Implementation Report: Concurrent Genetic Algorithm with Island Migration

Markus Solbach

Laboratory for Active and Attentive Vision
Department of Computer Science and Engineering
York University, Toronto, Ontario, Canada

November 10, 2015

Overview

- Genetic Algorithm revisited
- Genetic Algorithm Operators
- Concurrency
- Some Results
- Future Work

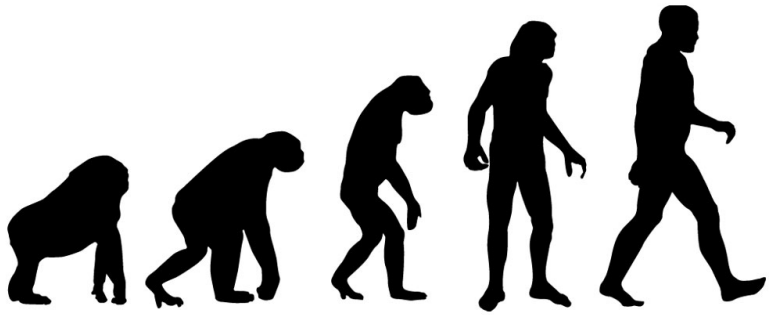
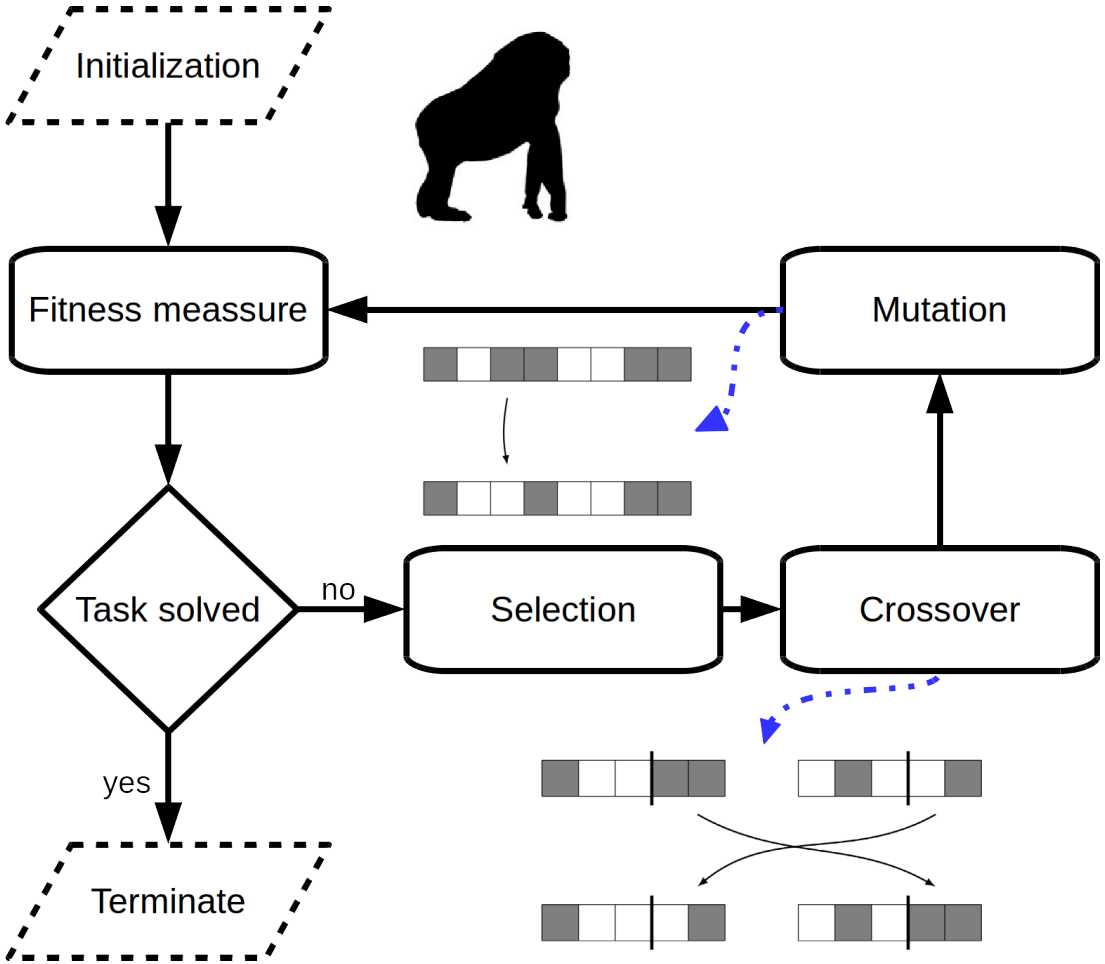


