Machine Learning

Lecture 10

Evolution Programming and Genetic Algorithms

Any associations …

Evolution theory

Evolution is based upon the distinction

Phenotype: The observable characteristics of organism (behavior, physical attributes, mental attributes)

Genotype: Controls development of characteristics, given an environment; governs inheritance of ability to express characteristics.

Gene: A collection of genetic units (e.g., nucleotides) which govern the development of some characteristic.

Chromosome: A string of genetic units.

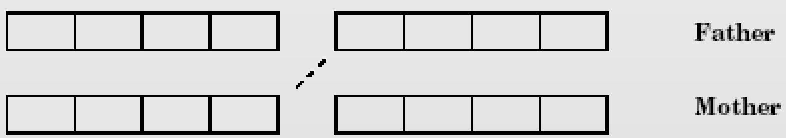
Evolution based on:

- **Reproduction** with **cross-over**. Inheritance and mixture of features provide the keeping of useful features (stability) and variability (plasticity). Offspring always distinct from parents
- **Selection** of most perspective individuals in **population** for producing of **next generation.** Selection (in AL) is controlled by value of **fitness-function.** It simulates natural selection (controlled by deaths).
- **Mutation** provides more strong variations than cross-over

Sexual reproduction

Sexual reproduction

- Brings together chromosome strings (homologous pairs)
- Crosses them over (mixes them)
- Separates the results



⇒ This is a source of genetic variation: Offspring are always distinct from parents.

Natural Selection

Individuals that fail to reproduce withdraw their ge
from the gene pool

Differential reproduction is the engine of evolution

What remains/flourishes in gene pool? Whatever enhances reproductive success:

- strength
- wit
- sexually attractive features (e.g., big feathers, bright colors, varied song)

"Nature red in tooth and claw"

Basically, a Social Darwinism/Nazi distortion of evol theory

How does it work?

Selection can only *shrink* the gene pool

Something fails to reproduce

That can't bring about **new** adaptations

- Eyeballs, wings, etc.

Selection must operate upon a gene pool constantly renewed by new sources of diversity:

Mutation via copy errors, radiation, chemical mutagens, cosmic rays, etc.

Sexual reproduction: crossover and recombination

Recessive and polygenic characters

Deletion, duplication, inversion

Self-Reproduction in Computers

- An old mathematical problem, to write a program that can reproduce (e.g., print out a copy of itself) leads to infinite regress.

Solution to Infinite Regress

- Solution in the von Neumann computer architecture. He also described a “self-reproducing automaton”
- Basic idea
  – Computer program stored in computer memory
  – A program has access to the memory where it is stored
  – Let’s say we have an instruction MEM that is the location in memory of the instruction currently being executed

A Working Self-Reproducing Program

```
Program copy
      L = MEM +1
      print("Program copy")
      print("L = MEM + 1")
      LOOP until line[L] = "end"
            print(line[L])
            L=L+1
      print("end")
end
```

Self-Reproducing Program

- Information used two ways
  - As instructions to execute
  - As data for the instructions
- Could make an analogy with DNA
  - DNA strings of nucleotides
  - DNA encodes the enzymes that effect copying: splitting the double helix, copying each strand with RNA, etc.

Evolution in Computers

- Genetic Algorithms – most widely known work by John Holland
- Based on Darwinian Evolution
  - In a competitive environment, strongest, “most fit” of a species survive, weak die
  - Survivors pass their good genes on to offspring
  - Occasional mutation

Evolution in Computers

- Same idea in computers
  - Population of computer program / solution treated like the critters above, typically encoded as a bit string
  - Survival Instinct – have computer programs compete with one another in some environment, evolve with mutation and sexual recombination

GA's for Computer Problems

Population of critters → Population of computer solutions
Surviving in environment → Solving computer problem
Fitness measure in nature → Fitness measure solving computer problem
Fit individuals life, poor die → Play God and kill computer solutions that do poorly, keep those that do well. i.e. "breed" the best solutions typically Fitness Proportionate Reduction
Pass genes along via mating → Pass genes along through computer mating
Repeat process, getting more and more fit individuals in each generation.
Usually represent computer solutions as bit strings.

“Digital life of programs”

- Cellular Automata: Artificial chemistry or physics substrate to support living systems.
- Alternative: Use the “chemistry” of the von Neumann computer as a substrate for lifelike behavior:
  – Core Wars
  – Tierra
  – Avida
- Analogy:
  – Organic life uses energy (from the sun) to organize matter.
  – Digital life uses CPU time to organize memory.
- Organic life evolves through natural selection as individuals compete for resources (light, food, space).
- Digital life evolves through the same process, as replicating algorithms compete for CPU time and memory space.

The Digital Environment

- Abiotic Environment
  - Memory
  - CPU
  - Operating system

- Creatures
  - Self-replicating assembly-language programs
  - Analogy to RNA world (same structure carries genetic information and performs metabolic activity)
  - Machine instructions <-> Amino acids (chemically active)

Tierra (1)

- Virtual computer:
  - MIMD (time-sharing model)
  - One processor for each creature.
- Each processor (CPU):
  - 2 address registers
  - 2 numeric registers
  - flags (for errors)
  - stack pointer
  - 10-word stack
  - instruction pointer
- Each CPU performs a perpetual cycle:
  - Fetch-Decode-Execute-Increment IP
  - Fetch the instruction addressed by IP
  - Decode it
  - Execute the instruction
  - Increment the IP to next sequential location in RAM (except in case of JMP, CALL, or RET)

Tierra (2)

- 1 RAM for all CPUs (the soup):
  - Memory protection within allocated block of memory (write protection, not read protection).
- Instruction set:
  - 32 instructions including operands
  - Pattern-based addressing
  - E,g,
    - JMP NOP0 NOP0 NOP0 NOP1
    - System will look outwards in both directions from JMP instruction to the nearest occurrence of a complementary pattern (NOP1 NOP1 NOP1 NOP0)
    - If Found, IP jumps to end of pattern and resumes execution.
  - Everything else standard (MOV,CALL, RET, POP, PUSH)
  - Copy and fork built in.
  - DIVIDE instruction: Creates a new IP (cell division)
- Reaper (death)
  - Removes creatures that have errors
  - Removes creatures that have lived the longest
- Slicer:
  - Allocates time slices

Tierra (3)

- Mutation:
  - Instructions sometimes off by +/- 1.
- Ancestor:
  - Hand-craft a single minimal self replicating program.
  - 80 instructions long.
  - Locate beginning and ending address.
  - Subtract to determine size.
  - Allocate block of memory for daughter.
  - Copy genome to new memory 1 cell at a time.
  - Execute DIVIDE to create a new IP (cell division).

