An Introduction to Genetic Algorithms

Sami Khuri
Dept. of Math. & Computer Science
San José State University
One Washington Square
San José, CA 95192-0103
U.S.A.
khuri@cs.sjsu.edu

Outline

- Introduction to the Simple Genetic Algorithm.
- Applying Genetic Algorithms to Tackle NP-hard Problems.
- Other Applications of Genetic Algorithms:
  - Tracking a Criminal Suspect.
  - Genetic Programming.
- Concluding Remarks.

Introduction to Genetic Algorithms

- GA's were developed by John Holland.

- GA's are search procedures based on the mechanics of natural selection and natural genetics.

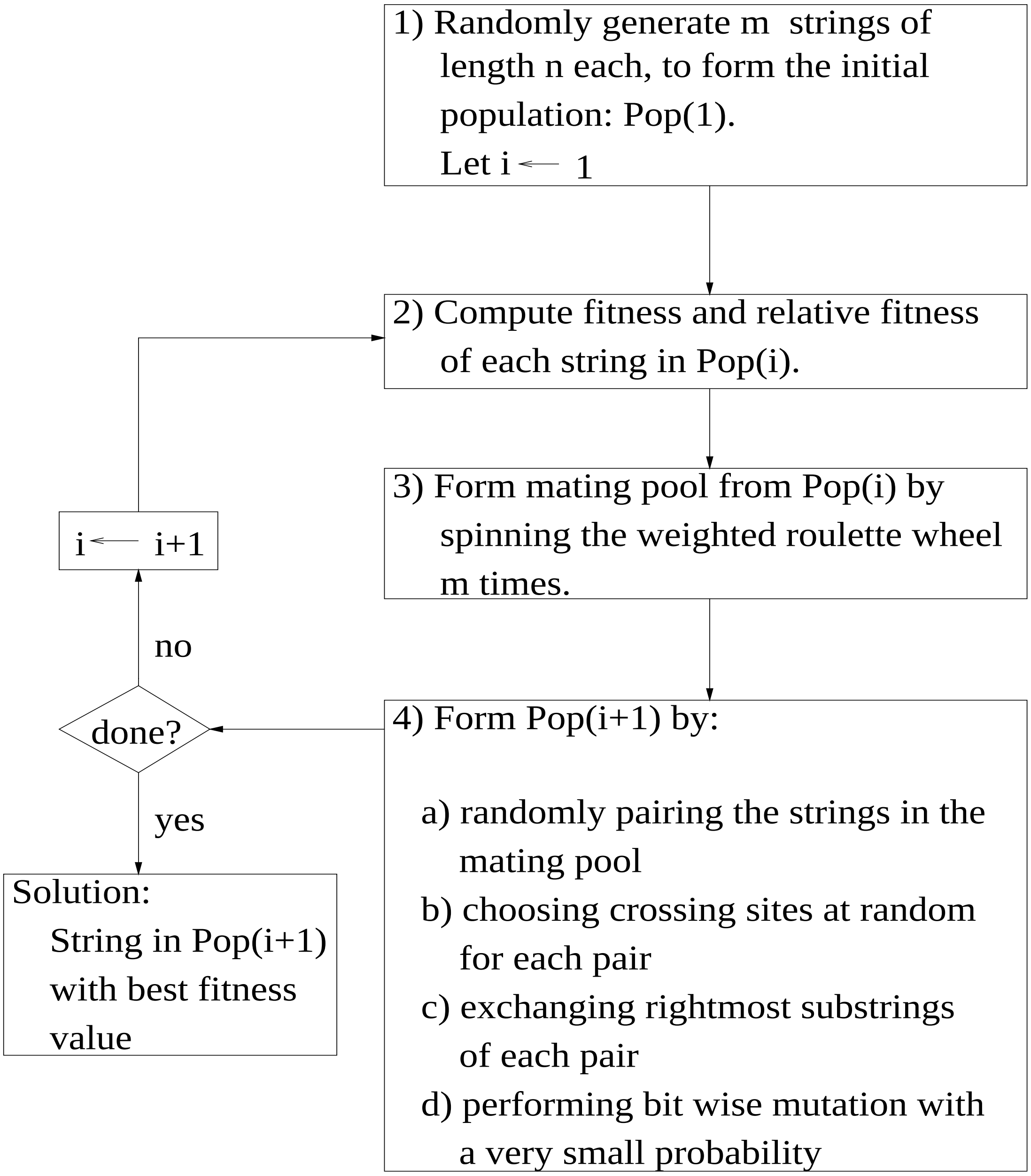
- GA's make few assumptions about the problem domain and can thus be applied to a broad range of problems.

- GA's are randomized, but not directionless, search procedures that maneuver through complex spaces looking for optimal solutions.

