Genetic Algorithms

Genetic Algorithms

- **Genetic Algorithms** (GAs) were **developed by Prof. John Holland** and his students at the University of Michigan during the 1960s and 1970s.

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional
Survival selection	Generational

Table 3.1. Sketch of the simple GA

Genetic Algorithms

- Maximizing the values of **x^2** for x in the range **0-31**.

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional
Survival selection	Generational

Table 3.1. Sketch of the simple GA

$$Prob_i = f_i / \sum f_j$$

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Table 3.2. The x^2 example, 1: initialisation, evaluation, and parent selection

Genetic Algorithms

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Table 3.2. The x^2 example, 1: initialisation, evaluation, and parent selection

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3.3. The x^2 example, 2: crossover and offspring evaluation

Genetic Algorithms

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3.3. The x^2 example, 2: crossover and offspring evaluation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	28	784
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	20	400
Sum				2538
Average				634.5
Max				784

Table 3.4. The x^2 example, 3: mutation and offspring evaluation

Representation of Individuals

1. **Binary Representations**

2. **Integer Representations**

3. **Real-Valued or Floating-Point Representation**

4. **Permutation Representations**

Mutation

1. Mutation for Binary Representations

Mutation Operators for Integer Representations

2. Mutation Operators for Integer Representations

Random Resetting

• "Bit-flipping" mutation of binary encodings is extended to "random resetting"

• With probability Pm a new value is chosen at random from the set of permissible values in each position.

Creep Mutation

• Tended to make small changes relative to the range of permissible values.

• Designed for ordinal attributes and works by adding a small (positive or negative) value to each gene with probability p.

Mutation Operators for Floating-Point Representations

2. Mutation Operators for Floating-Point Representations

Change the allele value of **each gene randomly within its domain** given by a lower *Li* and upper *Ui* bound, resulting in the following transformation:

$$\langle x_1, \ldots, x_n \rangle \rightarrow \langle x'_1, \ldots, x'_n \rangle, \quad \text{where} \quad x_i, x'_i \in [L_i, U_i].$$

Uniform Mutation

- The values of x' are drawn **uniformly randomly** from *[Li, Ui]*
- Analogous to bit-flipping for binary encodings and the random resetting sketched for integer encodings.
- Position wise mutation probability

Mutation Operators for Floating-Point Representations

2. Mutation Operators for Floating-Point Representations

Change the allele value of **each gene randomly within its domain** given by a lower *Li* and upper *Ui* bound, resulting in the following transformation:

$$\langle x_1, \ldots, x_n \rangle \rightarrow \langle x'_1, \ldots, x'_n \rangle, \quad \text{where} \quad x_i, x'_i \in [L_i, U_i].$$

Non-uniform Mutation with a Fixed Distribution

- Analogous to the creep mutation.
- Adding to the current gene value an amount drawn randomly from a Gaussian distribution with mean zero and user-specified standard deviation, and then curtailing the resulting value to the range [Li, Ui] if necessary.

Mutation Operators for Permutation Representations

- Swap Mutation

- Insert Mutation

- Scramble Mutation

- Inversion Mutation

Recombination

- Recombination, the process whereby a new individual solution is created from the **information contained within two** (or more) **parent** solutions.

- **Recombination Operators for Binary Representations**

- One-Point Crossover

Recombination Operators for Binary Representations

- N-Point Crossover

Fig. 3.7. n-point crossover: $n = 2$

Recombination Operators for Binary Representations

- Uniform Crossover
- In each position, if the **value is below a parameter p** (usually 0.5), the **gene is inherited from** the first parent; otherwise from the second. The second offspring is created using the **inverse mapping**.

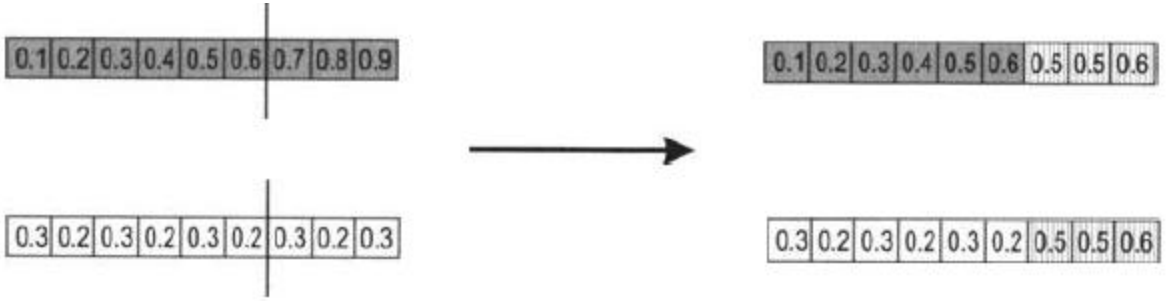
Fig. 3.8. Uniform crossover. In this example the array [0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36] of random variables drawn uniformly from [0,1) was used to decide inheritance

