Alexandar Mihaylov

Luisa Rojas

PARALLEL **GENETIC ALGORITHMS**

OVERVIEW

WHAT IS IT?

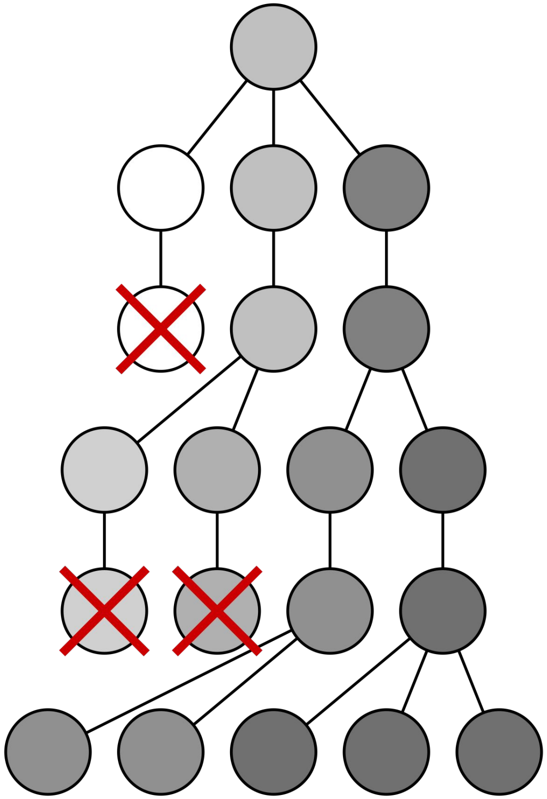
An adaptive heuristic **search algorithm** based on natural selection.

WHAT'S THE GOAL?

Find the target.

HOW?

It modifies a given population. At each step, it selects individuals at random from said population and uses them to produce a new child.



TERMINOLOGY

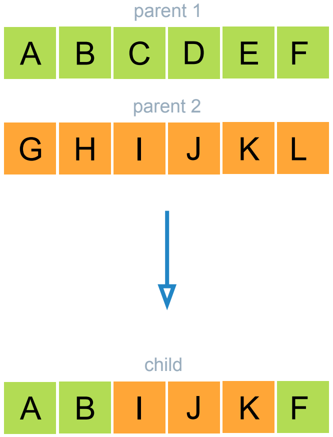
SELECTION

Select the individuals [*parents*], that will be used to create a new candidate for the next generation.

Randomized, but prioritizing the best candidates in the population [*genepool*].

CROSSOVER

Combine two parents to form children for the next generation.



parent 1

A B C D E F

parent 2

G H I J K L

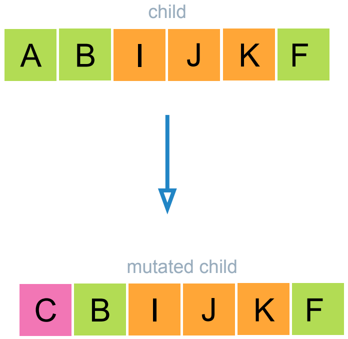
child

A B I J K F

MUTATION

Apply random changes to the resulting child.

This is to prevent the loss of potentially relevant data.



child

A B I J K F

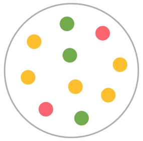
mutated child

C B I J K F

Sequential

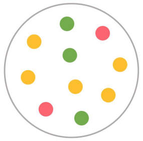
1.

GENERATE GENEPOOL



1.

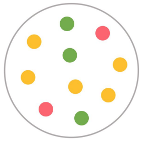
GENERATE GENEPOOL



2.

MODIFY POPULATION

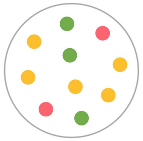
1.
GENERATE GENEPOOL



2.
MODIFY POPULATION

- → Sort genepool.

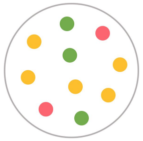
1.
GENERATE GENEPOOL



2.
MODIFY POPULATION

- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.

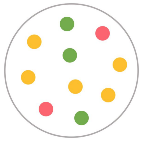
1.
GENERATE GENEPOOL



2.
MODIFY POPULATION

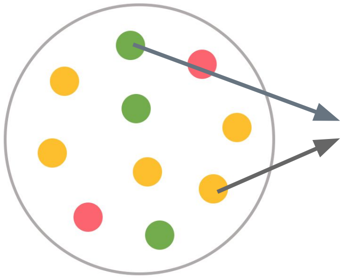
- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.
- ➔ Select two (2) random parents to create new child.

1.
GENERATE GENEPOOL

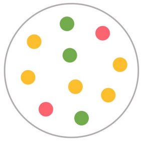


2.
MODIFY POPULATION

- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.
- ➔ Select two (2) random parents to create new child.

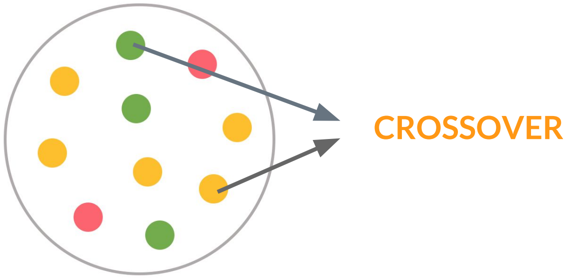


1.
GENERATE GENEPOOL

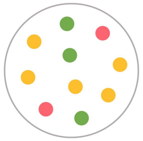


2.
MODIFY POPULATION

- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.
- ➔ Select two (2) random parents to create new child.

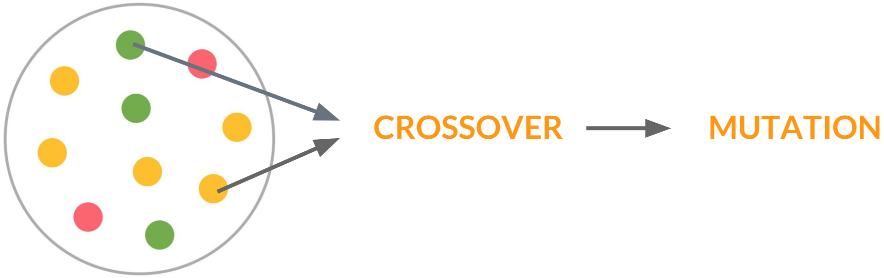


1.
GENERATE GENEPOOL



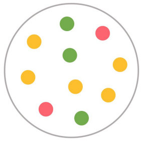
2.
MODIFY POPULATION

- → Sort genepool.
- → Check best candidate in genepool. Return if target is found.
- → Select two (2) random parents to create new child.



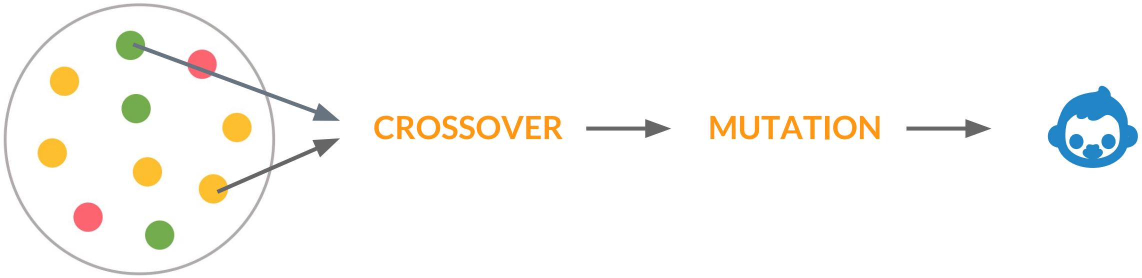
CROSSOVER → MUTATION

1.
GENERATE GENEPOOL



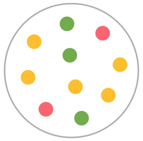
2.
MODIFY POPULATION

- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.
- ➔ Select two (2) random parents to create new child.