Comparison with other optimization procedures

Genetic Algorithm	Traditional Algorithm
• works with coding of parameter set	• works with parameters themselves
• searches from population of points	• searches from a single point
• uses payoff information: objective function	• uses auxiliary knowledge: derivatives, gradients, etc...
• uses probabilistic transition values	• is deterministic

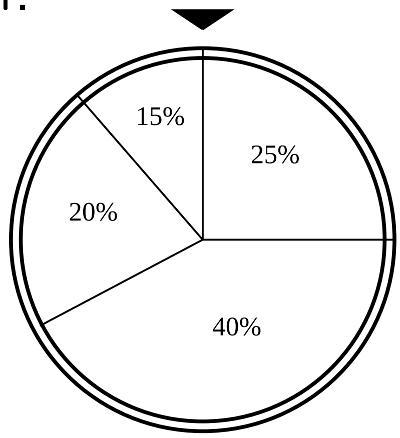
Simple Genetic Algorithm [KB90]



- Example:

Strings	Fitness	Relative Fitness
1101010	52.5	.25
1011001	84.0	.40
0100110	42.0	.20
1000101	31.5	.15

- Roulette Wheel:



- Mating Pool:

1011001
0100110
1011001
1101010

- **Crossover**
  - Choose 2 strings from mating pool.
    Choose a X-site:

1	0	1	|	1	0	0	1
1	1	0	|	1	0	1	0

  - Children:

1	0	1	1	0	1	0
1	1	0	1	0	0	1

- **Mutation:**
  - Flip a randomly chosen bit:

1	0	1	1	0	**1**	0
1	0	1	1	0	**0**	0

The 0/1 Knapsack Problem

Objects:	$1 \ldots i \ldots n$
Weights:	$w_1 \ldots w_i \ldots w_n$
Profits:	$p_1 \ldots p_i \ldots p_n$
Knapsack capacity:	M

Maximize $\sum_{i=1}^{n} x_i p_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq M$, where $x_i \in \{0, 1\}$ for $1 \leq i \leq n$.

Each string of the population represents a possible solution. If the j^{th} position of the string is 1 i.e. $x_j = 1$, then the j^{th} object is in the knapsack; otherwise it is not.

A string might represent an infeasible solution: the total sum of the weights of the objects exceeds the capacity of the knapsack.

Problem Instance:

Objects:	1	2	3	4	5	6	7	8	9	10
Weights:	20	18	16	16	12	8	6	5	3	1
Profits:	55	40	30	27	20	13	9	7	4	1

Knapsack capacity : 80

String 1001001011 is of weight 46 and has a profit of 96.

Two alternatives for the fitness functions are:

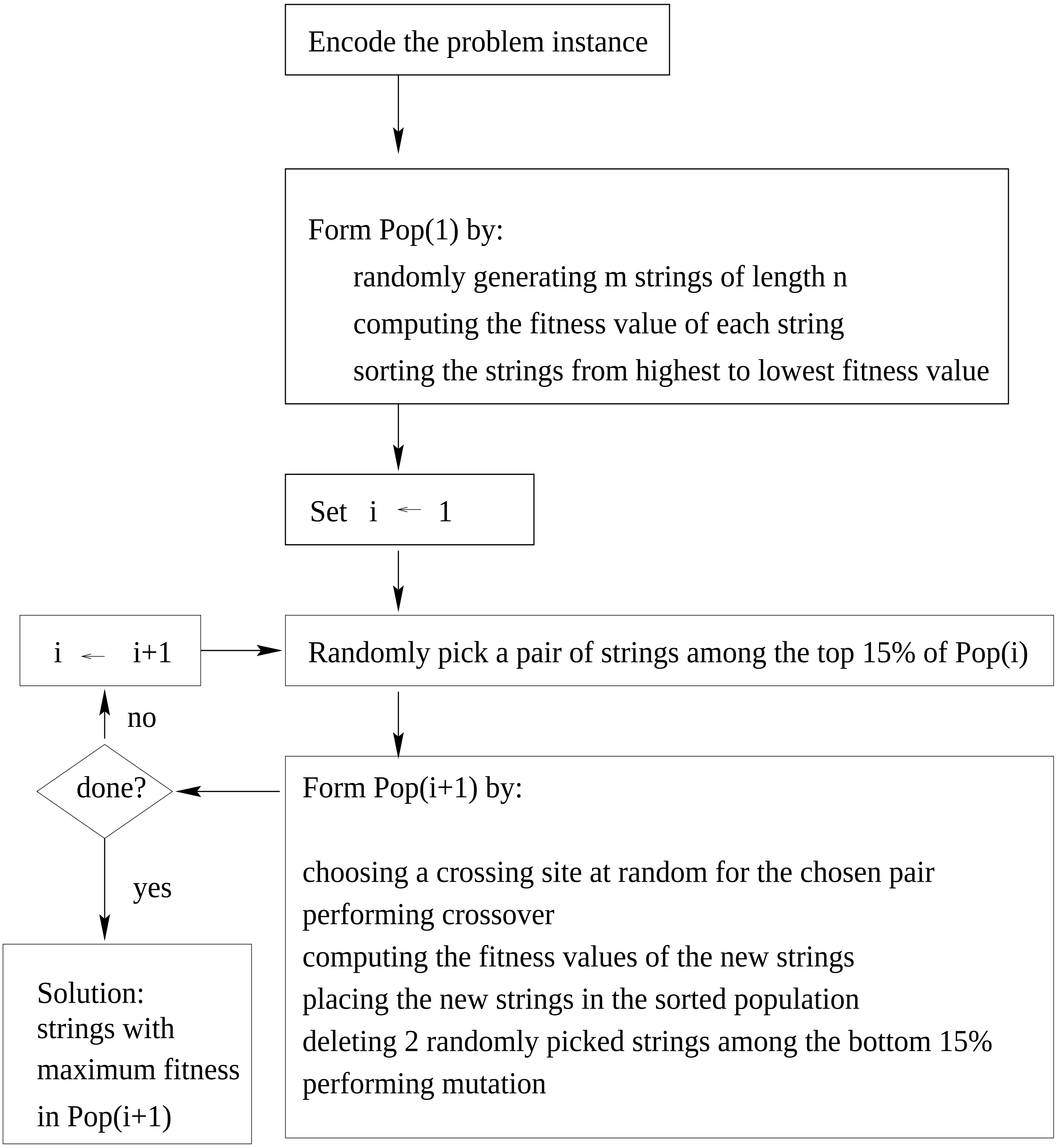
$$f(\vec{x}) = \sum_{i=1,n} x_i p_i,$$

where $\vec{x} = (x_1, x_2, \ldots, x_n)$ is a feasible solution; or

$$f(\vec{x}) = \sum_{i=1,n} x_i p_i - penalty$$

for any $\vec{x}$, where infeasible strings are penalized.

.


Encode the problem instance
Form Pop(1) by:
randomly generating m strings of length n
computing the fitness value of each string
sorting the strings from highest to lowest fitness value
Set i ← 1
i ← i+1
Randomly pick a pair of strings among the top 15% of Pop(i)
no
done?
yes
Form Pop(i+1) by:
choosing a crossing site at random for the chosen pair
performing crossover
computing the fitness values of the new strings
placing the new strings in the sorted population
deleting 2 randomly picked strings among the bottom 15%
performing mutation
Solution:
strings with
maximum fitness
in Pop(i+1)

Representations, Operators and Techniques:

- Integer, real and Gray code representations.

- Randomly generated and seeded initial populations.

- Inversion and mutation swap.

- 2-point, k-point, uniform, edge recombination, and partially mapped crossovers.

- Elitism.

Partially Mapped Crossover (PMX)

- PMX is used with permutations (eg. TSP).

- Parents: Choose 2 X-sites.

10	3	8	1	9	2	4	5	6	7

5	1	3	6	7	9	10	2	4	8

- Children: Swap middle strings, then fill the rest.

4	3	8	1	7	9	10	5	6	2

5	1	3	6	9	2	4	7	10	8

Why does the GA work?

- $H = \star\star 10 \star 11$ is a **schema**, hyperplane partition, or similarity template describing $\{00\mathbf{10}0\mathbf{11}, 01\mathbf{10}0\mathbf{11}, \ldots\}$.

- There are $3^{\ell} - 1$ schemata in the search space, where ℓ is the length of the encoded string.

- Every string in the population belongs to $2^{\ell} - 1$ different schemata.

Implicit Parallelism

- When sampling a population, we are sampling more than just the strings of the population.
- Many competing schemata are being sampled in an **implicitly parallel** fashion when a single string is being processed.
- The new distribution of points in each schema should change according to the average fitness of the strings in the population that are in the schema.

Schema Properties

- Order: $o(H)$
  - Number of fixed positions in the schema H.
  - $o(\star 0 \star 11 \star 1) = 4$.
- Defining length: $\delta(H)$
  - Distance between leftmost and rightmost fixed positions in H.
  - $\delta(\star 0 \star 11 \star 1) = 5$.

The Building Block Hypothesis

When $f(H) > \bar{f}$, the representation (instances) of H, in generation $t+1$, is expected to increase, if $\delta(H)$ and $o(H)$ are small.