Figure :
Evolution ||*i.livescience.com (Oct. 5. 15)*

Genetic Algorithm revisited

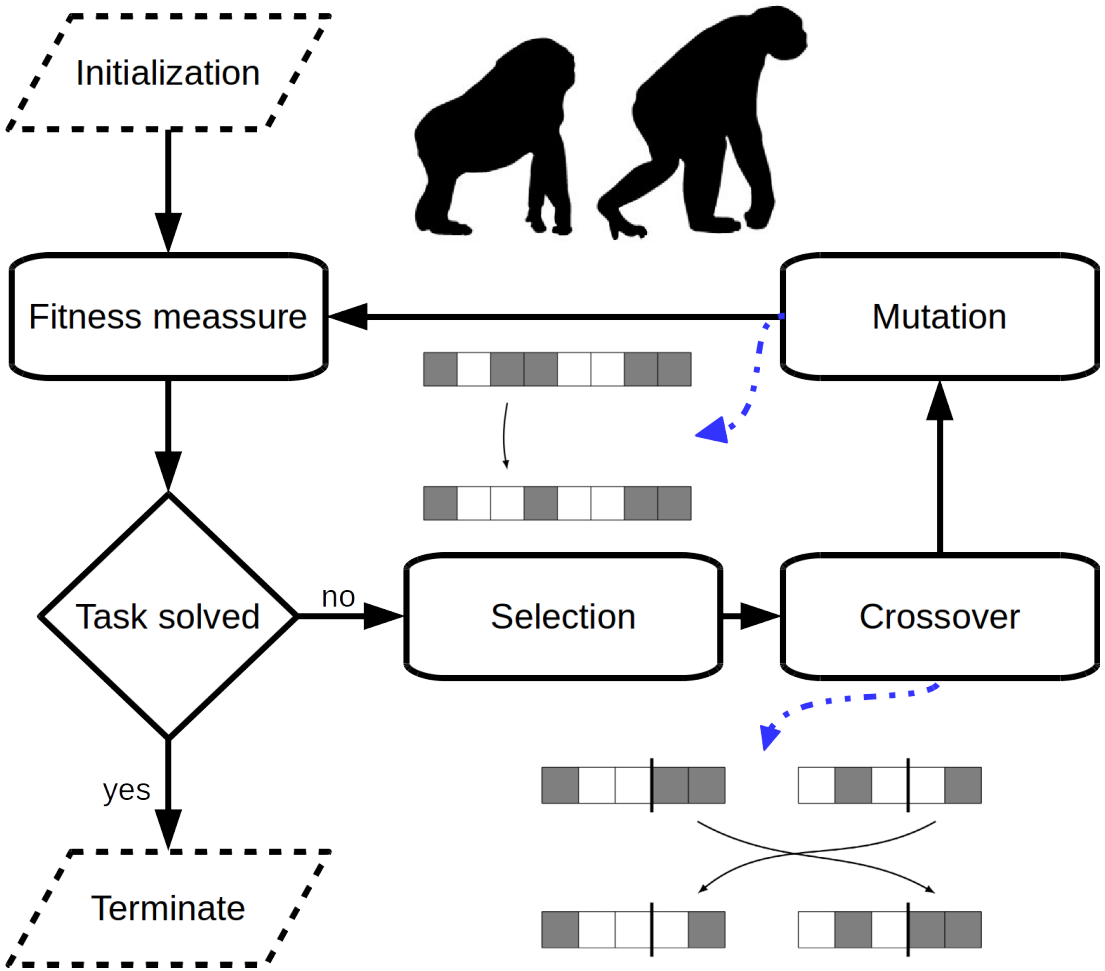
Genetic Algorithm revisited

Generation 0



Genetic Algorithm revisited

Generation n



VLSI Design Problem

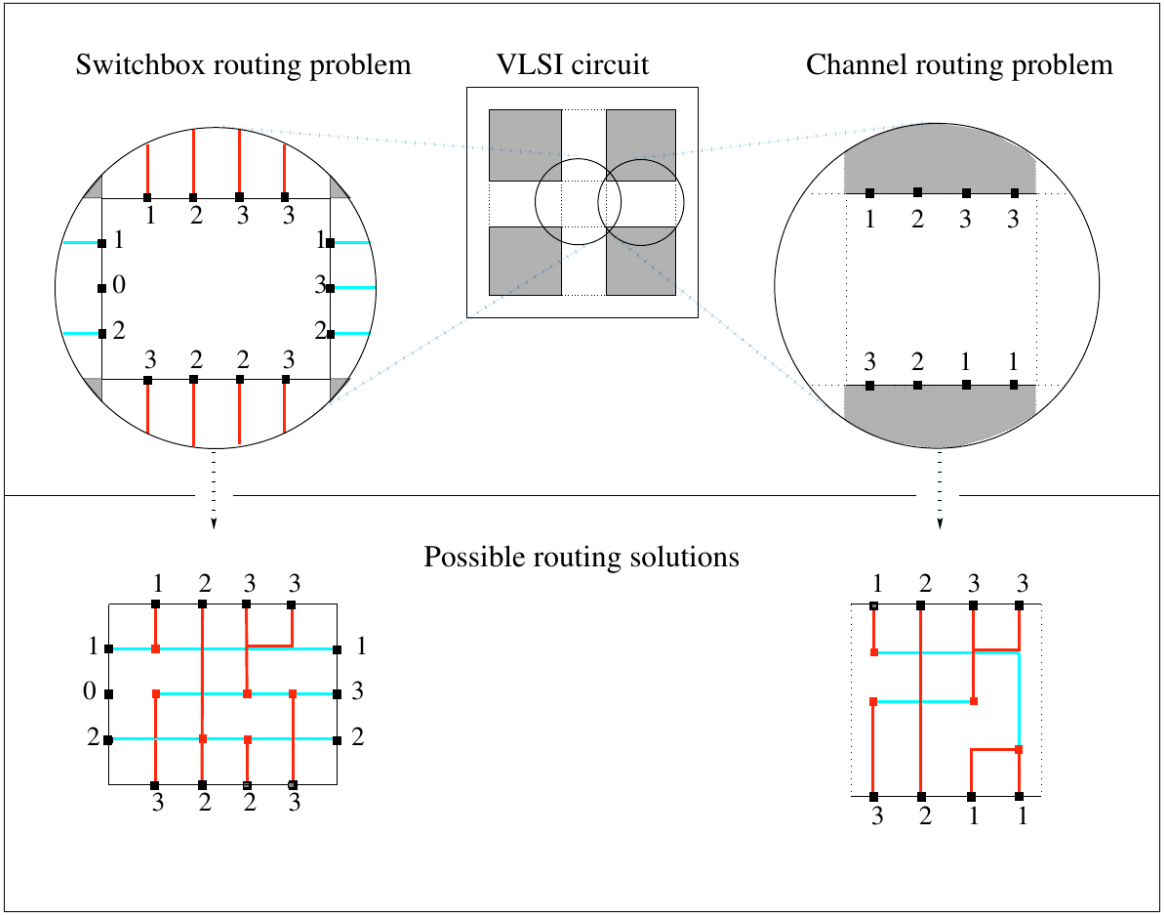


Figure :VLSI Design Problem

VLSI Design Problem - Traveling Salesman Problem

Problem changed for good reasons:

- VLSI very little implementation details
- VLSI faced problems had nothing to do with concurrency
- TSP well known problem
- TSP rich implementation details in literature (sequential)
- TSP able to concentrade more on concurrency

Traveling Salesman Problem

Find a route on a map

Requirements:

- Visit each City only once
- Find shortest path

Complexity (20 city):