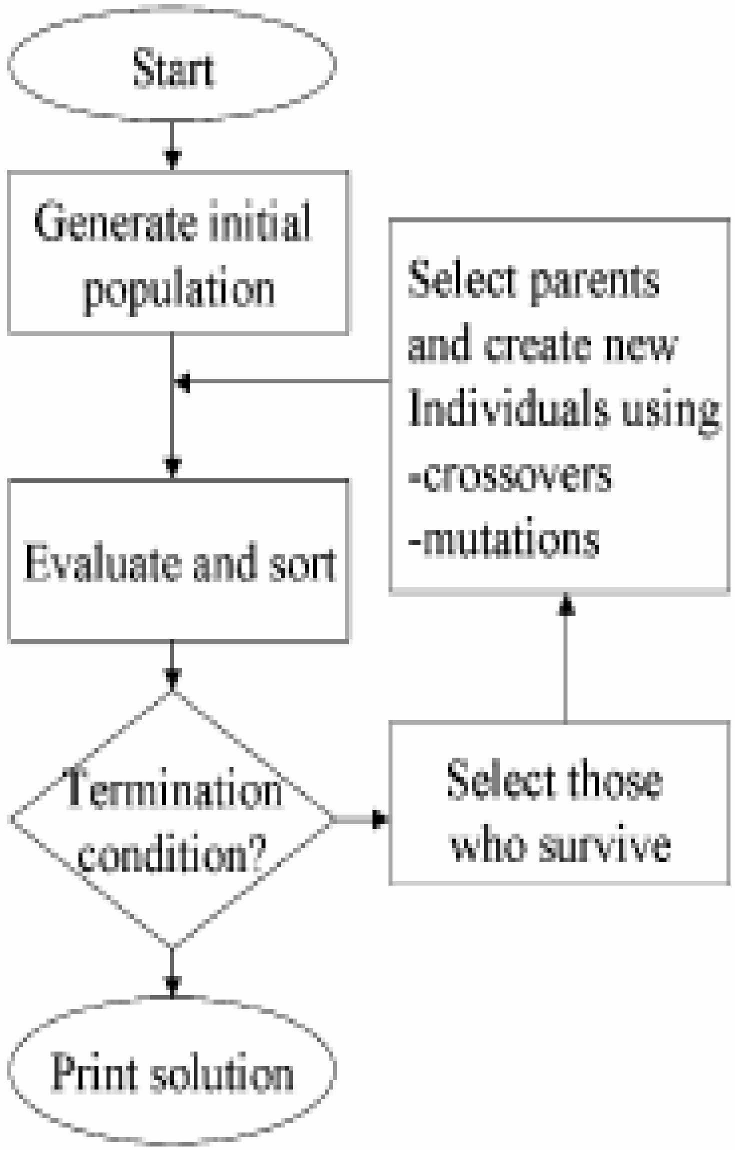
Tierra. Results

- Ancestor replicated to fill RAM.
- Diversified through mutation.
- Length shrank from 80 instructions to 45
  – Parasites hijacked other programs' copying routines.
- Parasites bommed, then crashed.
- Hosts evolved immunity to the parasites.
  – Analogy: E. coli develop immunity to bacteriophage.
- Arms Race:
  – Hosts
  – Parasites
  – Immune hosts
  – Parasites overcome immune hosts
  – Symbionts
  – Cheaters
  – Super-parasites

The Simple Genetic Algorithm

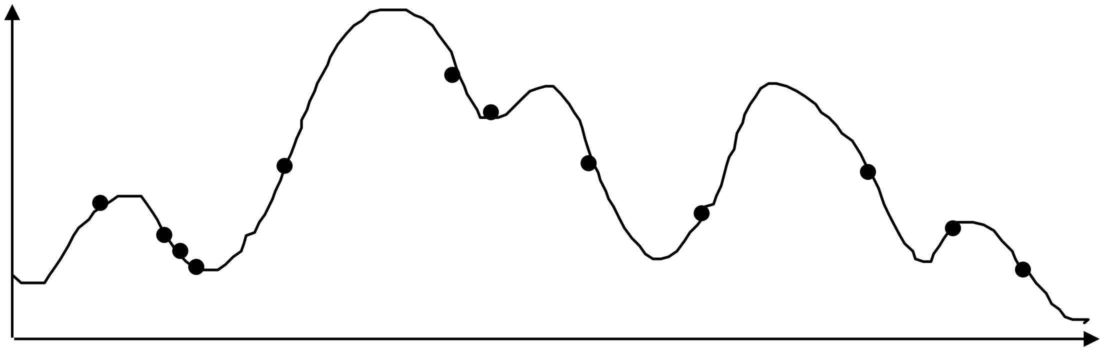
1. Generate an initial random population of M individuals (i.e. programs)
2. Repeat for N generations
    1. Calculate a numeric fitness for each individual
    2. Repeat until there are M individuals in the new population
        1. Choose two parents from the current population probabilistically based on fitness (i.e. those with a higher fitness are more likely to be selected)
        2. Cross them over at random points, i.e. generate children based on parents (note external copy routine)
        3. Mutate with some small probability
        4. Put offspring into the new population