Recombination Operators for Integer Representations

- Same set of operators as for binary representations.

Recombination Operators for Floating-Point Representations

- Arithmetic Recombination
  - Three types of arithmetic recombination
- Simple Recombination



Child 1: $\langle x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle$.

Child 2 is analogous, with x and y reversed

Recombination Operators for Floating-Point Representations

- Single Arithmetic Recombination
  - Pick a **random allele k**. At that position, take the **arithmetic average of the two parents**.

Child 1: $\langle x_1, \ldots, x_{k-1}, \alpha \cdot y_k + (1-\alpha) \cdot x_k, x_{k+1}, \ldots, x_n \rangle$.

The second child is created in the same way with x and y reversed

Recombination Operators for Floating-Point Representations

- Whole Arithmetic Recombination

This is the most commonly used operator and works by taking the weighted sum of the two parental alleles for each gene, i.e.:

$$\text{Child 1} = \alpha \cdot \bar{x} + (1 - \alpha) \cdot \bar{y}, \qquad \text{Child 2} = \alpha \cdot \bar{y} + (1 - \alpha) \cdot \bar{x}.$$

Fig. 3.11. Whole arithmetic recombination: $\alpha = 1/2$

Recombination Operators for Permutation Representations

- Partially Mapped Crossover

1. Choose two crossover points at random, and copy the segment between them from the first parent (P1) into the first offspring.
2. Starting from the first crossover point look for elements in that segment of the second parent (P2) that have not been copied.
3. For each of these (say i), look in the offspring to see what element (say j) has been copied in its place from P1.
4. Place i into the position occupied j in P2, since we know that we will not be putting j there (as we already have it in our string).
5. If the place occupied by j in P2 has already been filled in the offspring by an element k, put i in the position occupied by k in P2.
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2, and the second child is created analogously with the parental roles reversed.

Recombination Operators for Permutation Representations

- Partially Mapped Crossover

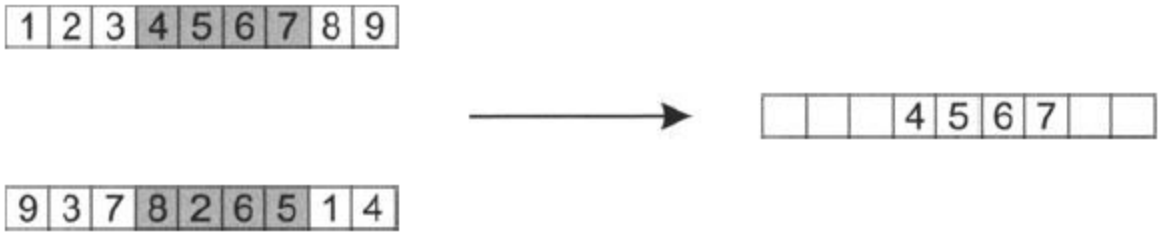


Fig. 3.12. PMX, step 1: copy randomly selected segment from first parent into offspring