- O(n!) (Brute Force, WC)
- $20! = 2.432902x10^{18}$

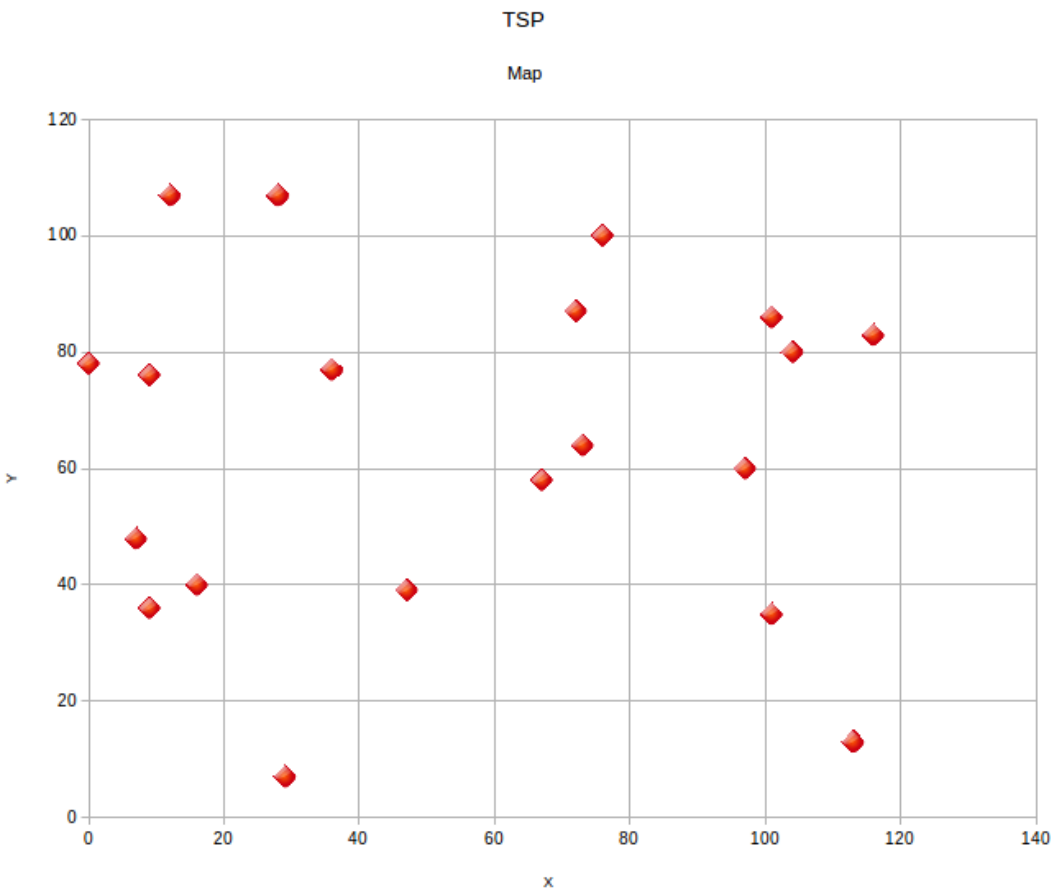
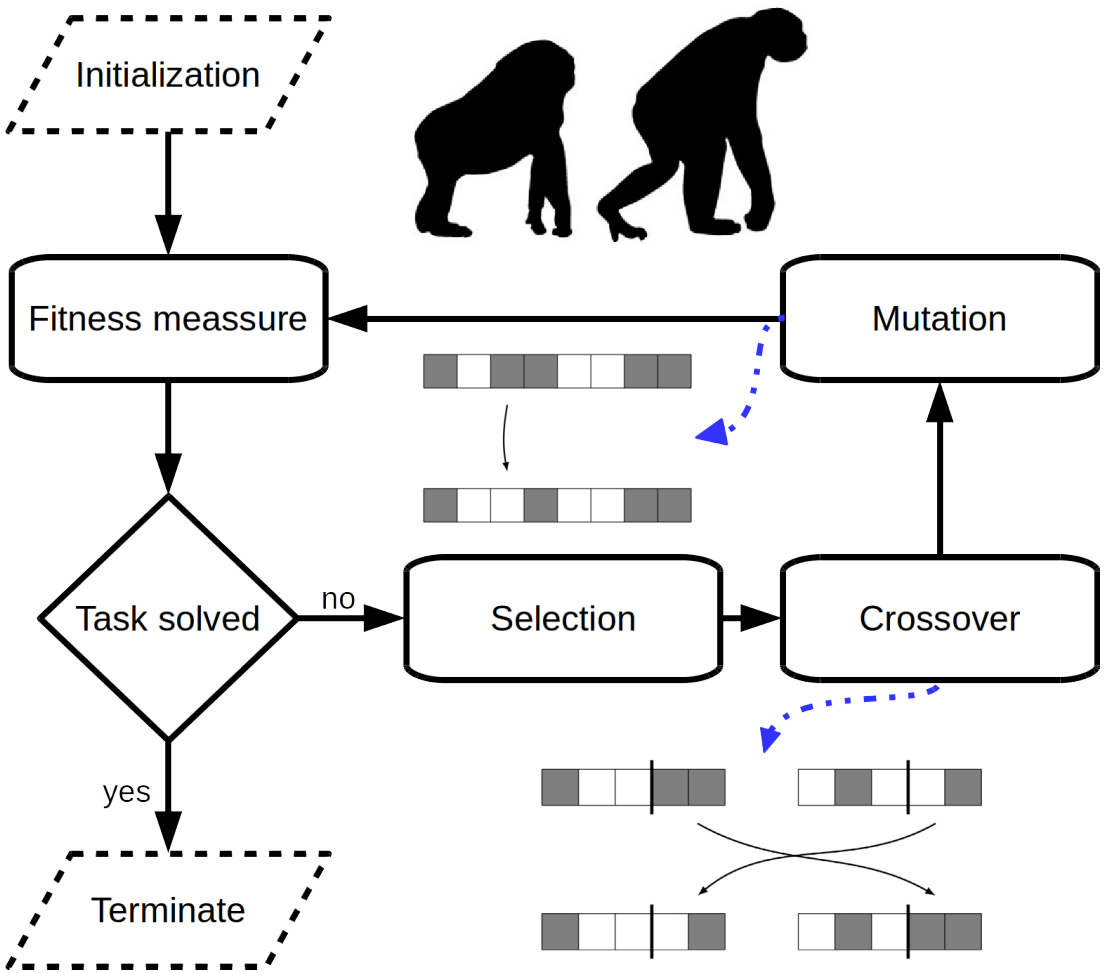