CROSSOVER → MUTATION →

1.
GENERATE GENEPOOL

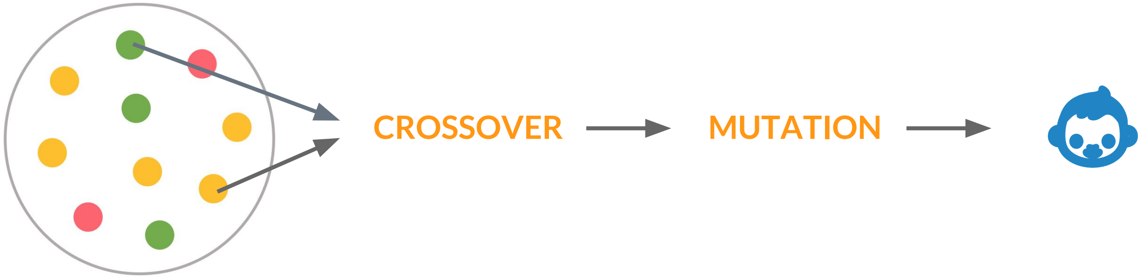


2.
MODIFY POPULATION

➔ Sort genepool.

➔ Check best candidate in genepool. Return if target is found.

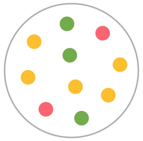
➔ Select two (2) random parents to create new child.



CROSSOVER → MUTATION →

➔ If **new child > worst** in genepool, replace.

1.
GENERATE GENEPOOL



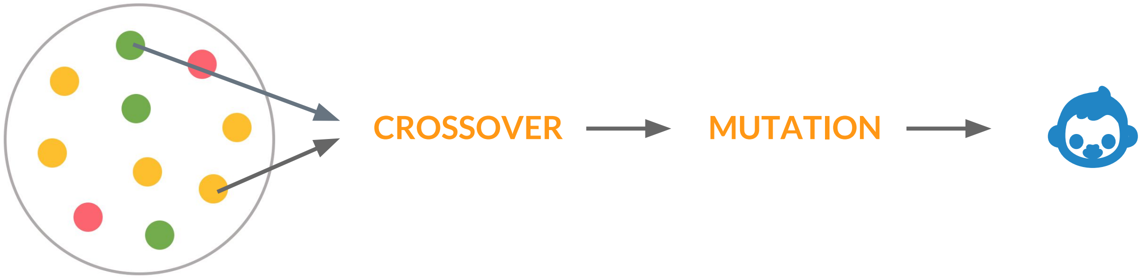
`while (true)`

2.
MODIFY POPULATION

→ Sort genepool.

→ Check best candidate in genepool. Return if target is found.

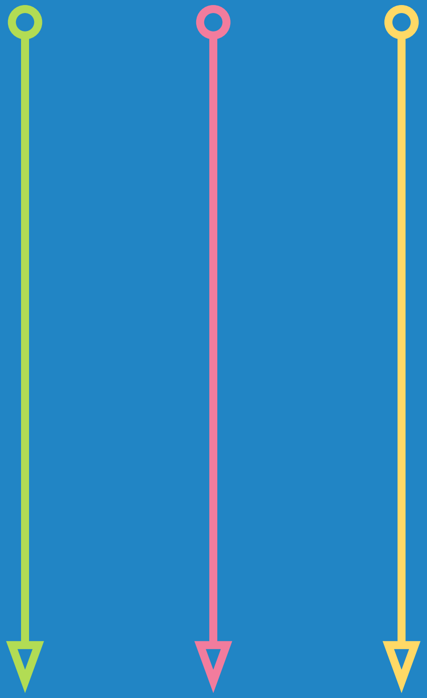
→ Select two (2) random parents to create new child.



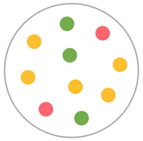
CROSSOVER → MUTATION →

→ If **new child > worst** in genepool, replace.

Parallel



1.
GENERATE GENEPOOL



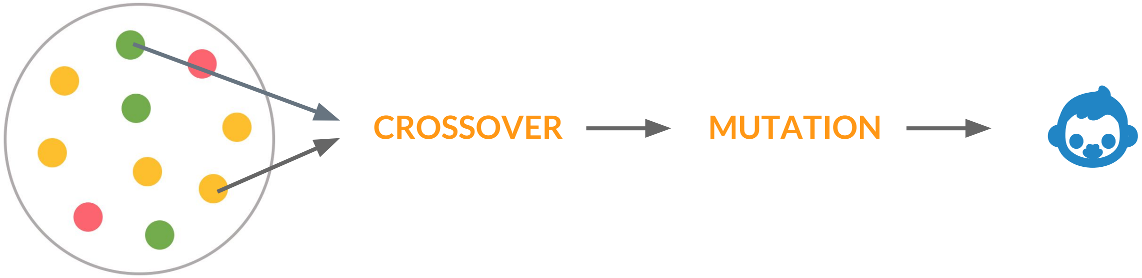
`while (true)`

2.
MODIFY POPULATION

→ Sort genepool.

→ Check best candidate in genepool. Return if target is found.

→ Select two (2) random parents to create new child.



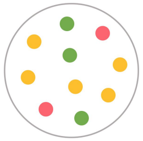
CROSSOVER → MUTATION →

→ If **new child > worst** in genepool, replace.

METHOD A

1.

GENERATE GENEPOOL

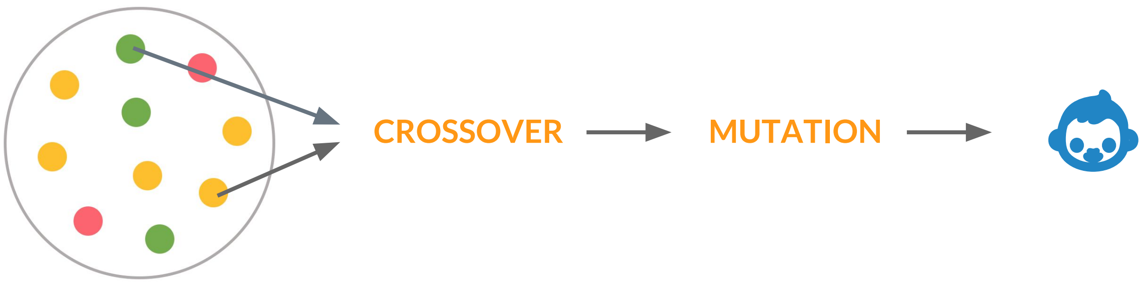


```
while (true)
```

2.

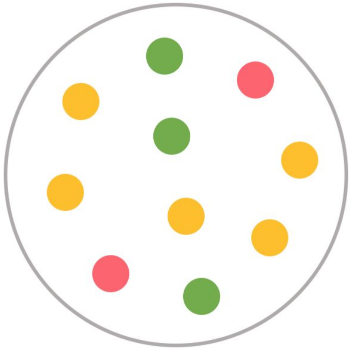
MODIFY POPULATION

- ➔ Sort genepool.
- ➔ Check best candidate in genepool. Return if target is found.
- ➔ Select two (2) random parents to create new child.

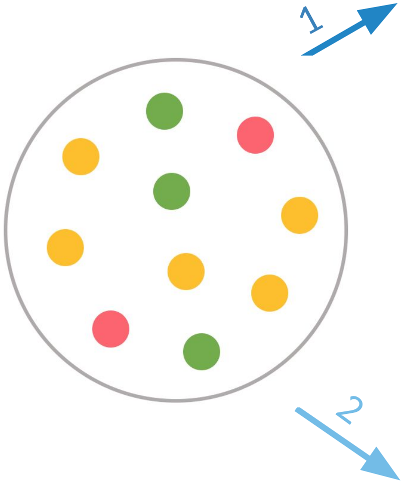


- ➔ If **new child > worst** in genepool, replace.