Recombination Operators for Permutation Representations

- Partially Mapped Crossover

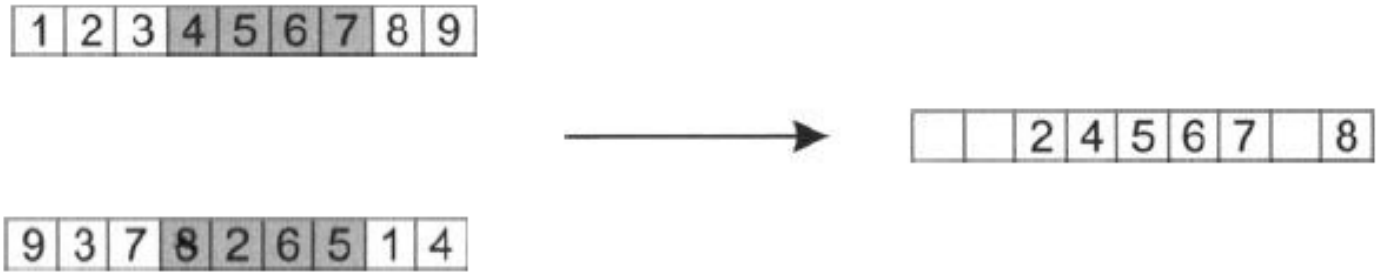


Fig. 3.13. PMX, step 2: consider in turn the placement of the elements that occur in the middle segment of parent 2 but not parent 1. The position that 8 takes in P2 is occupied by 4 in the offspring, so we can put the 8 into the position vacated by the 4 in P2. The position of the 2 in P2 is occupied by the 5 in the offspring, so we look first to the place occupied by the 5 in P2, which is position 7. This is already occupied by the value 7, so we look to where this occurs in P2 and finally find a slot in the offspring that is vacant – the third. Finally, note that the values 6 and 5 occur in the middle segments of both parents.

Recombination Operators for Permutation Representations

- Partially Mapped Crossover

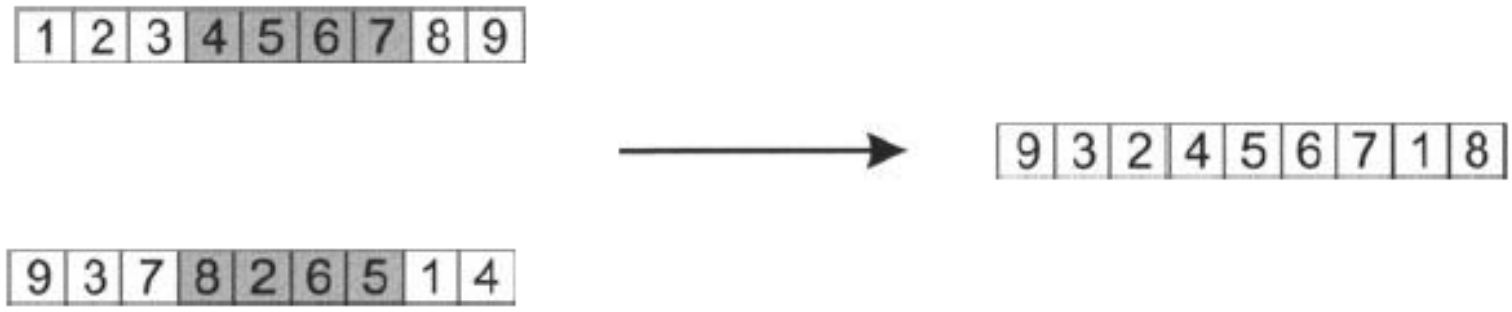


Fig. 3.14. PMX, step 3: copy remaining elements from second parent into same positions in offspring

Recombination Operators for Permutation Representations

- **Edge Crossover**
- Edge crossover is based on the idea that an **offspring should be created as far as possible using only edges that are present in one or more parent**.

- Most commonly used version: edge-3 crossover after **Whitley**, which is designed to ensure that common edges are preserved.

Recombination Operators for Permutation Representations

- **Edge Crossover**

1. Construct edge table
2. Pick an initial element at random and put it in the offspring
3. Set the variable *current_element* = *entry*
4. Remove all references to *current_element* from the table
5. Examine list for *current_element*
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
6. In the case of reaching an empty list, the other end of the offspring is examined for extension; otherwise a new element is chosen at random

Recombination Operators for Permutation Representations

- **Edge Crossover**

1. Let K be the empty list Let N be the first node of a random parent.
2. While Length(K) < Length(Parent):
    1. K := K, N (append N to K)
    2. Remove N from all neighbor lists
    3. If N's neighbor list is non-empty
    4. then let N* be the neighbor of N with the fewest neighbors in its list (or a random one, should there be multiple)
    5. else let N* be a randomly chosen node that is not in K
    6. N := N*

Recombination Operators for Permutation Representations

- **Edge Crossover** [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Table 3.5. Edge crossover: example edge table

