Genetic Algorithms: Applications

Sami Khuri
Dept. of Math. & Computer Science
San José State University
One Washington Square
San José, CA 95192-0103, U.S.A.
khuri@jupiter.sjsu.edu
June 26, 1996

Today's Topics

- Brief 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.

Applications done with the collaboration of:

- **Aida Batarekh**
  Stanley Morgan, New York

- **Thomas Bäck**
  Informatik Centrum Dortmund

- **Teresa Chiu**
  San José State University

- **Jörg Heitkötter**
  EUnet Deutschland GmbH

- **My Hoang**
  Stanford University

- **Martin Schütz**
  Universität Dortmund

Programs used in Applications

- **GENEsYs 1.0** by Thomas Bäck
  It is based on Grefenstette's popular software package GENESIS.

- **LibGA**
  Developed by J. Corman and R. Wainwright from the University of Tulsa.

Problem Instances Obtained From

- **OR-Library**
  Distribution test problems maintained by J. E. Beasley.

- **Knapsack Problems** by Martello and Toth's

- **Multi-Dimensional 0/1 Knapsack Problems**
  H. M. Weingartner and D. N. Ness.

- **An Approach to Linear Programming with 0/1 Variables**
  S. Senyu and Y. Toyoda.

Application Problems

- **Zero/One Knapsack Problem**
- **Subset Sum Problem**
- **Maximum Cut Problem**
- **Minimum Tardy Task Problem**
- **Zero/One Multiple Knapsack Problem**
- **Minimum Vertex Cover Problem**
- **Maximum Independent Set Problem**
- **Bin Packing Problem**
- **Maximum Set Covering Problem**
- **Terminal Assignment Problem**

Remark

- Other approaches:
  knowledge-based restrictions on search space and stochastic operators.
  Here, only the fitness function is problem-dependent.

Strategy for Fitness Functions

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

- Infeasible strings are weaker than feasible strings.

Remarks

- The infeasible string's lifespan is quite short.

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

Formal Definition: **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:

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

Test Problems for the Maximum Cut Problem

- Consider randomly generated graphs of small sizes. The global optimum can be *a priori* computed by complete enumeration.

- Construct two graphs of $n = 20$ vertices each.
  Vary the probability of placing an edge between arbitrary pairs of vertices.
  Weights of edges are uniformly chosen in the range $[0, 1]$.

- a) Sparse graph: “cut20-0.1”.
  Probability of an edge is 0.1
  b) Dense graph: “cut20-0.9”.
  Probability of an edge is 0.9.

- 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 $f_2^* = 21 + 11 \cdot (n - 4)$ for $n \geq 4$.
- One might be tempted to believe that the regularity in the formulation of the problem instance might favor the workings of genetic algorithms. To defuse any doubts, we introduce a preprocessing step which consists in randomly renaming the vertices of the problem instance. So consecutive bit positions no longer correspond to vertices that are close to each other within the graph itself.
- "cut100" is of size $n = 100$.

Runs for the **Maximum Cut Problem**

- Perform a total of 100 experimental runs for each of the three graphs.

- $\mu = 50$.
  The genetic algorithm runs for 200 generations for the small graphs and for 1000 generations for the large one.

- First row: optimal solutions.

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**

- The genetic algorithm is not affected by the “density” of the graphs with 20 nodes.

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

- 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.

- 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.

Formal Definition: **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.

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 minimize:

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

  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

- Generate a problem of size $n = 100$ ("mttp100").

- A total of 100 experiments is performed.
  The number of function evaluations per run is 10^4 for mttp20, ($\mu = 50$, and 200 generations). and ($\mu = 100$, and 200 generations) $2 \cdot 10^5$ for mttp100
  (about $1.6 \cdot 10^{-23}$ % of the search space).

mttp20		mttp100	
$f(\vec{x})$	N	$f(\vec{x})$	N
41	79	200	67
46	19	243	7
51	2	329	24
		452	1
		465	1

- The optimum is located within two third of the runs.
  The string of value 329 differs from the optimum by five bits located at positions widely spread over the bit string.
  Hamming distance between the strings of values 46 and 41 is 3.

Formal Definition: **Minimum Set-Covering Problem**

Problem instance:

A set of objects $E = \{e_1, \ldots, e_m\}$ and a family of subsets of E, $F = \{S_1, \ldots, S_n\}$, such that $\cup_{i=1}^{n} S_i = E$. Each S_i costs c_i where $c_i > 0$ for $i = 1, \ldots, n$. Or equivalently, a 0/1 $m \times n$ matrix

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

such that $\sum_{j=1}^{n} a_{ij} \geq 1$ for $i = 1, 2, \ldots, m$.

Feasible solution:

A vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ where $x_i \in \{0, 1\}$, such that

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1$$

for $i = 1, 2, \ldots, m$.

Objective function:

A cost function $P(\vec{x}) = \sum_{j=1}^{n} c_j x_j$ where $\vec{x} = (x_1, x_2, \ldots, x_n)$ is a feasible solution.

Optimal solution:

A feasible vector that gives the minimum cost, i.e., a vector that minimizes the objective function.