Genetic algorithm

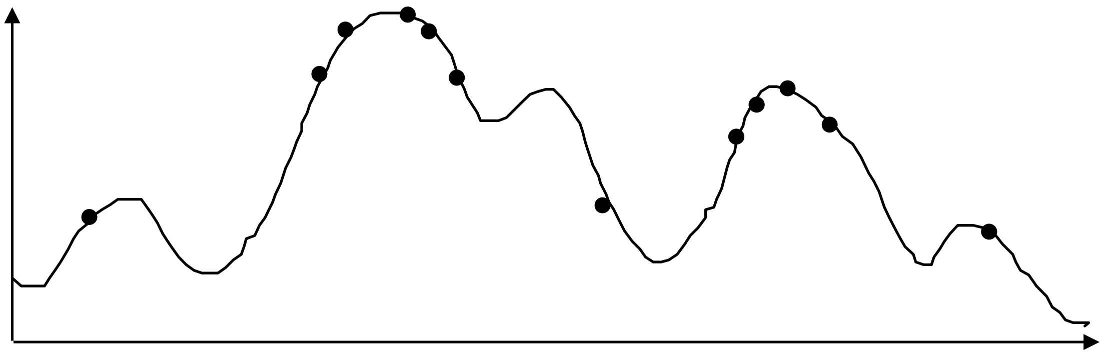


An Abstract Example

Fitness function



Distribution of Individuals in Generation 0



Distribution of Individuals in Generation N

GA parameters

Parameters to a GA simulation run include:

- Population size
- Selection method. Possible choices:
  - Fitness proportional probability
  - Retain top k
  - Tournament selection
  - etc.
- Mutation rate; meta-mutation?
- Crossover
  - One-point
  - Two-point
  - Uniform

Crossover

Typically use bit strings, but could use other structures

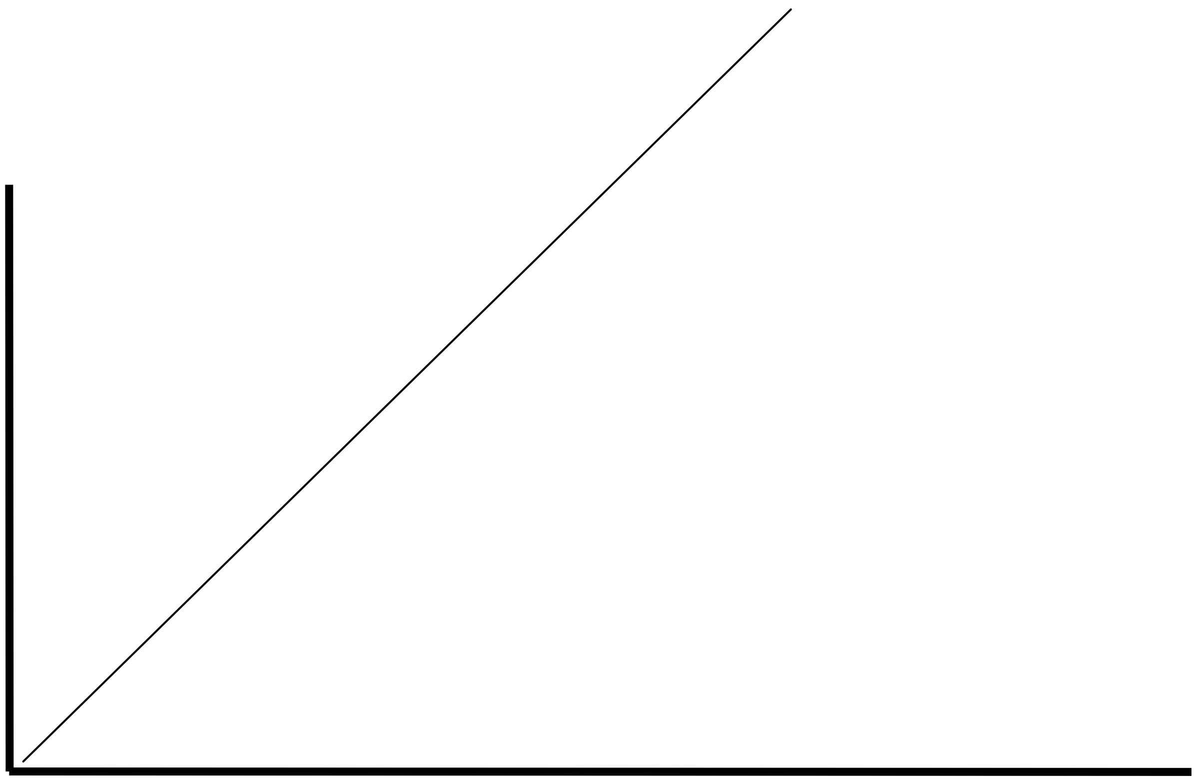
Bit Strings: Genotype representing some phenotype

Individual 1: 00101**0001** Individual 2: **10011**0110

New child : **100110001** has characteristics of both parents, hopefully better than before

Bit string can represent whatever we want for our particular problem; solution to a complex equation, logic problem, classification of some data, aesthetic art, music, etc.

Simple example: Find MAX of a function



To keep it simple, use y=x so bigger X is better

Chromosome Representation

Let's make our individuals just be numbers along the X axis, represented as bit strings, and initialize them randomly:

Individual 1	:	000000000
Individual 2	:	001010001
Individual 3	:	100111111
….		
Individual N	:	110101101

Fitness function: Y value of each solution. This is the fitness function. Note that even for NP complete problems, we can often compute a fitness (remember that solutions for NP Complete problems can be verified in Polynomial time).