Recombination Operators for Permutation Representations

- **Edge Crossover** [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Choices	Element selected	Reason	Partial result
All	1	Random	[*1*]
2,5,4,9	5	Shortest list	[*1 5*]
4,6	6	Common edge	[*1 5 6*]
2,7	2	Random choice (both have two items in list)	[*1 5 6 2*]
3,8	8	Shortest list	[*1 5 6 2 8*]
7,9	7	Common edge	[*1 5 6 2 8 7*]
3	3	Only item in list	[*1 5 6 2 8 7 3*]
4,9	9	Random choice	[*1 5 6 2 8 7 3 9*]
4	4	Last element	[*1 5 6 2 8 7 3 9 4*]

Table 3.6. Edge crossover: example of permutation construction

Recombination Operators for Permutation Representations

- **Edge Crossover** **CABDEF** and **ABCEFD**

Recombination Operators for Permutation Representations

- **Edge Crossover** **CABDEF** and **ABCEFD**

Answer: ABDFCE

Recombination Operators for Permutation Representations

- **Order Crossover [D**esigned by Davis for order-based permutation]

1. Choose two crossover points at random, and copy the segment between them from the first parent (P1) into the first offspring.
2. Starting from the second crossover point in the second parent, copy the remaining unused numbers into the first child in the order that they appear in the second parent, wrapping around at the end of the list.
3. Create the second offspring in an analogous manner, with the parent roles reversed.

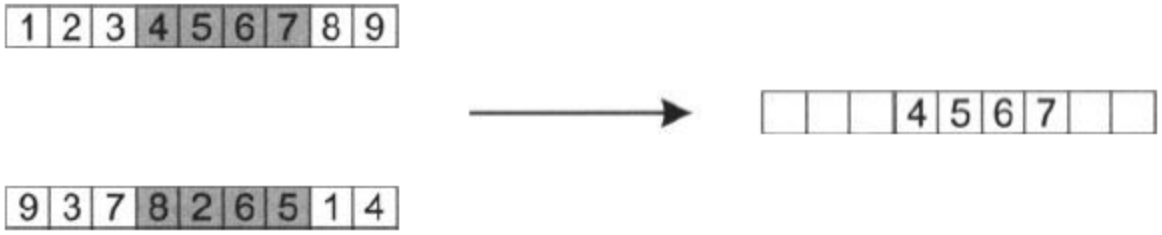


Fig. 3.15. Order crossover, step 1: copy randomly selected segment from first parent into offspring

Recombination Operators for Permutation Representations

- **Order Crossover**

1. Choose two crossover points at random, and copy the segment between them from the first parent (P1) into the first offspring.
2. Starting from the second crossover point in the second parent, copy the remaining unused numbers into the first child in the order that they appear in the second parent, wrapping around at the end of the list.
3. Create the second offspring in an analogous manner, with the parent roles reversed.

Fig. 3.16. Order crossover, step 2: copy rest of alleles in order they appear in second parent, treating string as toroidal

Recombination Operators for Permutation Representations

- **Cycle Crossover**
- The operator works by **dividing the elements into cycles**.
- A **cycle is a subset of elements** that has the property that **each element always occurs paired** with another element of the same cycle when the two parents are aligned.
- Having divided the permutation into cycles,
- The **offspring are created by selecting alternate cycles** from each parent.

Recombination Operators for Permutation Representations

- **Cycle Crossover**
- The procedure for constructing cycles is as follows:

1. Start with the first unused position and allele of P1
2. Look at the allele in the *same position* in P2
3. Go to the position with the *same allele* in P1
4. Add this allele to the cycle
5. Repeat steps 2 through 4 until you arrive at the first allele of P1