Figure :TSP Map

Genetic Algorithm Operators



Initialization

```
1 // create a random individual
2 public void generateIndividual() {
3     // Go through all available Cities and add it to the tour
4     for (int cityIndex = 0; cityIndex <
          CGaimDestinationPool.numberOfCities(); cityIndex++) {
5       setCity(cityIndex, CGaimDestinationPool.getCity(cityIndex));
6     }
7     // shuffle the tour
8     Collections.shuffle(tour);
9 }
```

TO	BE	AM	EH	SG	KL	MA

Figure :High Level Individual Representation

Mutation

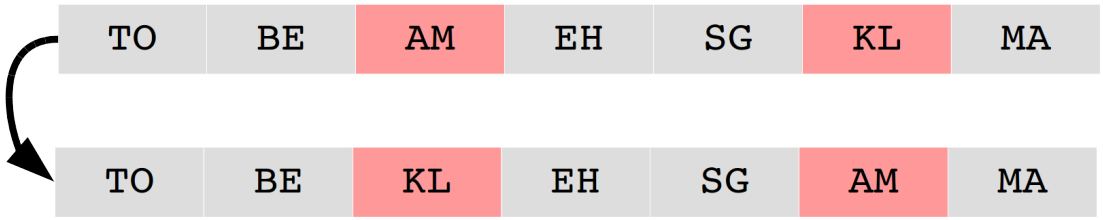


Figure :High Level Mutation Representation

Crossover

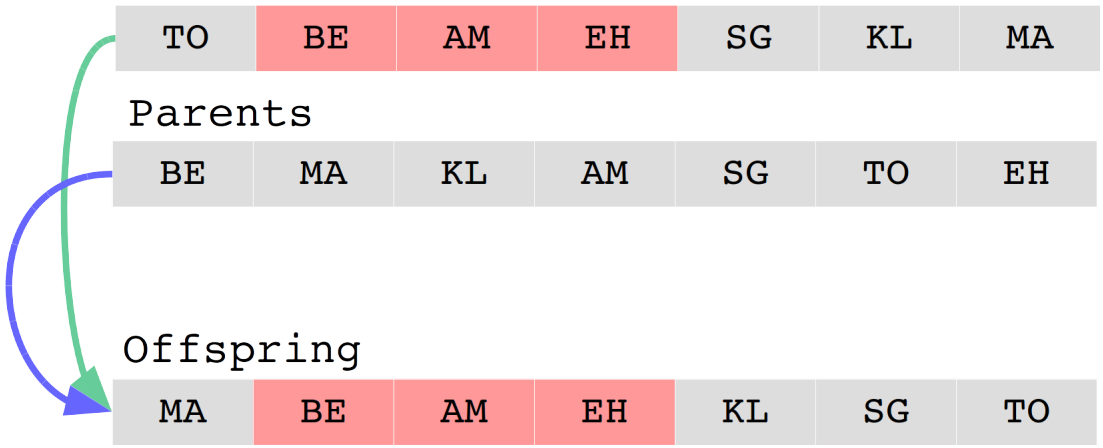


Figure :High Level Crossover Representation

Fitness Function

- Travelled distance over all city (inverse fitness)
- Each City has a location (x, y)
- Euclidean distance
- $Fitness_1 = \sum_{i=1}^{n}(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2$