Say for some parents we pick: 100111111 and 110101101

Crossover

Crossover: Randomly select crossover point, and swap code

100111111 and 110101101

Individual 1:	10011**1111**	Individual 2:	**11010**1101
New child :	110101111	has characteristics of both parents, hopefully better than before	

Or could have done:

Individual 1:	**10011**1111	Individual 2:	11010**1101**
New child:	100111101	; not better in this case	

Mutation

Mutation: Just randomly flip some bits ; low probability of doing this

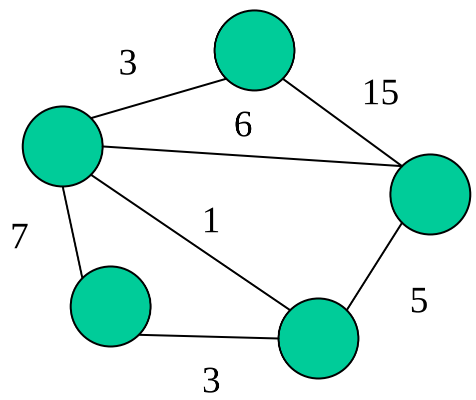
Individual:	**0**11100101
New:	**1**11100101

Mutation keeps the gene pool active and helps prevent stagnation.

Second Example : TSP

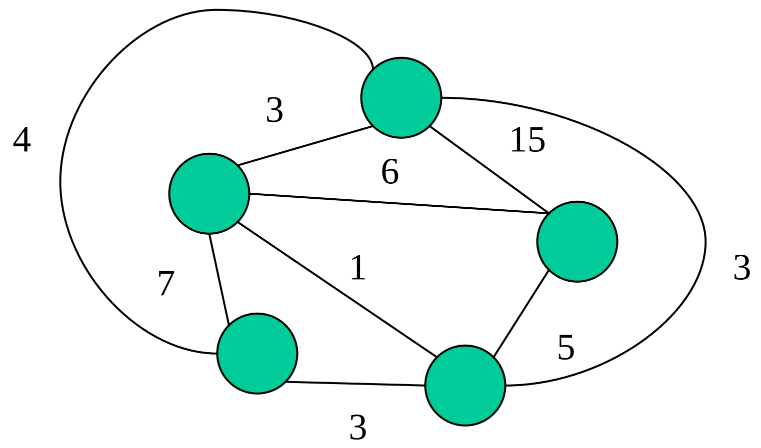
- NP-Complete
- NP-Complete problems are good candidates for applying GA's
  - Problem space too large to solve exhaustively
  - Multiple "agents" (each individual in the population) provides a good way to probe the landscape of the problem space
  - Generally not guaranteed to solve the problem optimally

- Formal definition for the TSP
  - Start with a graph G, composed of edges E and vertices V, e.g. the following has 5 nodes, 7 edges, and costs associated with each edge:



  - Find a loop (tour) that visits each node exactly once and whose total cost (sum of the edges) is the minimum possible

- Easy on the graph shown on the previous slide; becomes harder as the number of nodes and edges increases



- Adding two new edges results in five new paths to examine
- For a fully connected graph with n nodes, n! loops possible
  - Impractical to search them all for more than about 25 nodes
- Excluding degenerate graphs, an exponential number of loops possible in terms of the number of nodes/edges

- Guaranteed optimal solution to TSP
  - Evaluate all loops
- Approximation Algorithms
  - May achieve optimal solution but not guaranteed
  - Nearest Neighbor
  - Find minimum cost of edges to connect each node then turn into a loop
  - Heuristic approaches, simulated annealing
  - Genetic Algorithm

- A genetic algorithm approach
  - Randomly generate a population of agents
    - Each agent represents an entire solution, i.e. a random ordering of each node representing a loop
      - Given nodes 1-6, we might generate 423651 to represent the loop of visiting 4 first, then 2, then 3, then 6, then 5, then 1, then back to 4
      - In a fully connected graph we can select any ordering, but in a partially connected graph we must ensure only valid loops are generated
  - Assign each agent a fitness value
    - Fitness is just the sum of the edges in the loop; lower is more fit
  - Evolve a new, hopefully better, generation of the same number of agents
    - Select two parents randomly, but higher probability of selection if better fitness
    - New generation formed by crossover and mutation

- Crossover
  - Must combine parents in a way that preserves valid loops
  - Typical cross method, but invalid for this problem

    Parent 1 = 423651   Parent 2 = 156234

    Child 1 = 423234   Child 2 = 156651
  - Use a form of order-preserving crossover:

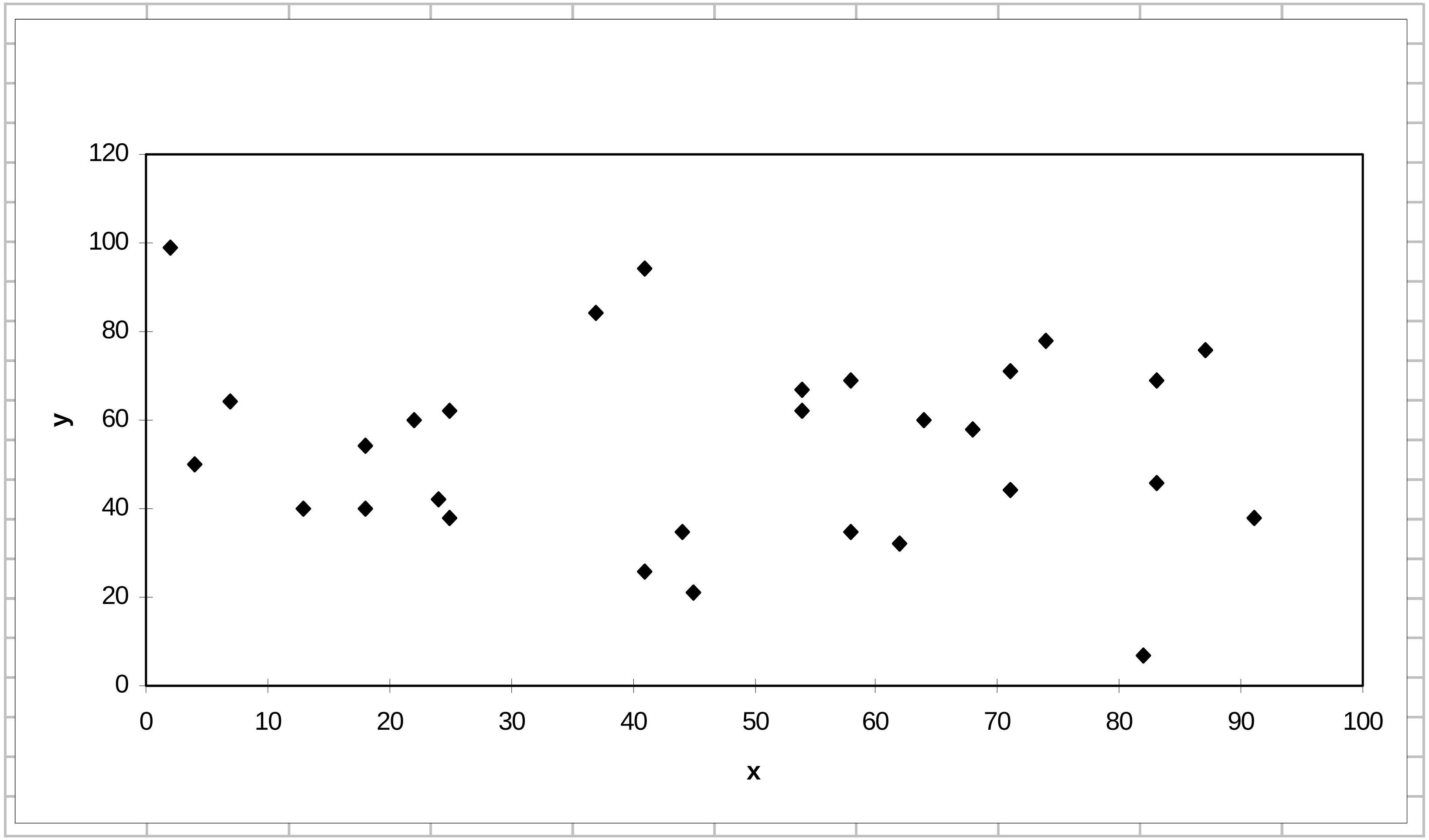
    Parent 1 = 423651   Parent 2 = 156234

    Child 1 = 123654
    - Copy positions over directly from one parent, fill in from left to right from other parent if not already in the child
- Mutation
  - Randomly swap nodes (may or may not be neighbors)

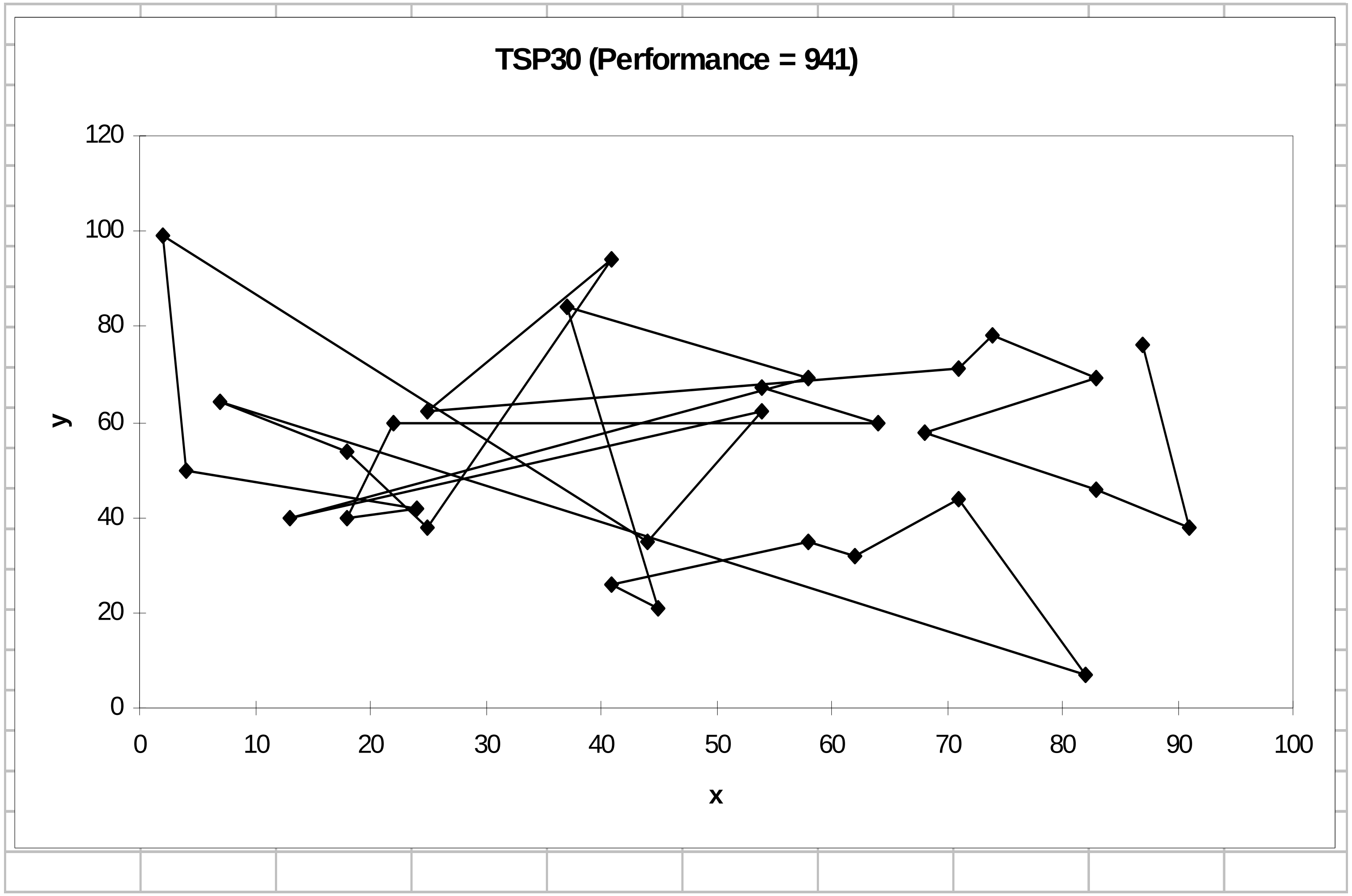
Why does this work?