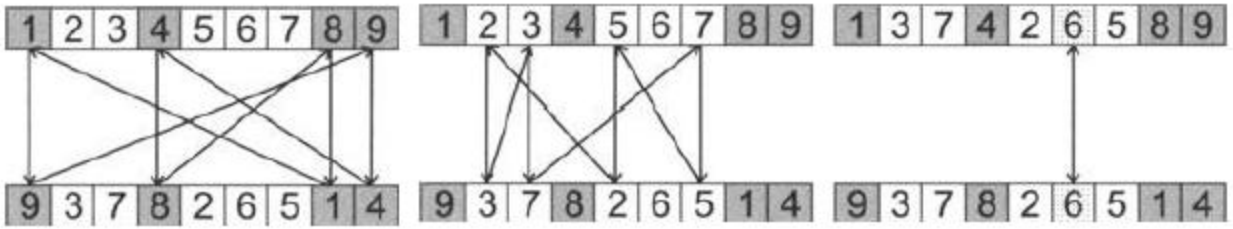


Fig. 3.17. Cycle crossover, step 1: identification of cycles

Recombination Operators for Permutation Representations

- **Cycle Crossover**

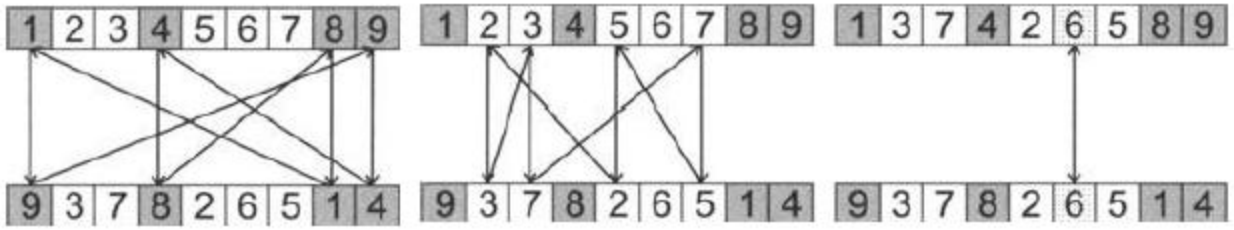
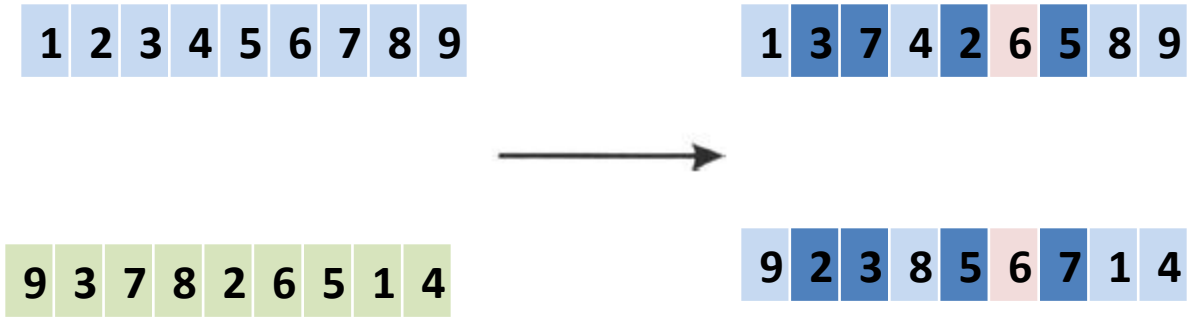


Fig. 3.17. Cycle crossover, step 1: identification of cycles



Population Model

- **Generational model**
- In each generation we begin with a population of size from which a mating pool of parents is selected.
- Next, offspring are created from the mating pool by the application of variation operators, and evaluated.
- After each generation, the whole population is replaced by its offspring, which is called the "next generation" .

μ parents $\quad$ $\lambda\ (= \mu)$ offspring

Population Model

- **Steady-state model**
- In the steady state model, the entire population is not changed at once, but rather a **part of it**.
- Select from **(λ+μ)**

- Generational gap
  - If **λ** parents and **μ** offspring Generation gap = **λ/μ**

Parent Selection

- Fitness Proportional Selection

  - The selection probability depends on the absolute fitness value of the individual compared to the absolute fitness values of the rest of the population.

$$f_i / \sum_{j=1}^{\mu} f_j$$

  - When fitness values are all very close together, there is almost no selection pressure.
  - Premature convergence.

Parent Selection

- Ranking Selection
  - It preserves a constant selection pressure by sorting the population on the basis of fitness, and then allocating selection probabilities to individuals according to their rank, rather than according to their actual fitness values.