Concurrency

Concurrent Random Number Generator

Genetic Algorithms rely heavily on random numbers

- *Math.random()* is not concurrent
- Multiple threads use similar or same seeds
- *ThreadLocalRandom* [1]
- Generator with an internally generated seed
- *java.util.concurrent.ThreadLocalRandom*

[1] `http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadLocalRandom.html`

Threads and Islands

Each Thread represents one Island

- Genetic Algorithm (GA) Logic
- Implements *Runnable*
- Concurrent Execution

= GA's Island Migration extension

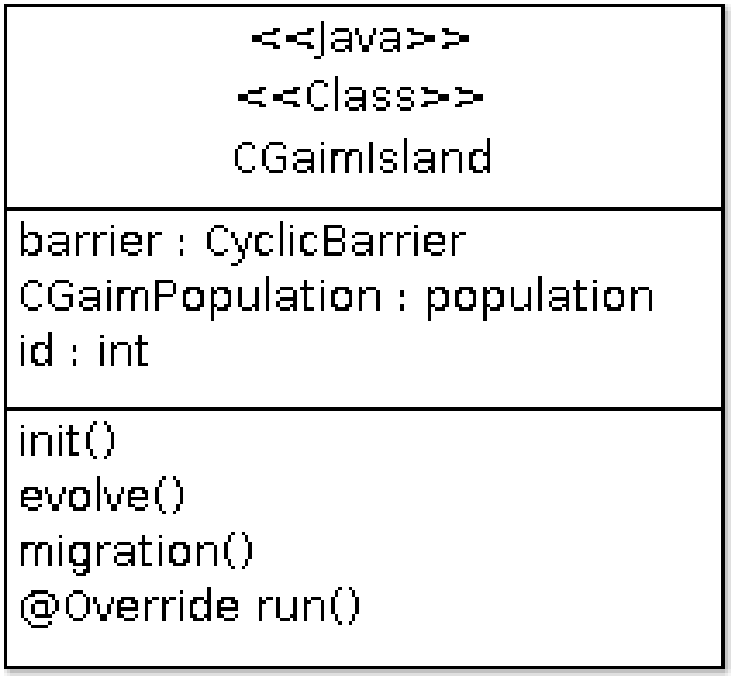


Figure :Simplified UML-Class of CGaimIsland

Threads and Islands

Each Thread represents one Island

- Genetic Algorithm (GA) Logic
- Implements *Runnable*
- Concurrent Execution

= GA's Island Migration extension

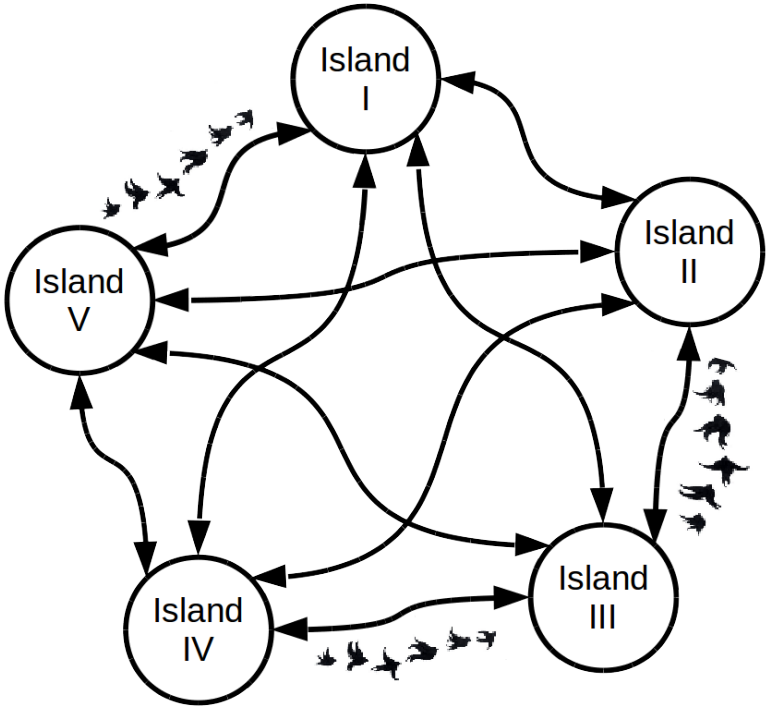


Figure :Island Migration overview

Threads and Barriers

Barriers[2] for synchronization

- *new CyclicBarrier(# Islands)*
- Waits that all Islands are ready
- Evolution $\hookrightarrow$ random process

```
1 @Override
2 public void run() {
3   barrier.await();
4   /* start evolution */
5   this.evolve();
6 }
```