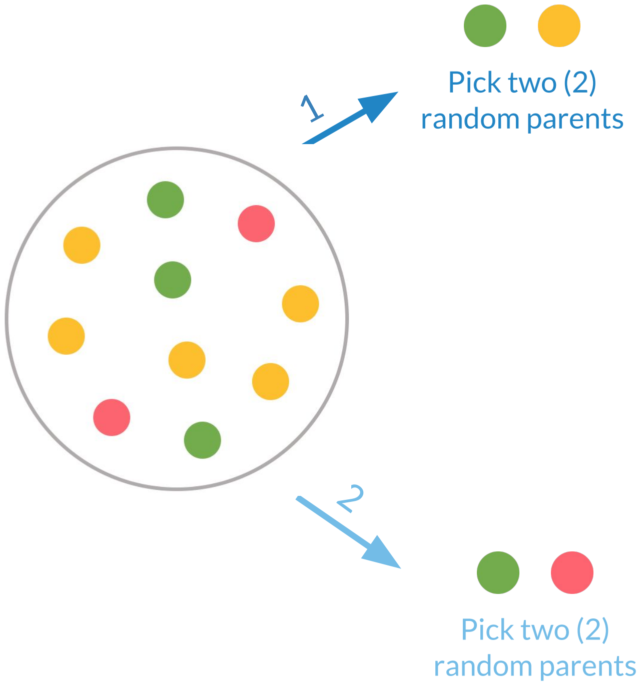
METHOD **A**



METHOD A



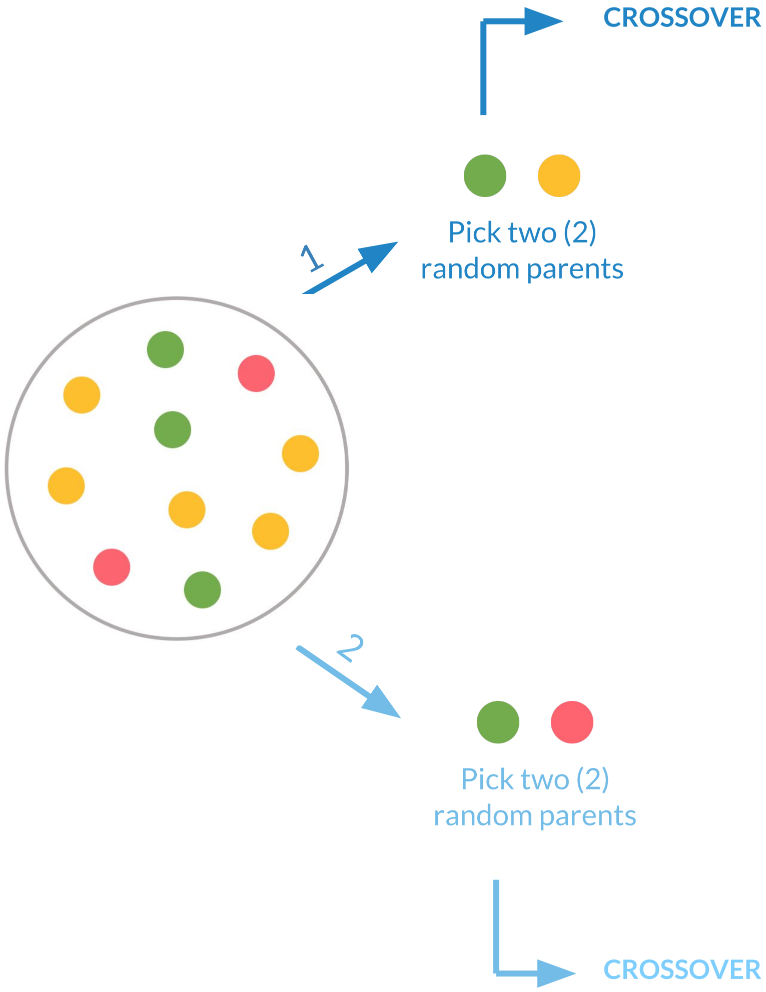
METHOD A



Pick two (2) random parents

Pick two (2) random parents

METHOD A



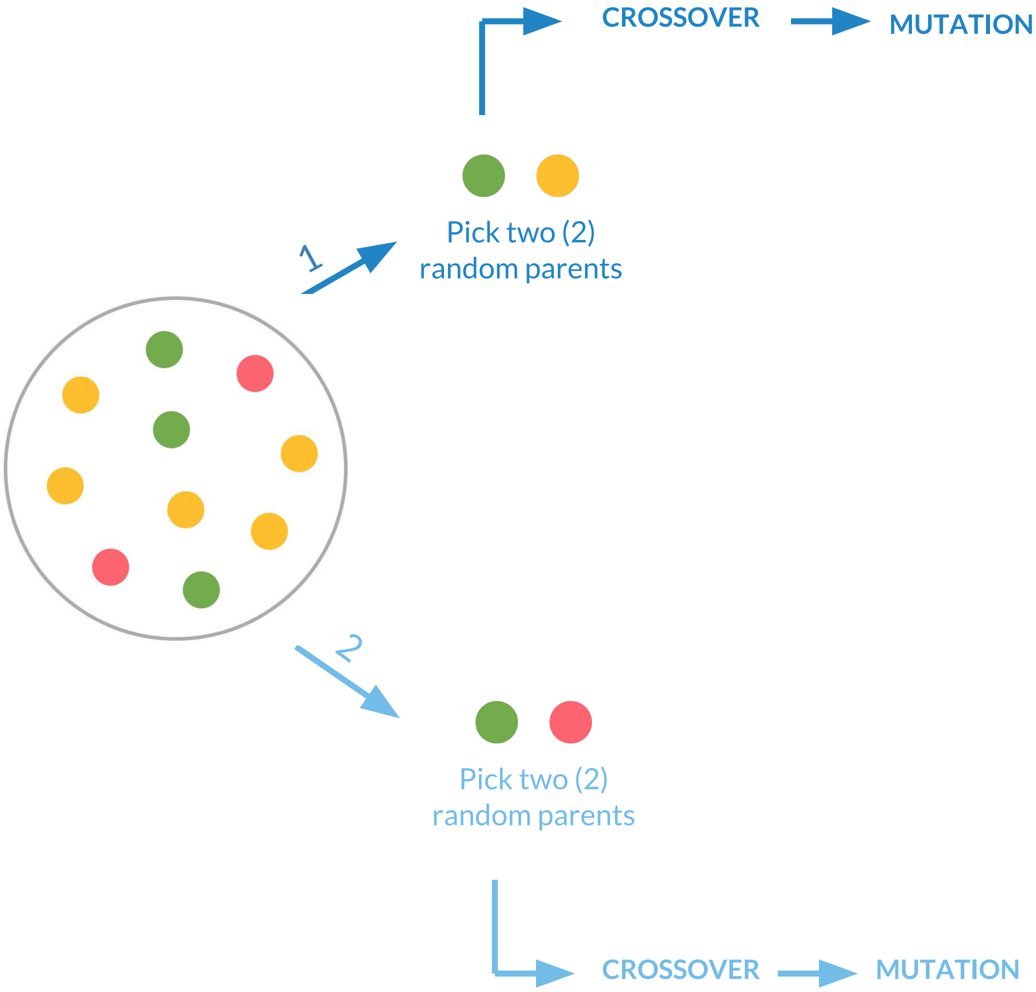
CROSSOVER

Pick two (2)
random parents

Pick two (2)
random parents

CROSSOVER

METHOD A



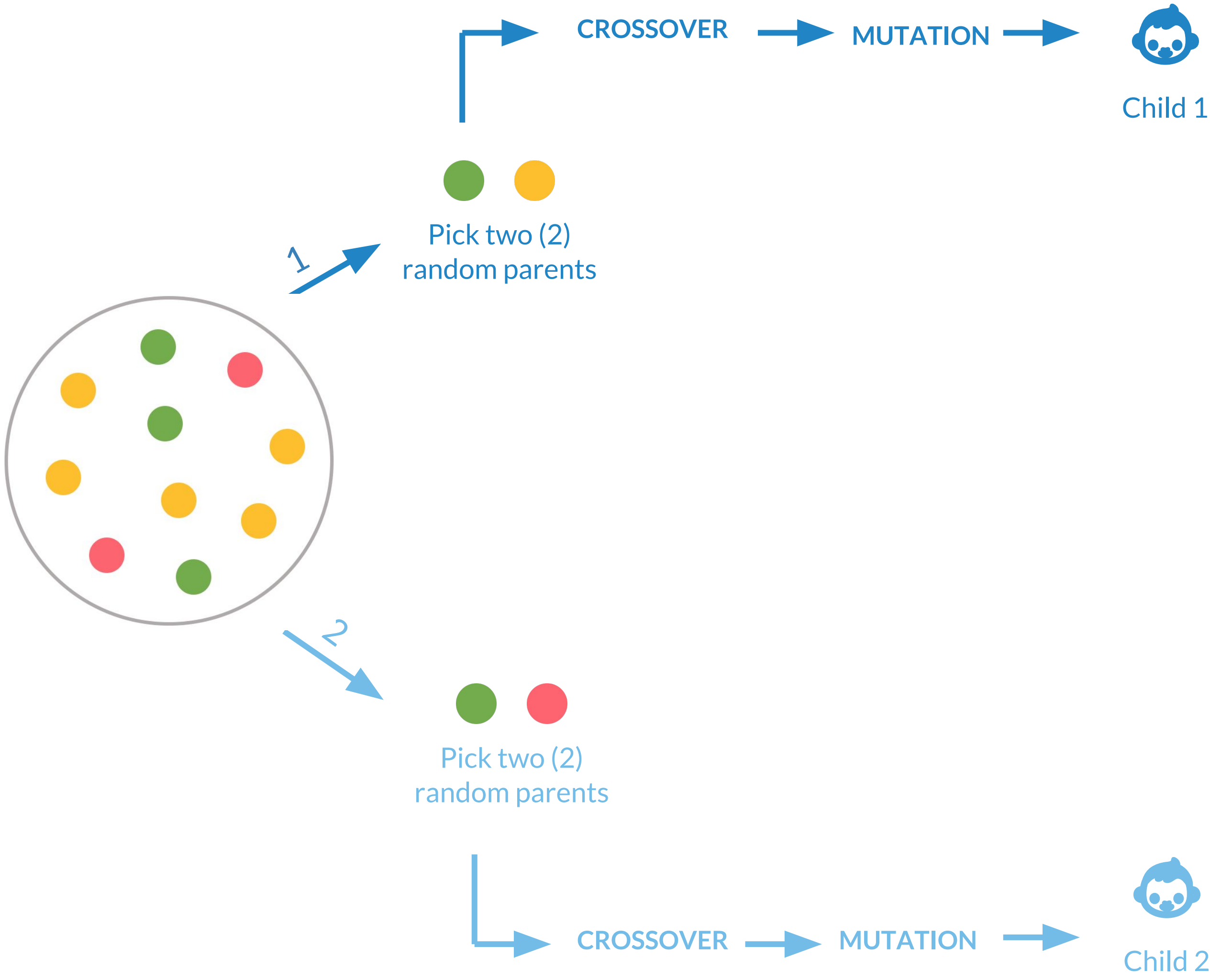
CROSSOVER → MUTATION

Pick two (2) random parents

Pick two (2) random parents

CROSSOVER → MUTATION

METHOD A



1

Pick two (2)
random parents

CROSSOVER

MUTATION

Child 1