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}.$$

$$P_{exp-rank}(i) = \frac{1-e^{-i}}{c}.$$

	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	1	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Table 3.7. Fitness proportionate (FP) versus linear ranking (LR) selection

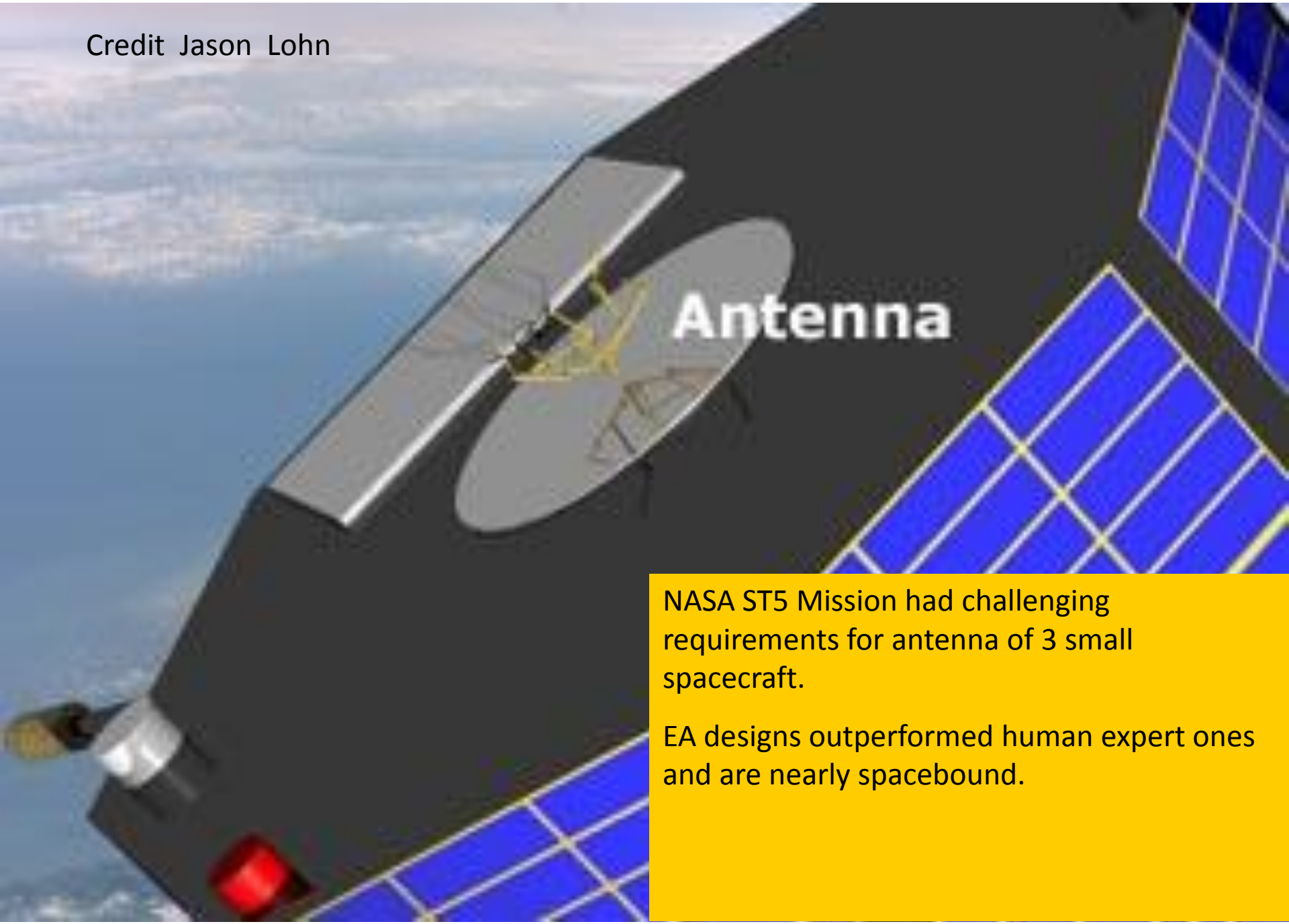
Parent Selection

- Tournament Selection

```
BEGIN
  set current_member = 1;
  WHILE ( current_member ≤ μ ) DO
    Pick k individuals randomly, with or without replacement;
    Select the best of these k comparing their fitness values;
    Denote this individual as i;
    set mating_pool[current_member] = i;
    set current_member = current_member + 1;
  OD
END
```

Survivor Selection

- **Age-Based Replacement**
  - Aged individuals will be changed
- **Fitness-Based Replacement**
  - fitness proportionate and tournament selection
  - Replace Worst (GENITOR)
  - Elitism