Fitness Function

- $B = \sum_{i=1}^{n} c_i$.
  U: the set of elements from E for which the subsets represented by $\vec{x}$ have failed to cover.
  The function considers the minimum cost required to transform the infeasible string into a feasible one:

$$f(\vec{x}) = \sum_{i=1}^{n} c_i x_i + s(B + \sum_{i=1}^{m} c_i^*)$$

  where

$$s = \begin{cases} 0 & , \quad \text{if } \vec{x} \text{ is feasible} \\ 1 & , \quad \text{if } \vec{x} \text{ is infeasible} \end{cases}$$

  and

$$c_i^* = \begin{cases} \min_{e_i \in U}\{c_j \mid e_i \in S_j\} & , \quad \text{if } e_i \in U \\ 0 & , \quad \text{if } e_i \notin U \end{cases}$$

- **Repair heuristic:**
  Transform an infeasible string into a feasible one by changing $x_j = 0$ to $x_j = 1$ when $c_i^* = \min_{e_i \in U}\{c_j \mid e_i \in S_j\}$ is calculated.
  The repaired string is always feasible; the fitness function in this case is simply $f'(\vec{x}) = \sum_{i=1}^{n} c_i x_i$.

GA Stochastic Operators and Test Problems

- The GA is used with the one-point, two-point, uniform, and fusion crossover operators.

- **Fusion crossover:**
  If $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$ are the parent strings with fitness $f_{\vec{a}}$ and $f_{\vec{b}}$, then the offspring $\vec{c} = (c_1, c_2, \ldots, c_n)$ is determined in the following fashion.

- **for all** $i \in \{1, \ldots, n\}$
  if $a_i = b_i$
  then $c_i := a_i$;
  else $c_i := \begin{cases} a_i & \text{with probability } p = \frac{f_{\vec{b}}}{f_{\vec{a}}+f_{\vec{b}}} \\ b_i & \text{with probability } 1-p \end{cases}$

- The crossover rate is 0.6
  Mutation rate $p_m = 1/n$.
  Population size $\lambda = 100$.
  Each experiment was run for $5,000$ generations.

Problem	m	n	Density (%)
4.1–4.10	200	1000	2
5.1–5.10	200	2000	2
6.1–6.5	200	1000	5
a.1–a.5	300	3000	2
b.1–b.5	300	3000	5
c.1–c.5	400	4000	2
d.1–d.5	400	4000	5
e.1–e.5	50	500	20

Experimental Results

Prob	1-pt	2-pt	unif	fusion	greedy	bt kn
4.1	439	440	440	438	436	429
4.2	536	551	525	556	533	512
4.3	529	532	532	532	540	516
4.4	528	517	508	512	512	494
4.5	522	526	518	527	520	512
4.6	568	579	572	568	605	560
4.7	444	437	437	437	447	430
4.8	504	504	509	499	525	492
4.9	678	689	686	675	671	641
4.10	549	538	553	553	533	514
5.1	274	271	270	274	269	253
5.2	318	319	319	320	330	302
5.3	238	235	235	239	232	226
5.4	247	245	245	244	250	242
5.5	219	219	219	219	218	211
5.6	229	236	238	232	227	213
5.7	313	310	309	309	310	293
5.8	302	309	304	306	311	288
5.9	301	304	298	292	292	279
5.10	275	276	276	277	278	265
6.1	146	145	147	147	144	138
6.2	154	154	153	154	157	146
6.3	154	150	151	151	157	145
6.4	140	133	136	133	141	131
6.5	180	180	177	179	186	161
a.1	255	261	260	260	264	253
a.2	270	268	267	267	272	252
a.3	247	247	244	245	245	232
a.4	244	246	249	247	251	234
a.5	240	239	239	239	248	236

Experimental Results - Continued

Prob	1-pt	2-pt	unif	fusion	greedy	bt kn
b.1	77	77	79	75	74	69
b.2	87	83	84	81	78	76
b.3	84	85	83	83	85	80
b.4	84	82	84	84	83	79
b.5	72	75	72	77	75	72
c.1	234	235	234	235	239	227
c.2	231	232	226	233	225	219
c.3	266	261	263	264	259	243
c.4	237	234	238	237	240	219
c.5	220	219	222	219	219	215
d.1	62	61	61	62	68	60
d.2	72	72	70	70	71	66
d.3	77	79	79	79	80	72
d.4	66	64	66	66	67	62
d.5	64	65	65	64	66	61
e.1	8	7	7	8	5	5
e.2	7	6	6	7	5	5
e.3	7	7	8	6	5	5
e.4	6	7	7	7	6	5
e.5	7	8	7	7	5	5

Conclusions on the **MSCP**

- Repairing infeasible strings according to the minimum cost principle yields better results - the final values are close to the best known solutions.
- The repair procedure competes well with the greedy heuristic and yields better solutions in most cases.
- The choice of the crossover operator has no significant impact on the quality of the results.
- Our simple greedy heuristic always outperforms the genetic algorithm without repairing. (Table not shown here)

Conclusion and Future Work

- Various other genetic algorithms and stochastic operators.
- Various other evolutionary algorithms.
- More experimentation with repair mechanism.
- Seeding the initial population seems to work well.
- Hybrid algorithms have proven to be very efficient.
- The technology behind the parallelization of evolutionary algorithms is growing at a very rapid pace.