2

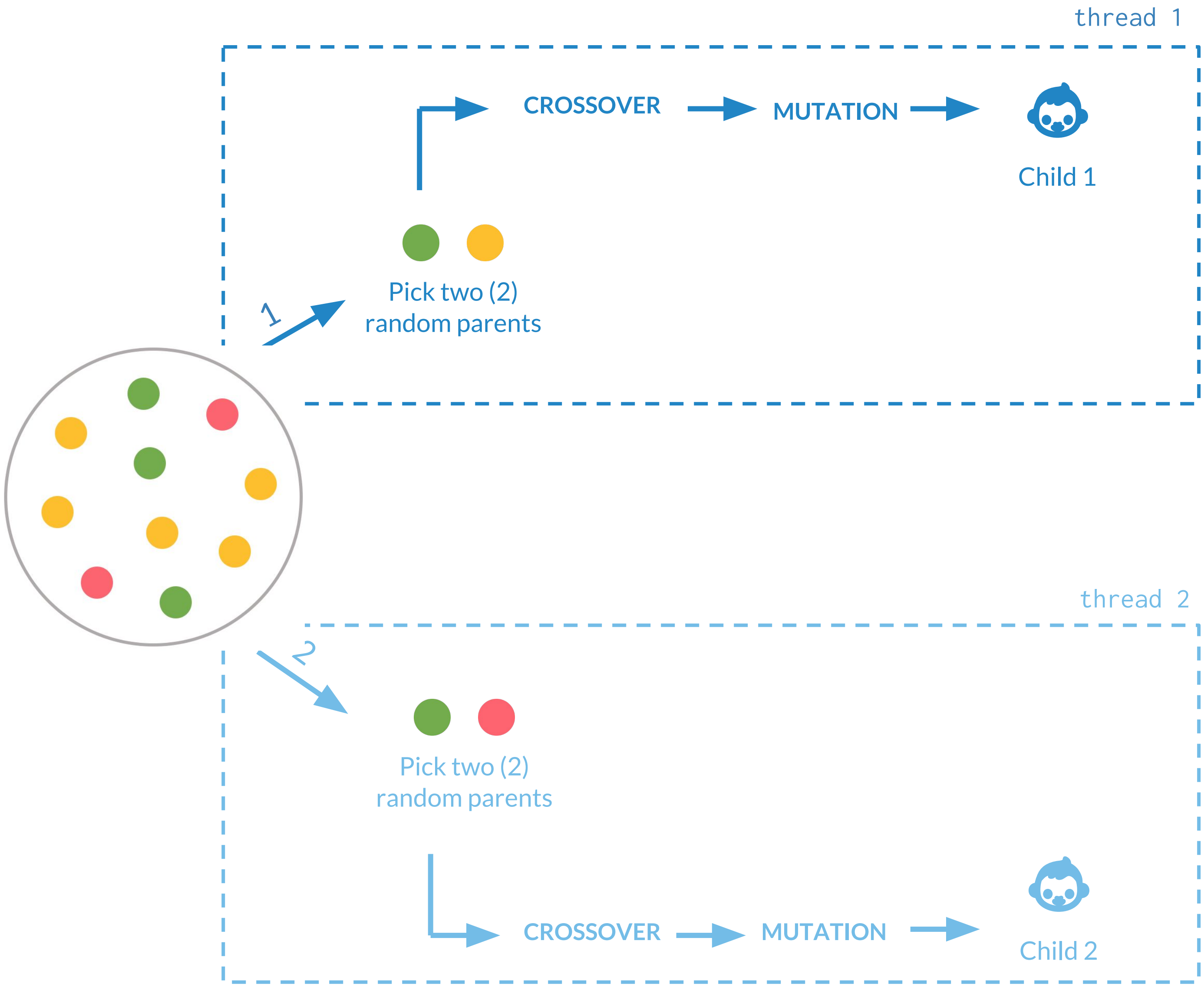
Pick two (2)
random parents

CROSSOVER

MUTATION

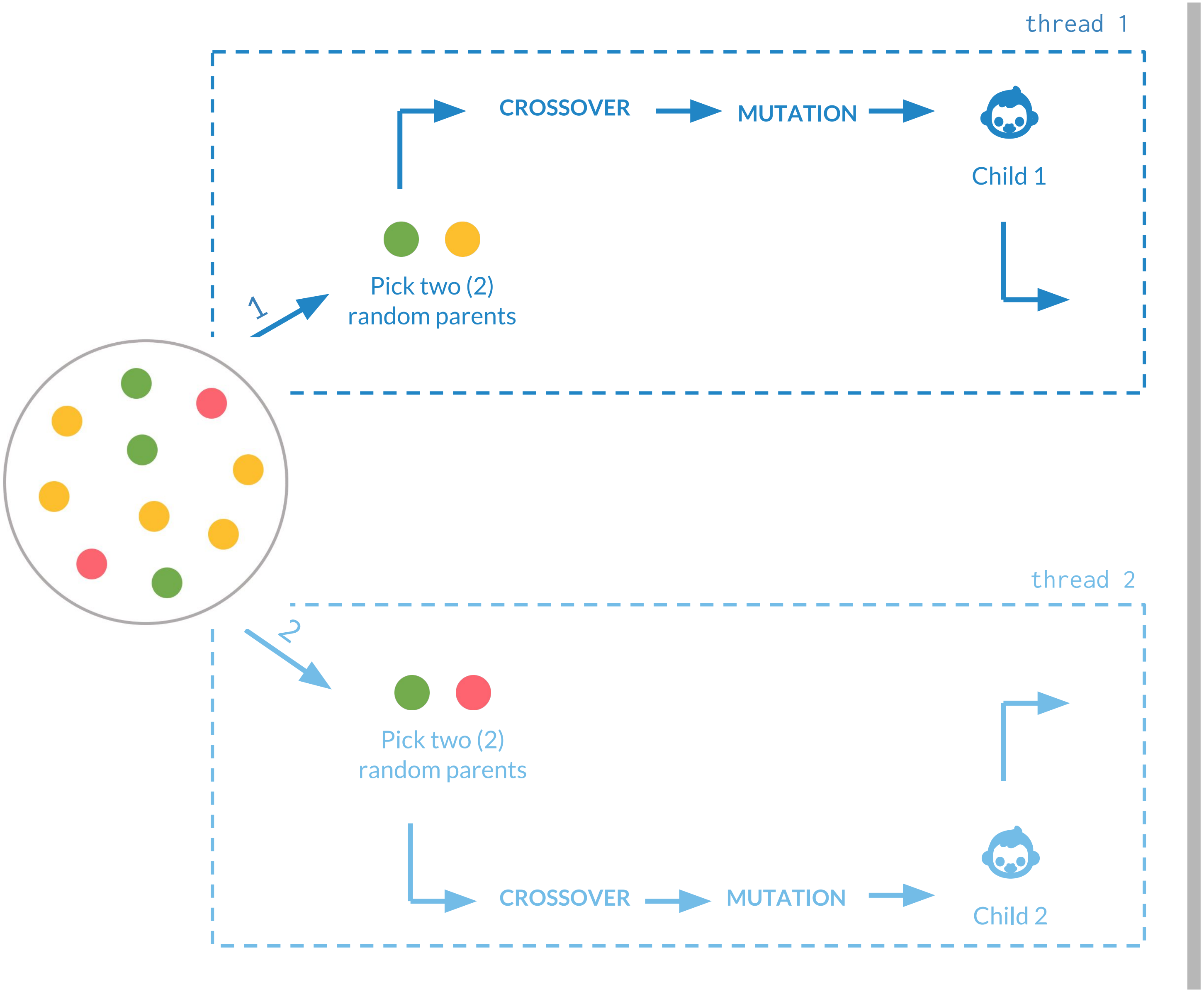
Child 2

METHOD **A**

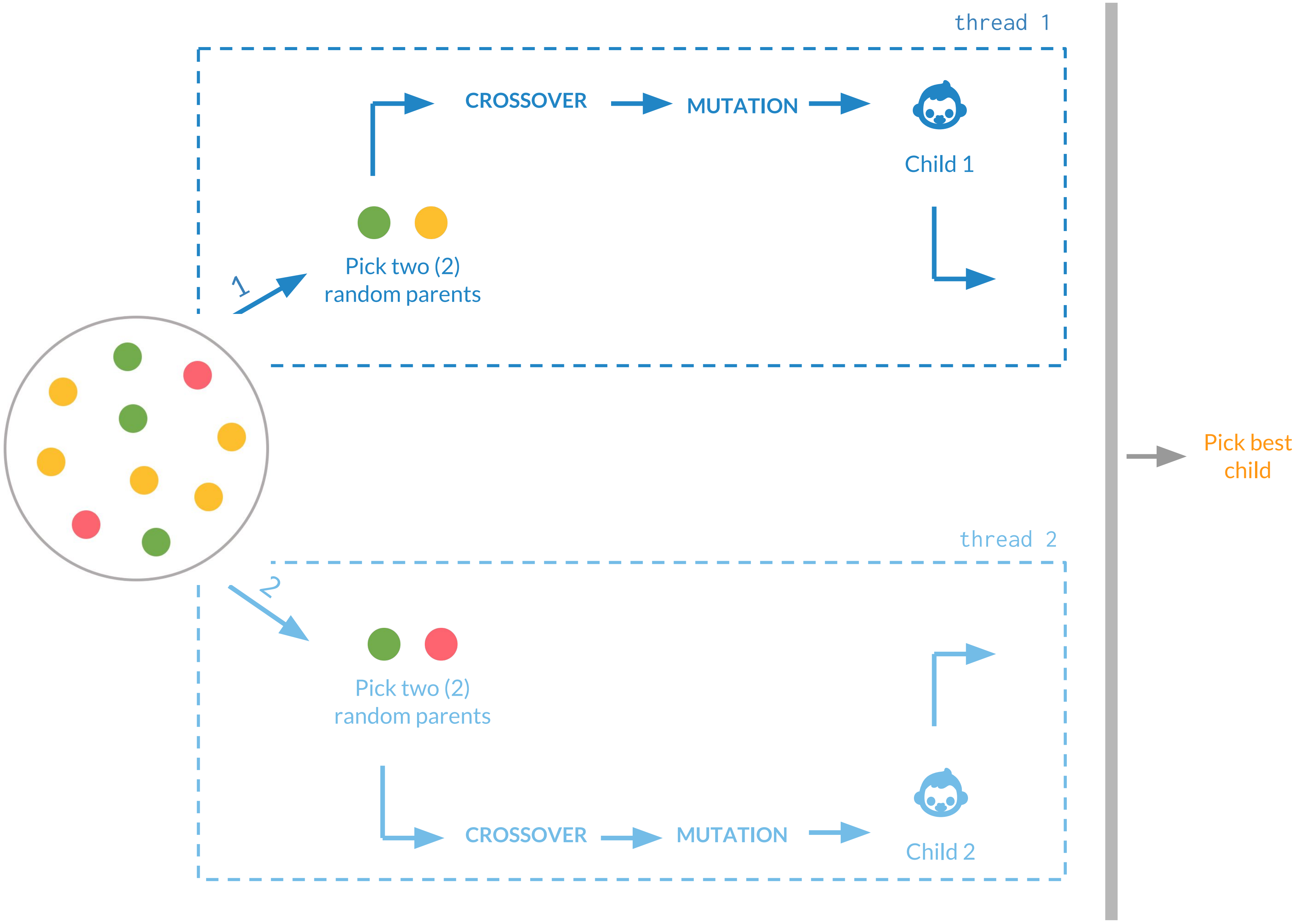


thread 1
CROSSOVER
MUTATION
Child 1
1
Pick two (2)
random parents
thread 2
2
Pick two (2)
random parents
CROSSOVER
MUTATION
Child 2

METHOD A


thread 1
CROSSOVER
MUTATION
Child 1
1
Pick two (2)
random parents
thread 2
2
Pick two (2)
random parents
CROSSOVER
MUTATION
Child 2

METHOD A



thread 1

1

Pick two (2) random parents