[2] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CyclicBarrier.html

Island Migration

Sequentially (as proposed)

- Dependens on epoch length
- Joins all Threads
- Avoids shared memory access
- Performs cyclic Migration
- Copies Individuals
- Java's *array.clone()[i]*

```
1 /* Wait for Threads */
2 for(int i = 0; i < numberIslands; i++)
3 {
4   thread[i].join();
5 }
```

Island Migration

Sequentially (as proposed)

- Dependens on epoch length
- Joins all Threads
- Avoids shared memory access
- Performs cyclic Migration
- Copies Individuals
- Java's *array.clone()[i]*

```
1  /* Perform cyclic Migration */
2  island[1].setMigrants()
3     = island[0].getMigrants();
4  ...
```

Some Results

Some Results (Island Migration)

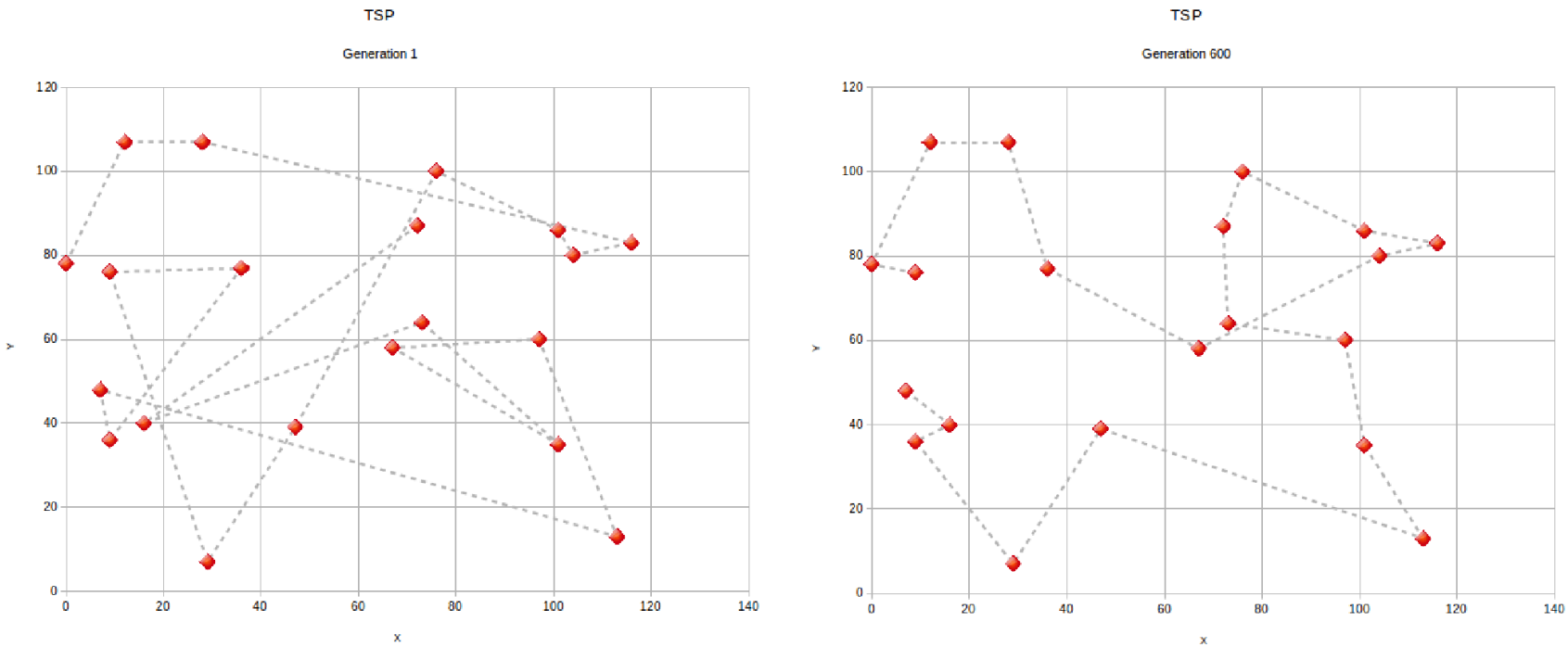


Figure :4 Islands - 150 Individuals - 2 Migrants - 70 Generations Epoch ($\approx$ 5 Sec.)