The Schema Theorem

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}\left[1 - p_c\frac{\delta(H)}{\ell-1} - p_m o(H)\right]$$

where

H	:	schema
t	:	generation number
$f(H,t)$	:	average fitness of H
$\bar{f}(t)$	:	fitness average of strings
$m(H,t)$	:	expected number of instances of H
$\delta(H)$	:	defining length of H
$o(H)$	:	order of H
p_c	:	crossover probability
p_m	:	mutation probability
ℓ	:	length of strings

Limitations of the Schema Theorem

- Computing average fitnesses to evaluate $f(H)$ at time t is often misleading in trying to predict $f(H)$ after several generations.

- The Schema Theorem cannot predict how a particular schema behaves over time.

- Other models, of a more dynamic nature, have been proposed.

Applications

- Subset Sum Problem
- 0/1 and Multiple Knapsack Problems
- Scheduling Problems
- Maximum Cut Problem
- Set Covering Problem
- Maximum Independent Set Problem
- Minimum Vertex Cover Problem
- Vertex and Edge Graph Coloring Problems
- Terminal Assignment Problem
- Encodng and Decoding of Group Codes
- DNA Fragment Assembly Problem

Different Approaches:

- Knowledge–based restrictions on search space and stochastic operators.

- In the coming examples, only the fitness function is problem-dependent.

Strategy for Fitness Functions:

- The fitness function uses a graded penalty term. The penalty is a function of the distance from feasibility.

- Infeasible strings are weaker than feasible strings.

- **Note**: The infeasible string's lifespan is quite short.

Maximum Cut Problem

Problem Instance: A weighted graph $G = (V, E)$. $V = \{1, \ldots, n\}$ is the set of vertices and E the set of edges. w_{ij} represents the weight of edge $\langle i, j \rangle$.

Feasible Solution: A set C of edges, the cut-set, containing all the edges that have one endpoint in V_0 and the other in V_1, where $V_0 \cup V_1 = V$, and $V_0 \cap V_1 = \emptyset$.

Objective Function: The cut-set weight $W = \sum_{\langle i,j \rangle \in C} w_{ij}$, which is the sum of the weights of the edges in C.

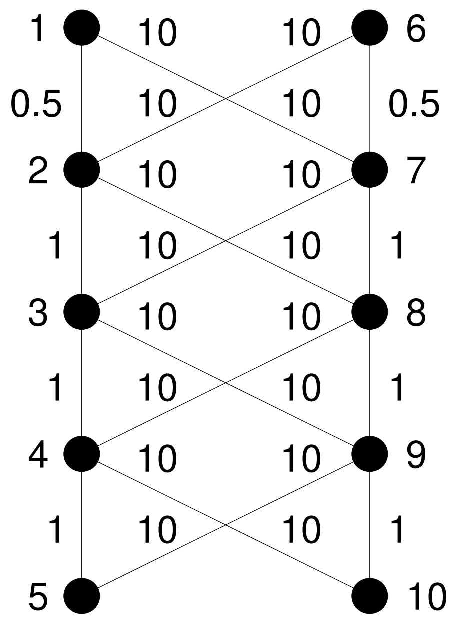
Optimal Solution: A cut-set that gives the maximum cut-set weight.

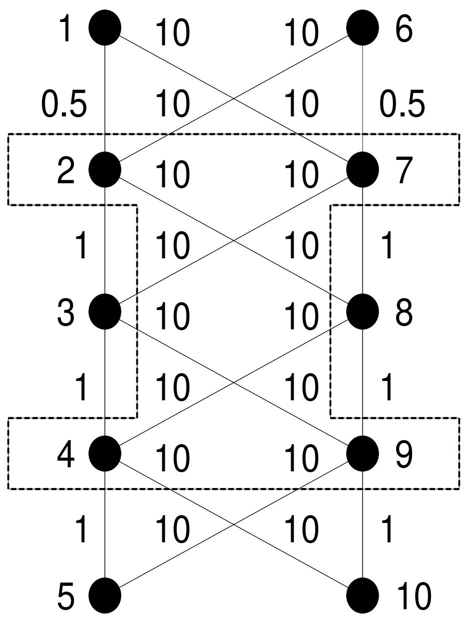
GA encoding for the Maximum Cut Problem:

- Encode the problem by using binary strings $(x_1, x_2, \ldots, x_n)$ where each digit corresponds to a vertex.
- The **fitness function** to be maximized is:

$$f(\vec{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \cdot [x_i(1-x_j) + x_j(1-x_i)] \,.$$

In absence of test problems of significantly large sizes, use scalable test problems.


1
10
10
6
0.5
10
10
0.5
2
10
10
7
1
10
10
1
3
10
10
8
1
10
10
1
4
10
10
9
1
10
10
1
5
10


1
10
10
6
0.5
10
10
0.5
2
10
10
7
1
10
10
1
3
10
10
8
1
10
10
1
4
10
10
9
1
10
10
1
5
10

- For $n = 10$, the maximum cut value is $f^* = 87$.

- Graph can be scaled up, for any even n.