CROSSOVER

MUTATION

Child 1

thread 2

2

Pick two (2) random parents

CROSSOVER

MUTATION

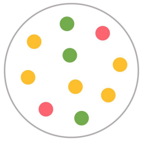
Child 2

Pick best child

METHOD **B**

METHOD **B**

1.
GENERATE GENEPOOL



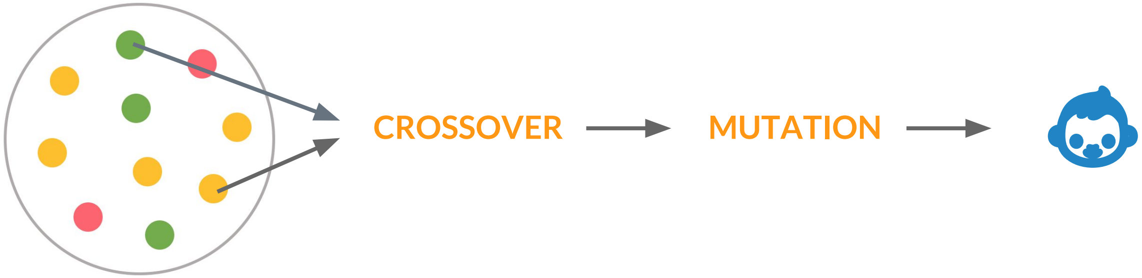
`while (true)`

2.
MODIFY POPULATION

→ Sort genepool.

→ Check best candidate in genepool. Return if target is found.

→ Select two (2) random parents to create new child.



CROSSOVER → MUTATION →

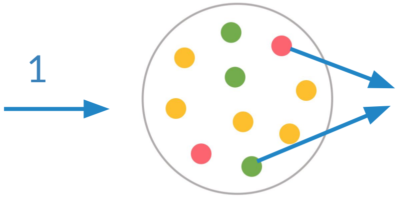
→ If **new child > worst** in genepool, replace.