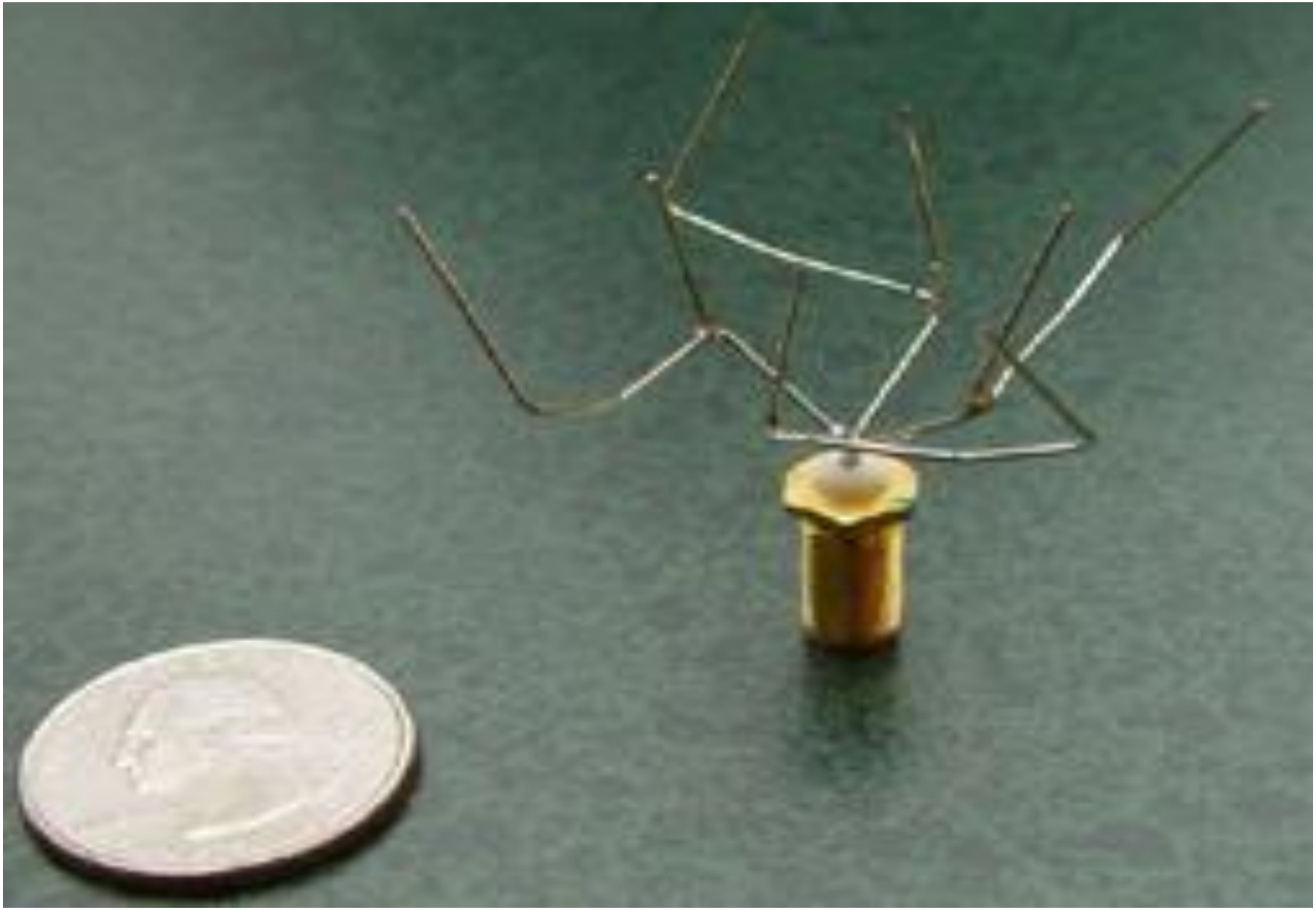


Credit Jason Lohn

NASA ST5 Mission had challenging requirements for antenna of 3 small spacecraft.

EA designs outperformed human expert ones and are nearly spacebound.

Credit Jason Lohn



Reference

- **Prof. Dr. A. E. Eiben, Dr. J. E. Smith auth. Introduction to Evolutionary Computing, Corrected second printing,Springer-ACM,2007**

Thank you