- How does a GA differ from random search?
  - Pick best individuals and save their “good” properties, not random ones
- What information is contained in the strings and their fitness values, that allows us to direct the search towards improved solutions?
  - Similarities among the strings with high fitness value suggest a relationship between those similarities and good solutions.
    - A schema is a similarity template describing a subset of strings with similarities at certain string positions.
    - Crossover leaves a schema unaffected if it doesn't cut the schema.
    - Mutation leaves a schema unaffected with high probability (since mutation has a low probability).
    - Highly-fit, short schema (called building blocks) are propagated from generation to generation with high probability.
  - Competing schemata are replicated exponentially according to their fitness value.
  - Good schemata rapidly dominate bad ones.

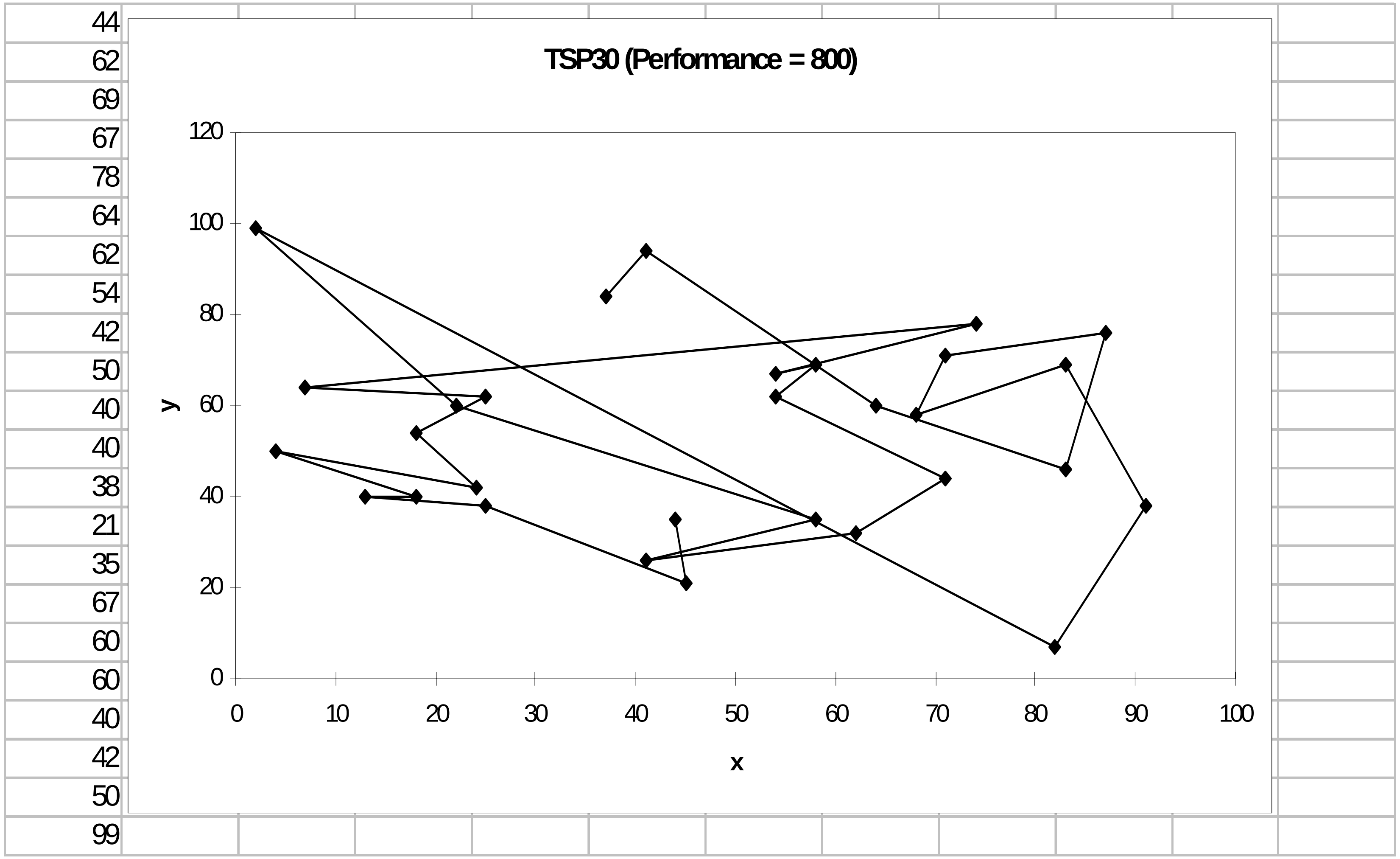
TSP Example: 30 Cities
(J.Abonyi, J.Madar)