1 →

2 →

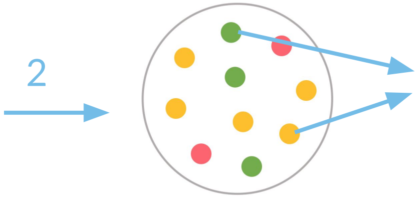
METHOD **B**

METHOD **B**



1

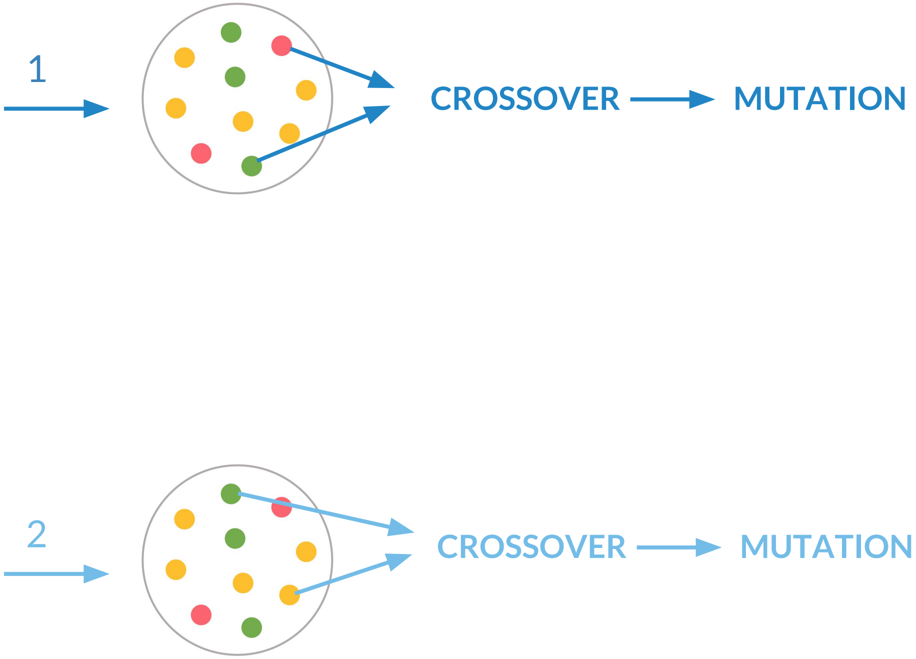
CROSSOVER



2

CROSSOVER

METHOD **B**



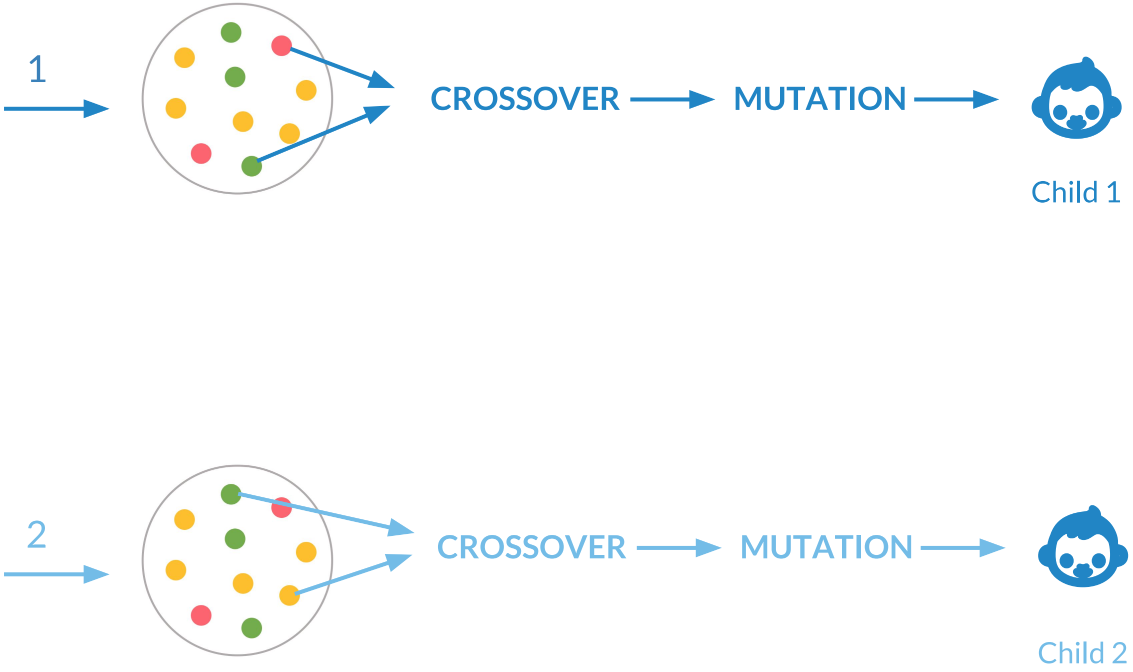
1

CROSSOVER → MUTATION

2

CROSSOVER → MUTATION

METHOD **B**



1

CROSSOVER → MUTATION →

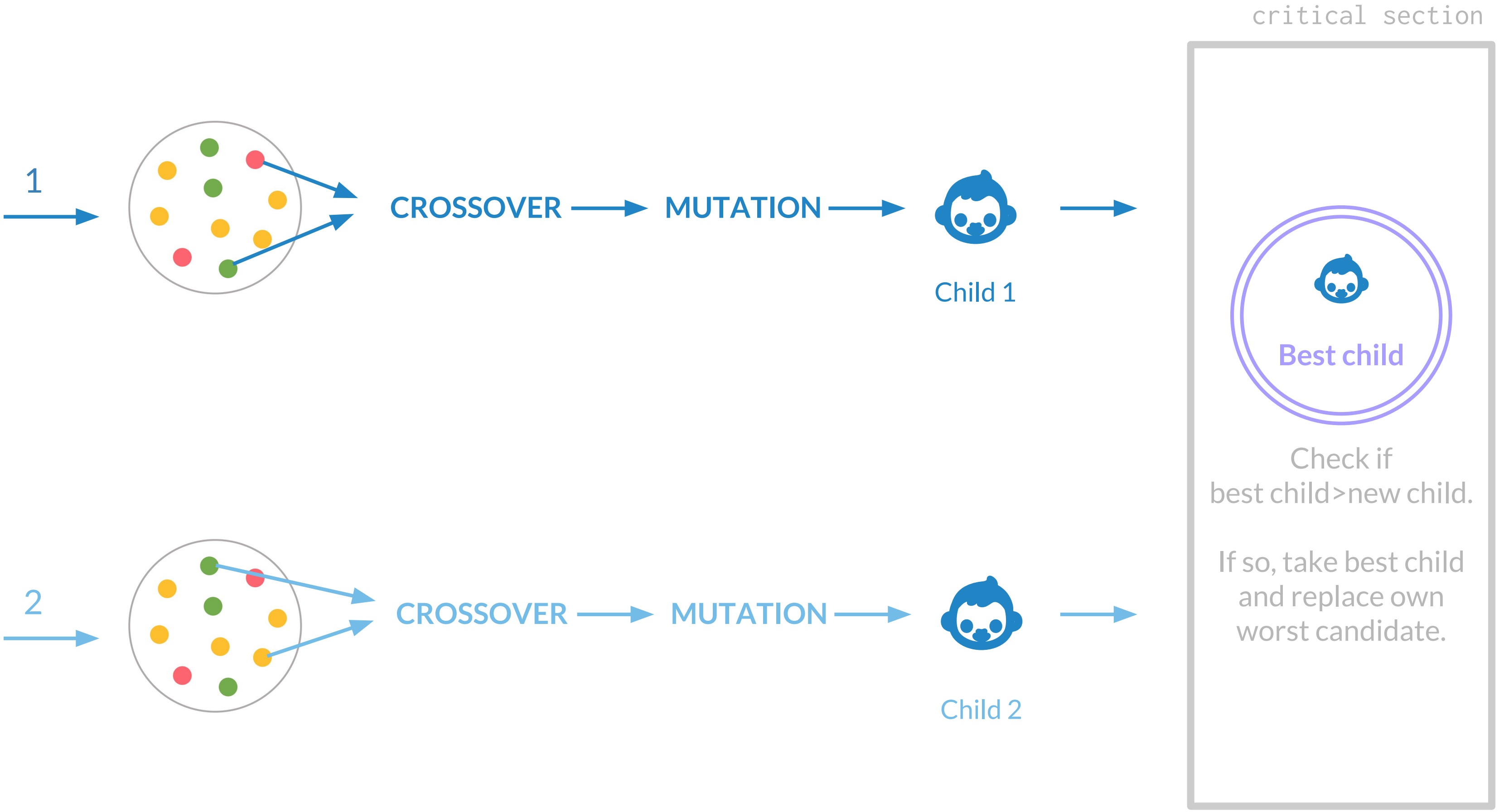
Child 1

2

CROSSOVER → MUTATION →

Child 2

METHOD B



critical section

1

CROSSOVER → MUTATION →

Child 1

2

CROSSOVER → MUTATION →

Child 2

Best child

Check if
best child>new child.

If so, take best child
and replace own
worst candidate.

Results & Analysis

TOOLS USED

Python

Prototyping of sequential genetic algorithm.

