knowledge about the problem domain is gained
- Can exploit historical / alternative solutions
- Can be easily

When to Use

- Other solutions too slow or overly complicated (intractable mathematically)
- As an exploratory tool to examine new approaches

Selected GA Applications