- The optimal partition is described by the $n/2$-fold repetition of the bit pattern `01` (or its complement) and has objective function value:
  $f^* = 21 + 11 \cdot (n - 4)$ for $n \geq 4$.

Runs for the Maximum Cut Problem:

- "cut20-0.1" is a sparse graph, and "cut20-0.9" is a dense graph.
- Perform a total of 100 runs with population size: 50. Use 200 generations for the small graphs, and 1000 generations for the large one.

cut20-0.1		cut20-0.9		cut100	
$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N
10.11981	68	56.74007	70	1077	6
9.75907	32	56.12295	1	1055	13
		56.03820	11	1033	30
		55.84381	18	1011	35
				989	12
				967	3
				945	1

Conclusions: Maximum Cut Problem

1. With the dense and the sparse graphs, the global optimum is found by more than two thirds of the runs.

2. The genetic algorithm performs 10^4 function evaluations, and the search space is of size 2^{20}. Thus, the genetic algorithm searches only about one percent of the search space.

3. Similarly for cut100.
   The optimum is found in six of the runs.
   Average value from remaining runs:
   $\bar{f} = 1022.66$, which is about 5% from the global optimum.
   Only $(5 \cdot 10^4)/2^{100} \cdot 100\% \approx 4 \cdot 10^{-24}$ % of the search space is explored.

Minimum Tardy Task Problem

Problem instance:

Tasks:	1	2	...	n	,	i	$>$	0
Lengths:	l_1	l_2	...	l_n	,	l_i	$>$	0
Deadlines:	d_1	d_2	...	d_n	,	d_i	$>$	0
Weights:	w_1	w_2	...	w_n	,	w_i	$>$	0

Feasible solution: A one-to-one scheduling function g defined on $S \subseteq T$, $g : S \longrightarrow Z^+ \cup \{0\}$ that satisfies the following conditions for all $i, j \in S$:

1. If $g(i) < g(j)$ then $g(i) + l_i \leq g(j)$.

2. $g(i) + l_i \leq d_i$.

Objective function: The tardy task weight $W = \sum_{i \in T-S} w_i$, which is the sum of the weights of unscheduled tasks.

Optimal solution: The schedule S with the minimum tardy task weight W.

Fact S is feasible if and only if the tasks in S can be scheduled in increasing order by deadline without violating any deadline.

Example

Tasks:	1	2	3	4	5	6	7	8
Lengths:	2	4	1	7	4	3	5	2
Deadlines:	3	5	6	8	10	15	16	20
Weights:	15	20	16	19	10	25	17	18

- $S = \{1, 3, 5, 6\}$
  S is feasible with $W = 74$

- $S' = \{2, 3, 4, 6, 8\} \Rightarrow g(4) + l_4 = 5 + 7$ and $d_4 = 8$ i.e. $g(4) + l_4 > d_4$
  Therefore S' is infeasible.

GA Specifications

- S can be represented by a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ where $x_i \in \{0, 1\}$. The presence of task i in S means that $x_i = 1$.

- The **fitness function** to be minimized is the sum of three terms:

$$\begin{aligned} f(\vec{x}) &= \textstyle\sum_{i=1}^{n} w_i \cdot (1 - x_i) + (1 - s) \cdot \sum_{i=1}^{n} w_i \\ &+ \textstyle\sum_{i=1}^{n} w_i x_i \cdot \mathbf{1}_{R+} \Big(l_i + \sum_{j=1}^{i-1} l_j x_j - d_i \Big). \end{aligned}$$

  In the third term, x_j is for schedulable jobs only. The whole term keeps checking the string to see if a task could have been scheduled. It makes use of the indicator function:

$$\mathbf{1}_A(t) = \begin{cases} 1 & \text{if } t \in A \\ 0 & \text{otherwise.} \end{cases}$$

  $s = 1$ when $\vec{x}$ is feasible, and $s = 0$ when $\vec{x}$ is infeasible.

Scalable Problem Instance

- Construct a problem instance of size $n = 5$ that is scalable.

Tasks:	1	2	3	4	5
Lengths:	3	6	9	12	15
Deadlines:	5	10	15	20	25
Weights:	60	40	7	3	50

- To construct a **MTTP** instance of size $n = 5t$ from above model:

  The first five tasks of the large problem are identical to the 5-model problem instance.

  The length l_j, deadline d_j, and weight w_j of the j^{th} task, for $j = 1, 2, \ldots, n$, is given by: $l_j = l_i$, $d_j = d_i + 24 \cdot m$ and

  $$w_j = \begin{cases} w_i & \text{if } j \equiv 3 \bmod 5 \text{ or } j \equiv 4 \bmod 5 \\ (m+1) \cdot w_i & \text{otherwise ,} \end{cases}$$

  where $j \equiv i$ mod 5 for $i = 1, 2, 3, 4, 5$ and $m = \lfloor j/5 \rfloor$.