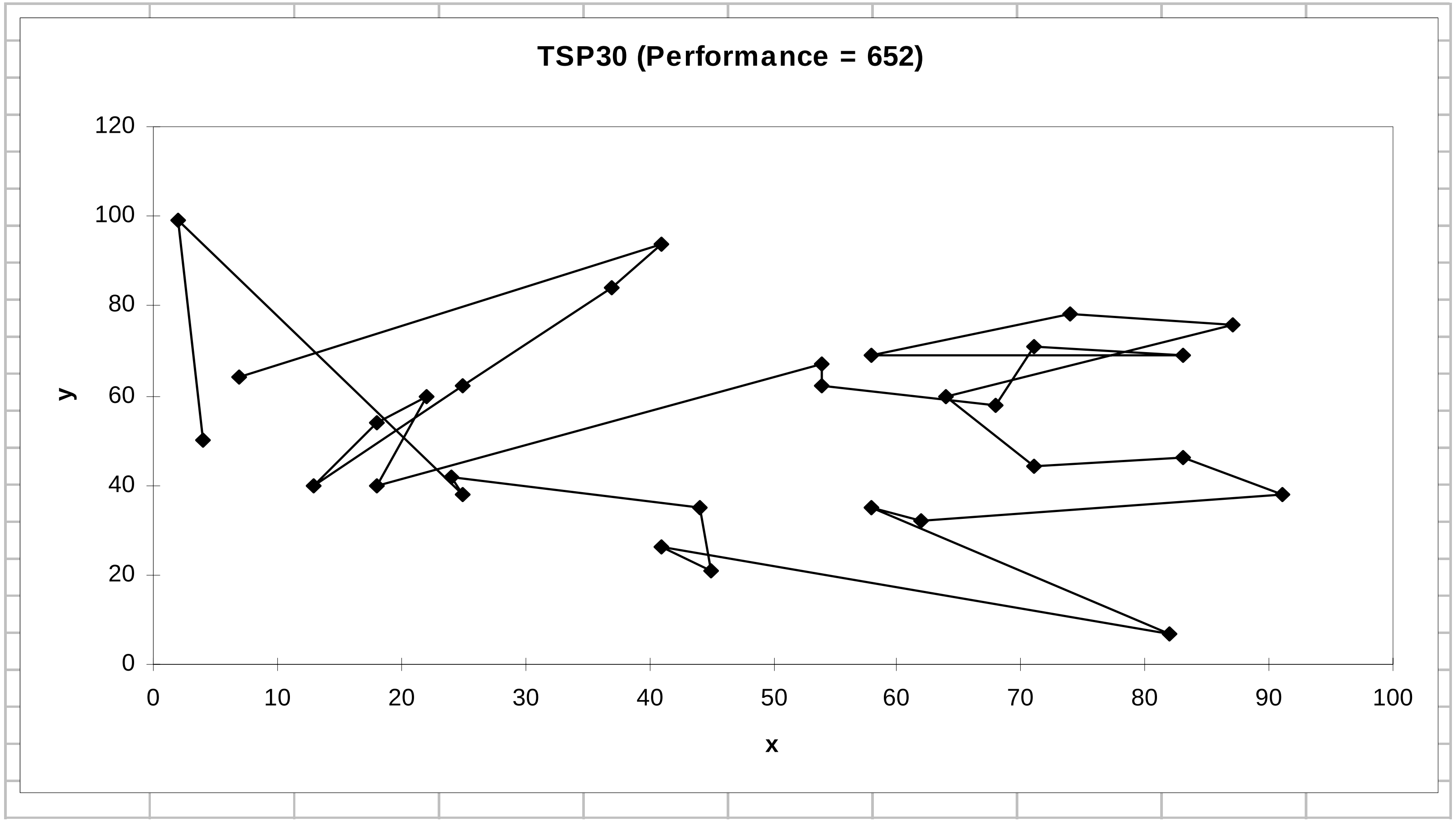
Solution (Distance=941)



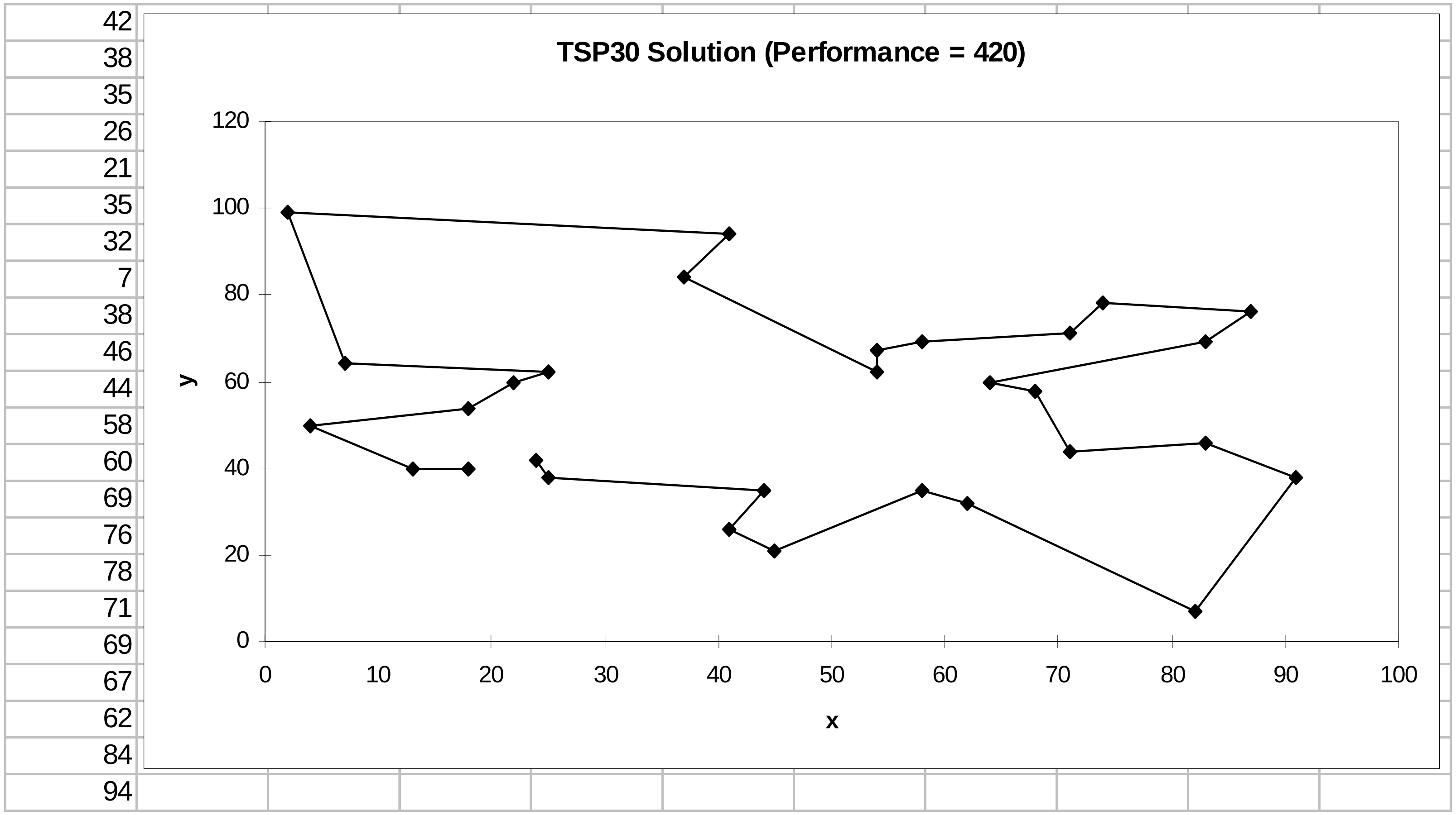
Solution (Distance=800)