Java

- Java Threads

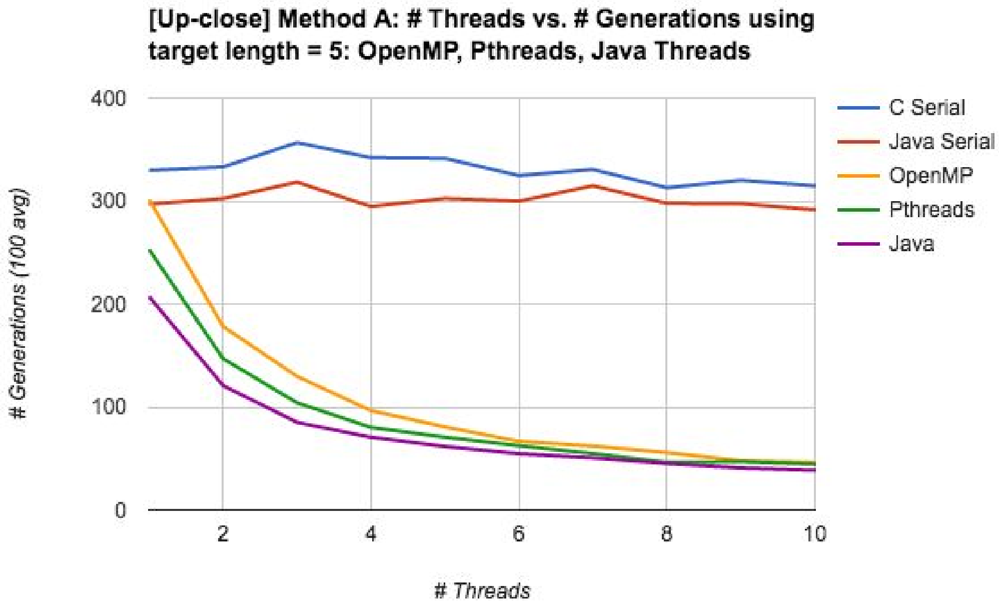
C

- OpenMP
- Pthreads

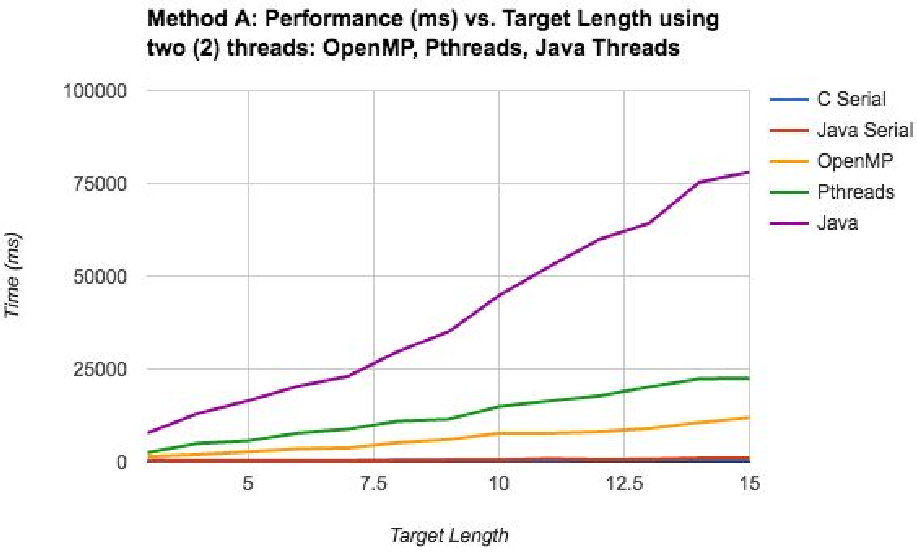
HOW WAS THE DATA EVALUATED?

- Number of generations vs. Number of threads
- Time performance vs. Length of target

METHOD A



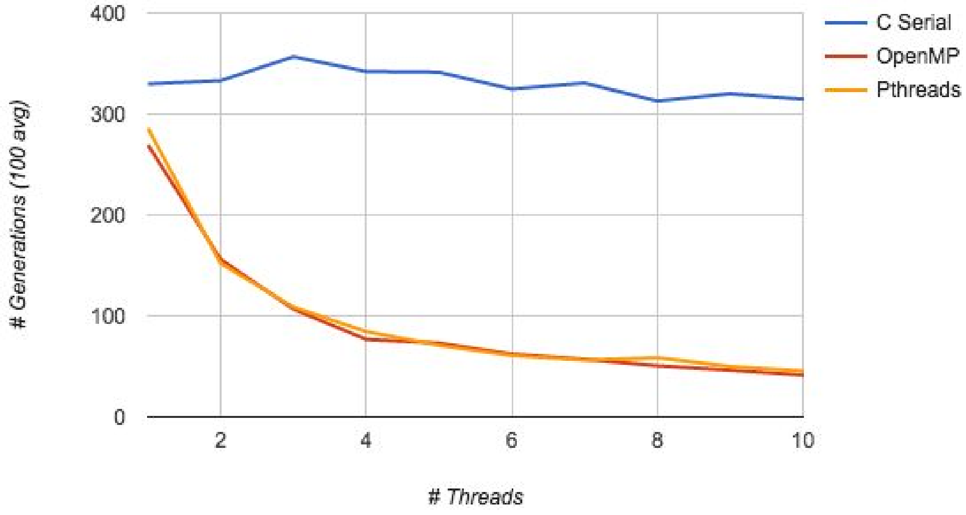
[Up-close] Method A: # Threads vs. # Generations using target length = 5: OpenMP, Pthreads, Java Threads



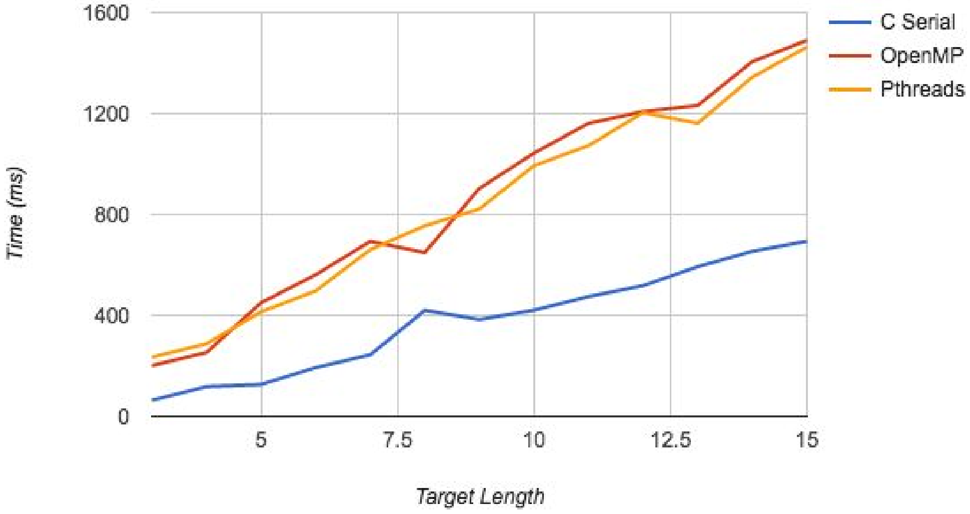
Method A: Performance (ms) vs. Target Length using two (2) threads: OpenMP, Pthreads, Java Threads