  The tardy task weight for the globally optimal solution of this problem is $2 \cdot n$.

Runs for the Minimum Tardy Task Problem

mttp20

- Population Size: 50
  Number of Generations: 200
  Crossover: Uniform
  Probability of Crossover: 0.6
  Mutation: Swap
  Probability of Mutation: 0.05

- A total of 100 runs is performed.

$f_1(\vec{x})$	$d(\vec{x}, opt)$	N
41	0	88
46	3	11
51	1	1

mttp100

- Scalable problem of size $n = 100$.

- Population Size: 200
  Number of Generations: 100
  Crossover: Uniform
  Probability of Crossover: 0.6
  Mutation: Swap
  Probability of Mutation: 0.05
  Note: About $1.6 \cdot 10^{-23}$ % of the search space is investigated.

- $f_2(\vec{x})$ replaces infeasible strings by feasible ones.

$f_1(\vec{x})$	$d(\vec{x}, opt)$	N	$f_2(\vec{x})$	$d(\vec{x}, opt)$	N
200	0	56	200	0	35
240	1	2	–	–	–
243	2	29	243	2	44
276	3	3	276	3	4
280	4	1	280	4	1
–	–	–	296	4	1
329	5	9	329	5	14
–	–	–	379	5	1

Terminal Assignment Problem

Problem Instance:

Terminals:	$l_1, l_2, \ldots, l_T$
Weights:	$w_1, w_2, \ldots, w_T$
Concentrators:	$r_1, r_2, \ldots, r_C$
Capacities:	$p_1, p_2, \ldots, p_C$

Feasible Solution: Vector: $\vec{x} = x_1\, x_2\, \ldots\, x_T$

$x_i = j$: i^{th} terminal assigned to j.
Assign all terminals to concentrators without exceeding the concentrators' capacities.

Objective Function: A function $Z(\vec{x}) = \sum_{i=1}^{T} cost_{ij}$, where $cost_{ij}$ is the distance between terminal i and concentrator j.

Optimal Solution: A feasible vector $\vec{x}$ that yields the smallest possible $Z(\vec{x})$.

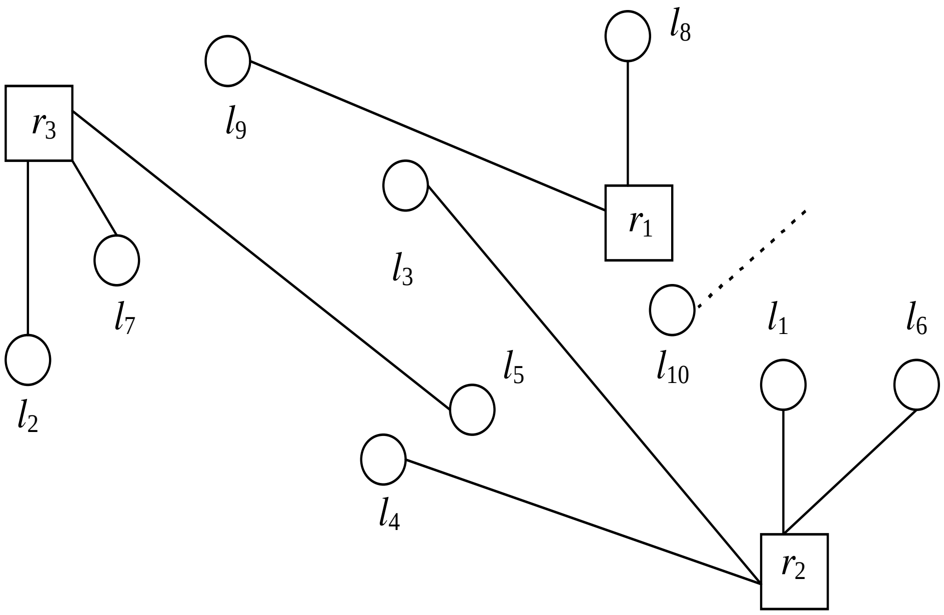
Encoding for TA:

Use non-binary strings: $x_1 \, x_2 \, \ldots \, x_T$

The value of x_i is the concentrator to which the i^{th} terminal is assigned.

Example:

1	2	3	4	5	6	7	8	9	10
2	3	2	2	3	2	3	1	1	2



Fitness Function:

It is the sum of two terms:

1. the objective function which calculates the total cost of all connections.

2. a penalty function used to penalize infeasible strings. It is the sum of two terms:

   - **offset term:**
     the product of the number of terminals and the maximum distance on the grid.

   - **graded term:**
     the product of the sum of excessive load of concentrators and the number of concentrators that are in fact overloaded.