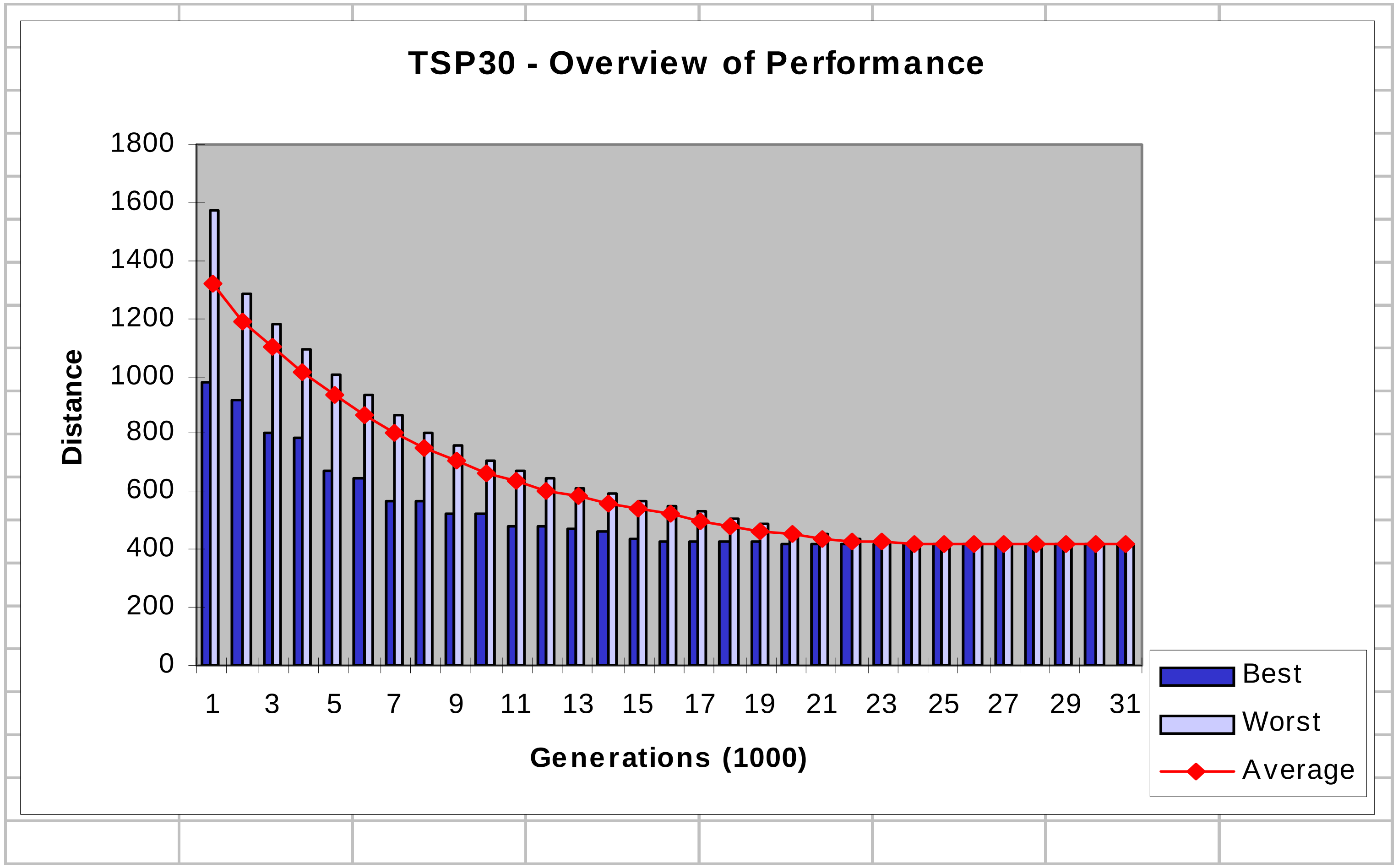
Solution (Distance=652)



Best solution (Distance=420)



Overview of performance



Advantages of GA’s

Easy to understand and implement
Easy to adapt to many problems
Work surprisingly well
Modular, separate from application
Supports multi-objective optimization
Good for “noisy” environments
Inherently parallel; easily distributed
Many variations are possible
(elitism, niche populations, hybrid w/other techniques)
Less likely to get stuck in a local minima due to randomness

Problems of GA’s

- Need diverse genetic pool, or we can get inbreeding :
  stagnant population base

- No guarantee that children will be better than parents
  could be worse, could lose a super individual

  elitism- when we save the best individual
- Very slow methods for optimization
- Sometimes definition of task and programming are difficult.

Effectiveness of usage of GA depend on definition of task (e.g. structure of chromosome and semantics of genes)