METHOD B

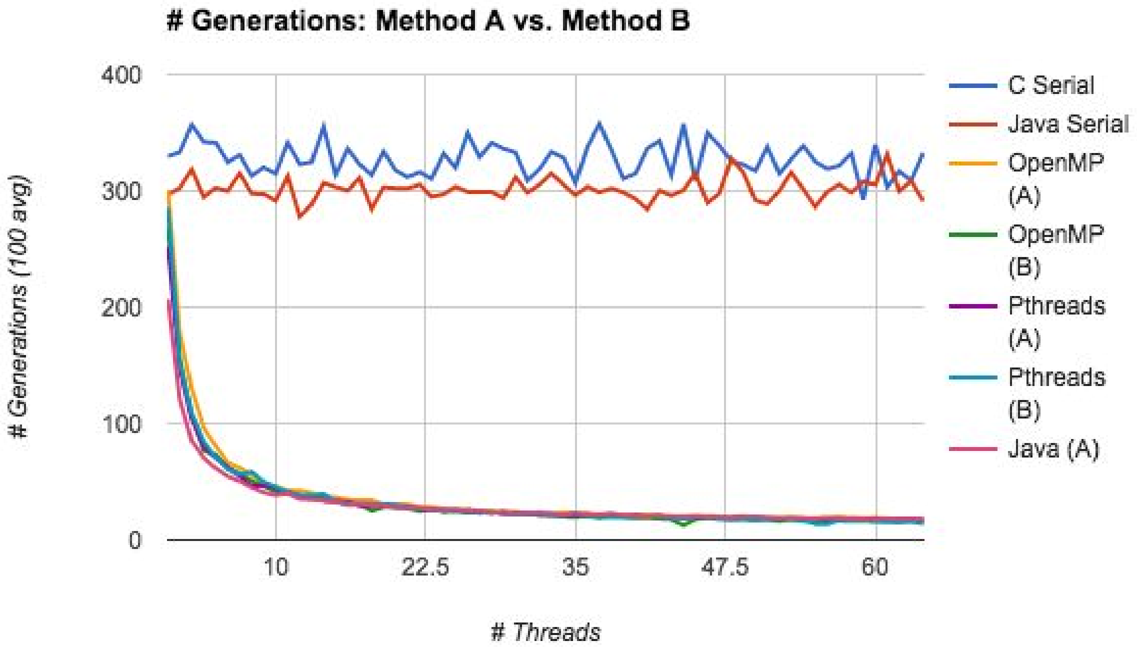
[Up-close] Method B: # Threads vs. # Generations using target length = 5: OpenMP, Pthreads

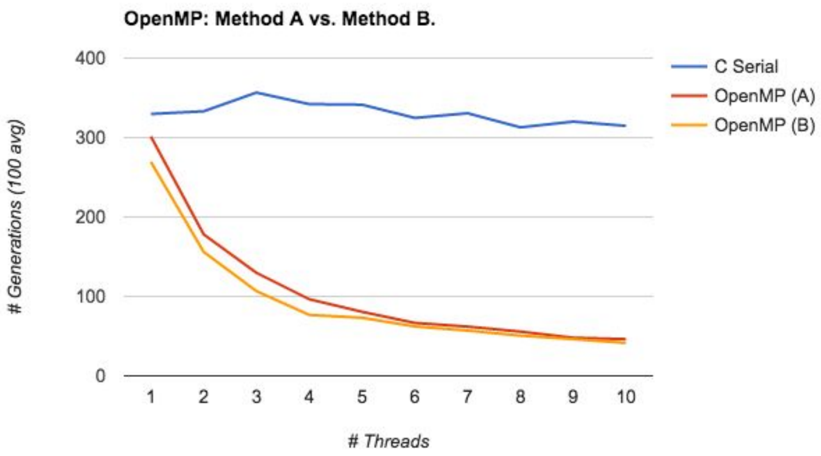


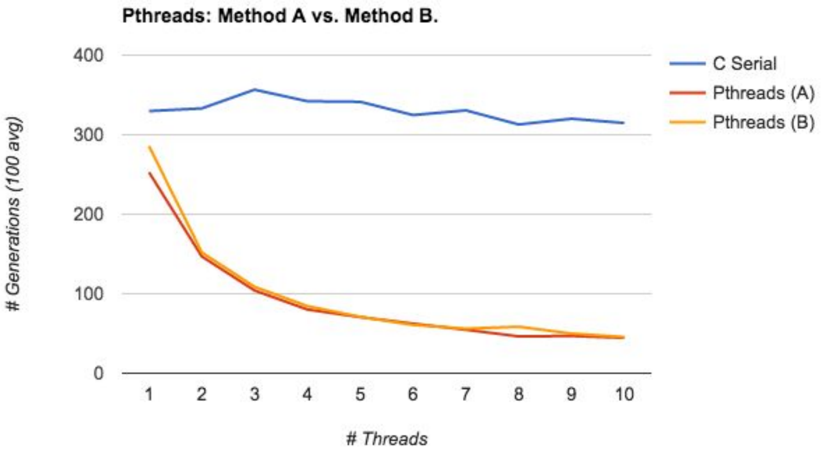
Method B: Performance (ms) vs. Target Length using two (2) threads: OpenMP, Pthreads



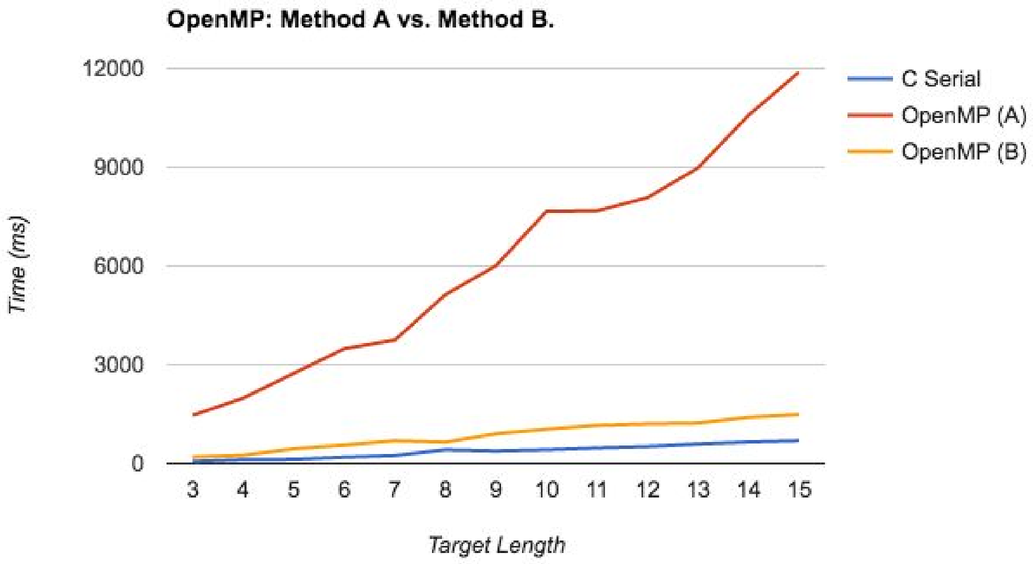
METHOD A VS. METHOD B (Gens)


Generations: Method A vs. Method B

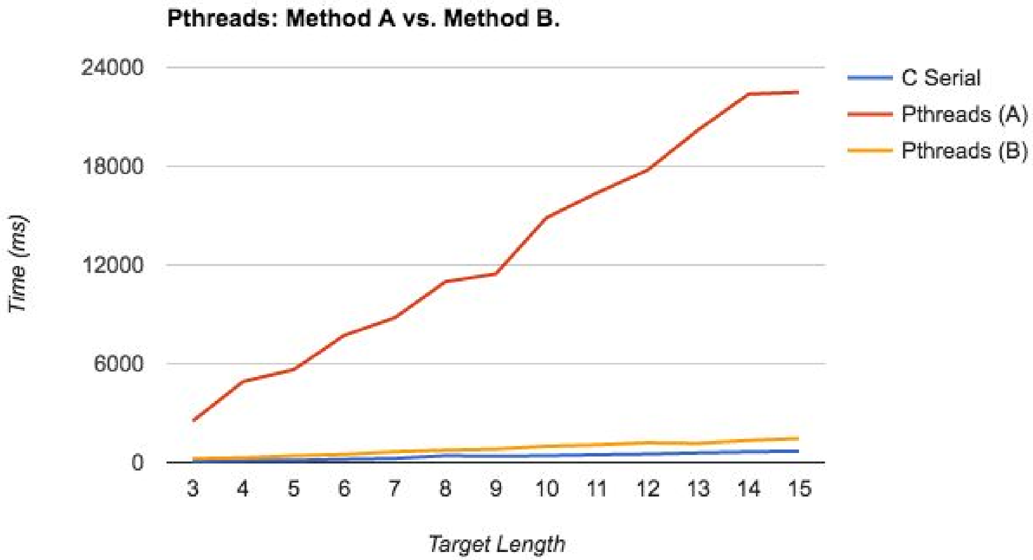

OpenMP: Method A vs. Method B.


Pthreads: Method A vs. Method B.

METHOD A vs. METHOD B (Time)



OpenMP: Method A vs. Method B.



Pthreads: Method A vs. Method B.

CONCLUSION

	METHOD A	METHOD B
OpenMP	🙂	🙂
Pthreads	😕	🙂
Java Threads	☹	⊘

* Time performance comparison in microseconds as a function of target length.

Alexandar Mihaylov

Luisa Rojas

PARALLEL GENETIC ALGORITHMS