Experimental Runs

- Ten problem instances of 100 terminals each.
  Each concentrator has capacity between 15 and 25.
  The number of concentrators is between 27 and 33.

- Number of generations: 20,000
  Population size: 500
  Crossover rate: 0.6
  Mutation rates: Between 0.025 and 0.1 for LibGA and for GeneSyS.

- The results for the Genetic Algorithm are the best obtained after 20,000 generations on each problem instances.

- Probably for 100_1, 100_2, and 100_3, the solutions obtained by LibGA (and Procedure Greedy) are the global optima.

Problem Instances	Greedy Algorithm	Genetic Algorithms	
		GENEsYs	LibGA
100_1	**928**	963	**928**
100_2	**935**	973	**935**
100_3	**846**	881	**846**
100_4	1076	1123	**1069**
100_5	1116	1178	**1111**
100_6	1071	1106	**1070**
100_7	1315	1404	**1279**
100_8	1211	1225	**1186**
100_9	1088	1169	**1065**
100_10	902	977	**901**

Grouping Problems

- Many problems consist in partitioning a given set U of items into a collection of pairwise disjoint subsets U_j under some constraint.

- In essence, we are asked to distribute the elements of U into small groups U_j. Some distributions (i.e., groupings) are forbidden.

- Some examples: Bin packing problem, terminal assignment, graph coloring.

- The objective is to minimize a cost function over the set of all possible legal groupings.

The Edge Coloring Problem

Problem Instance

A graph $G = (V, E)$,
$V = \{v_1, \ldots, v_n\}$ is the set of vertices,
$E = \{e_1, \ldots, e_m\}$ is the set of edges.

Feasible Solution

Vector $\vec{y} = y_1\, y_2\, \cdots\, y_m$,
$y_i = j$ means that the i^{th} edge e_i is given color j, $1 \leq j \leq m$.
The components of $\vec{y}$ are essentially the colors assigned to the m edges of G, where:

- Two adjacent edges have different colors.
- The colors are denoted by $1, 2, \ldots, q$, where $1 \leq q \leq m$.

Objective function

For a feasible vector $\vec{y}$, let $P(\vec{y})$ be the number of colors used in the coloring of the edges represented by $\vec{y}$. More precisely,

$$P(\vec{y}) = max\{y_i \mid y_i \text{ is a component of } \vec{y}\}.$$

Optimal solution

A feasible vector $\vec{y}$ that yields the smallest $P(\vec{y})$.

Applications

Practical applications include scheduling problems, partitioning problems, timetabling problems, and wavelength-routing in optical networks.

NP-Hard

Holyer proved that the edge coloring problem is NP-hard.

Definition

Δ is maximum vertex degree of G.
$\chi`(G)$: the chromatic index of G, is the minimum number of colors required to color the edges of G.

Vizing's Theorem

If G is a nonbipartite, simple graph, then $\chi`(G) = \Delta$ or $\chi`(G) = \Delta + 1$.

Edge-Coloring Bipartite Graphs

- Journal of Algorithms, February 2000
  Article by A. Kapoor and R. Rizzi:

$$O(m \log_2 \Delta + \frac{m}{\Delta} \log_2 \frac{m}{\Delta} {\log_2}^2 \Delta)$$

- SIAM, Journal on Computation, 1982
  Article by Hopcroft and Cole:

$$O(m \log_2 \Delta + \frac{m}{\Delta} \log_2 \frac{m}{\Delta} {\log_2}^3 \Delta)$$

The Genetic Algorithm

Encoding:

Use non-binary strings of length m (number of edges):

$$y_1\ y_2\ \cdots\ y_m.$$

The value of y_j represents the color assigned to the j^{th} edge.

For example, if a graph has 12 edges then the following string is a possible coloring, where edge e_1 has color number 5, edge e_2 has color number 3, and so on.

1	2	3	4	5	6	7	8	9	10	11	12
5	3	5	5	1	3	2	1	4	4	1	3

The GA Operators

- generational genetic algorithms
- roulette wheel selection
- uniform crossover
- uniform mutation

Each string in the population occupies a slot on the roulette wheel of size *inversely* proportional (since the edge coloring problem is a minimization problem) to the fitness value.

Fitness Function

The fitness function of $\vec{y} = y_1 y_2 \dots y_m$ is given by:

$$f(\vec{y}) = P(\vec{y}) + s[(\Delta + 1) + T(\vec{y})]$$

It is the sum of two terms:

1. the objective function $P(\vec{y})$, which calculates the total number of colors used to color the edges of the graph.

2. a penalty term used to penalize infeasible strings. It is the sum of two parts:

   - The first part is the maximum vertex degree to which one is added, i.e., $\Delta + 1$. It is an offset term.

   - The second part of the penalty term, $T(\vec{y})$, is the number of violated pairs.