Some Results (Island Migration vs. Sequential)

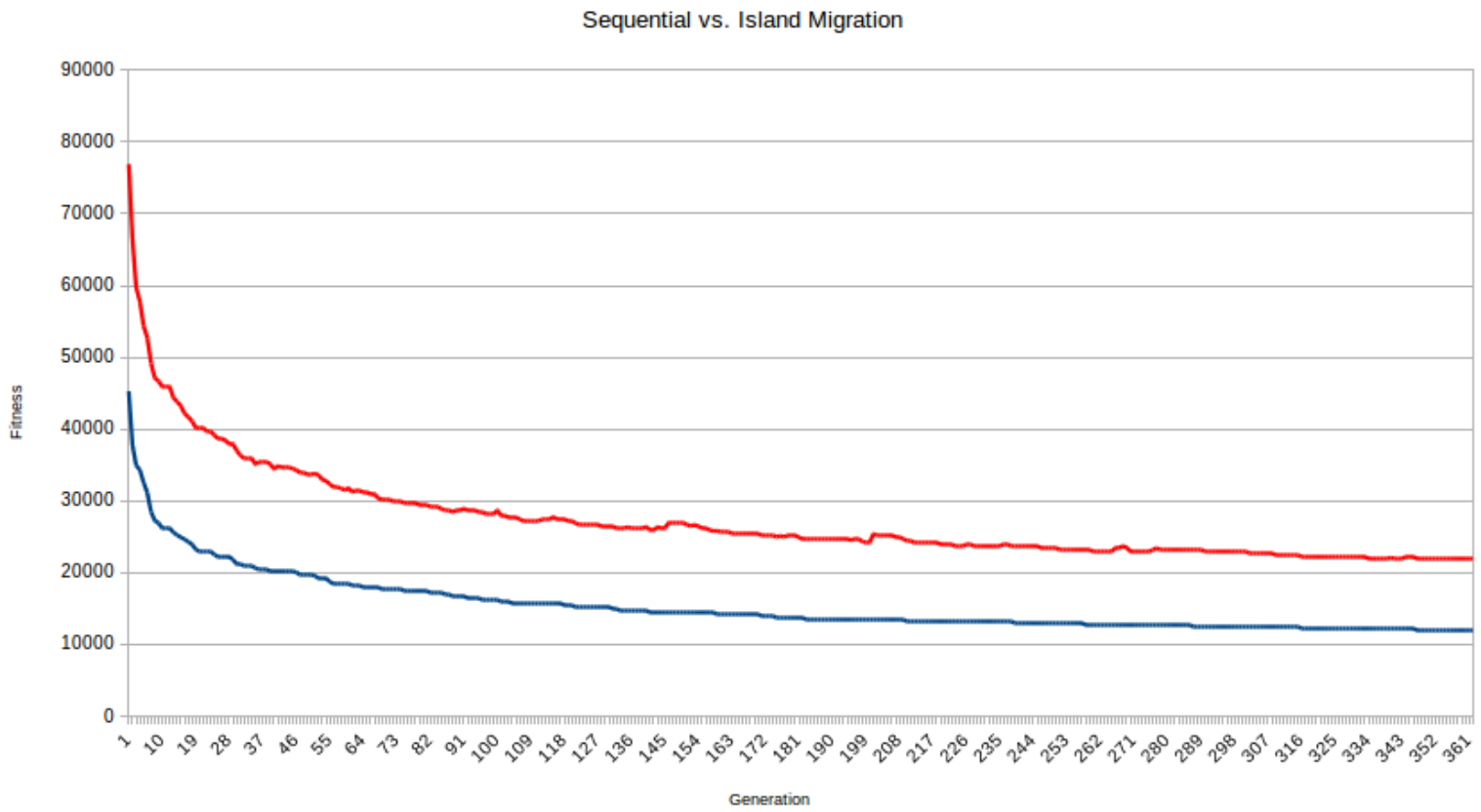


Figure :100 City - Sequential (150 Individuals) vs. 4 Islands (as before)

Future Work

Future Work

- More tests / Parameter improvements
  - intel i7 (8 cores x 4GHz) using 8 Islands
  - 5 times faster than sequential GA on same machine
- More debugging
  - bugs in Migration
  - lack of *.clone()*
  - ...
- Execution time differences based on number of Islands