The Grouping Genetic Algorithm

Encoding

To capture the grouping entity of the problem, the simple (standard) chromosome is augmented with a group part.

For example, the chromosome of our hypothetical 12-edge graph:

5	3	5	5	1	3	2	1	4	4	1	3

A	5	8	11	B	7	C	2	6	12	D	9	10	E	1	3	4

The colors are labeled with letters, the edges – with numbers.

The first half is the object part (the entire chromosome of the previous section), the second half is the group part.

Fitness Function

The fitness function of a string considers the object part of the string only and computes the fitness value as explained before.

Crossover Operator

Two parents are randomly chosen from the mating pool and two crossing sites are chosen for each parent:

A	5	8	11	B	7	C	2	6	12	D	9	10	E	1	3	4

A	1	3	6	7	B	8	11	C	2	9	D	10	12	E	4	5

Inject the entire group part between the two crossing sites of the second parent into the first parent at the beginning of its crossing site:

A	5	8	11	B	8	11	C	2	9	D	10	12	B	7	C	2	6	12	D	9	10	E	1	3	4

Dealing with Redundancy

Resolve redundancy by eliminating groups B, C, and D from their old membership (i.e., from the first parent string):

A	5	8	11	B	8	11	C	2	9	D	10	12	E	1	3	4

Remove duplicate edges (8 and 11) from their old membership (from the first parent string):

A	5	B	8	11	C	2	9	D	10	12	E	1	3	4

Use the first–fit heuristic to reassign lost edges (6 and 7): assign the edge to the first group that is able to service it, or to a randomly selected group if none is available.

Reversing the roles of the two parents allows the construction of a second offspring.

The Mutation Operator

Probabilistically remove some group from the chromosome and reassign its edges.

Each edge is assigned to the first group that is able to take it.

After crossover and mutation are performed on the group part, the object part is rearranged accordingly to reflect the changes that the group part has undergone.

Benchmarks

- Twenty eight simple graphs with the number of edges between 20 and 2596:
  - Five book graphs from the Stanford GraphBase.
  - Two miles graphs from the Stanford GraphBase.
  - Nine queen graphs from the Stanford GraphBase.
  - Four problem instances based on the Mycielski transformation from the DIMACS collection.
  - Eight complete graphs.
- Fifteen multigraphs generated by repeatedly assigning edges to randomly chosen pairs of vertices until each vertex has a fixed degree.

  We denote the multigraphs we generated by rexmg10_x, rexmg20_x, and rexmg30_x, where 10, 20 and 30 are the number of vertices and x is the degree and has the values of 10, 20, 30, 40 and 50.

Tracking a Criminal Suspect

Genetic Programming

- The structures undergoing adaptation in Genetic Programming are general, hierarchical computer programs of dynamically varying size and shape.

- The search space consists of all possible computer programs composed of **functions** and **terminals** appropriate to the problem domain.

Conclusion

1. Simple Genetic & Grouping Genetic Algorithms. Genetic Programming. Other Evolutionary-based Algorithms: Evolution Strategies and Evolutionary Programming.

2. Hybrid Algorithms and Parallel Genetic Algorithms.

3. What problems are suitable for Genetic Algorithms?
   Weak classification of problems between hard and easy.

4. The study of Genetic Algorithms is still in its embryonic stage.

References

1. [CJ91] C. Caldwell and V. Johnson. **Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm**, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, CA, 1991, pp. 416-421.

2. [CW93] A. Corcoran and R. Wainwright. **LibGA: A User-friendly Workbench for Ordered-based Genetic Algorithm Research**, *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, 1993, pp. 111-117.

3. [Fal93] E. Falkenauer. **The Grouping Genetic Algorithms - Widening the Scope of Genetic Algorithms**, *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 33, 1993, pp. 416-421.

4. [Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, NY, 1989.

5. [Hol75] J. Holland. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

6. [KB90] S. Khuri and A. Batarekh. **Heuristics for the Integer Knapsack Problem**, *Proceedings of the X^{th} International Conference of the Chilean Computer Science Society*, Santiago, Chile, 1990, pp. 161-172.

7. [KBH94] S. Khuri, T. Bäck and J. Heitkötter. **An Evolutionary Approach to Combinatorial Optimization Problems**, *Proceedings of the* 22^{nd} *ACM Computer Science Conference*, editor, D. Cizmar, ACM Press, NY, 1994, pp. 66-73.

8. [KC97] S. Khuri and T. Chiu. **Heuristic Algorithms for the Terminal Assignment Problem**, *Proceedings of the 1997 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, 1997.

9. [KWS00] S. Khuri, T. Walters and Y. Sugono. **A Grouping Genetic Algorithm for Coloring the Edges of Graphs**, *Proceedings of the 2000 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, 2000.

10. [Koz92] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, MIT Press, 1992.