An Experimental Evaluation for OneMax Problem

cGA versus GA

Fabio López Pires

Genetic Algorithm (GA)

Algorithm 3.2.1: Pseudocode for the Genetic Algorithm.

```
Input: Population_size, Problem_size, P_crossover, P_mutation
Output: S_best
Population ← InitializePopulation(Population_size, Problem_size);
EvaluatePopulation(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    Parents ← SelectParents(Population, Population_size);
    Children ← ∅;
    foreach Parent_1, Parent_2 ∈ Parents do
        Child_1, Child_2 ← Crossover(Parent_1, Parent_2, P_crossover);
        Children ← Mutate(Child_1, P_mutation);
        Children ← Mutate(Child_2, P_mutation);
    end
    EvaluatePopulation(Children);
    S_best ← GetBestSolution(Children);
    Population ← Replace(Population, Children);
end
return S_best;
```

Compact Genetic Algorithm (cGA)

Algorithm 5.4.1: Pseudocode for the cGA.

```
Input: Bits_num, n
Output: S_best
V ← InitializeVector(Bits_num, 0.5);
S_best ← ∅;
while ¬StopCondition() do
    S_1 ← GenerateSamples(V);
    S_2 ← GenerateSamples(V);
    S_winner, S_loser ← SelectWinnerAndLoser(S_1, S_2);
    if Cost(S_winner) ≤ Cost(S_best) then
        S_best ← S_winner;
    end
    for i to Bits_num do
        if S^i_winner ≠ S^i_loser then
            if S^i_winner ≡ 1 then
                V^i_i ← V^i_i + 1/n;
            else
                V^i_i ← V^i_i − 1/n;
            end
        end
    end
end
return S_best;
```

OneMax Problem

- The problem is a maximizing binary optimization problem called OneMax.
- Seeks a binary string of unity (all '1' bits).
- The objective function only provides an indication of the number of correct bits in a candidate string, not the positions of the correct bits.
- The objective function is the sum of the elements of the string.

Experimental Environment

root@flopez-Vostro-1720: /home/flopez/Downloads/TP03

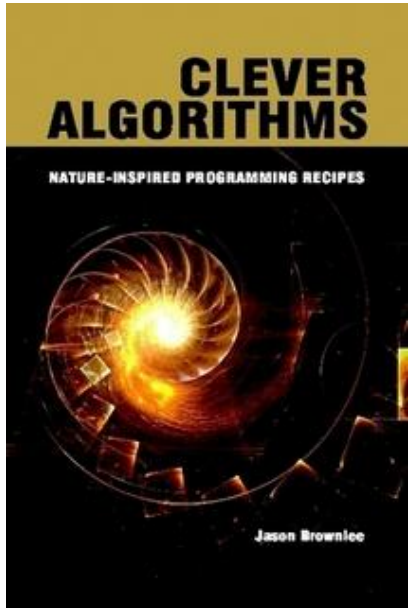
```
#!/bin/bash
for population in 100 200 300 400 500
do
        for numb_bits in 16 32 64 128 256 512
        do
                for corrida in 1 2 3 4 5 6 7 8 9 10
                do
                        ruby GA.rb $numb_bits $population > logs/GA.log.$numb_bits-$population-$corrida
                        ruby cGA.rb $numb_bits $population > logs/cGA.log.$numb_bits-$population-$corrida
                done
        done
done
```

Ruby PROGRAMMING Language

ubuntu

ruby-prof / **ruby-prof**

Different population size:	100, 200, 300, 400, 500
Different problem size:	16, 32, 64, 128, 256, 512
Different metrics:	comparisons, memory, CPU
Number runs each instance:	10
Different algorithms:	cGA, GA
Total:	**1800 runs**



Brownlee, Jason. Clever algorithms: nature-inspired programming recipes. Jason Brownlee, 2011.

Experimental Metrics

- M1. Number of Iterations / Generations
  - For different problem sizes and population:
    - Average
    - Standard Deviation
    - Minimum and Maximum

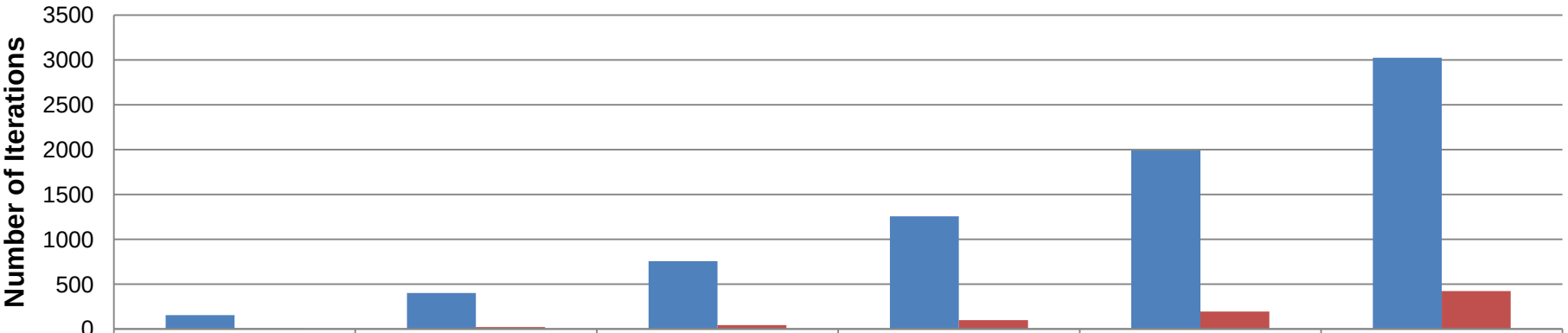
- M2. Number of Comparisons
  - For different problem sizes and population:
    - Average
    - Standard Deviation
    - Minimum and Maximum

Experimental Metrics

- M3. Memory Usage
  - For different problem sizes and population:
    - Average
    - Standard Deviation
    - Minimum and Maximum

- M4. CPU Time
  - For different problem sizes and population:
    - Average
    - Standard Deviation
    - Minimum and Maximum

M1. Iterations / Generations



	16	32	64	128	256	512
cGA	155,100	401,000	757,300	1256,800	1992,200	3025,200
GA	5,700	20,800	44,200	100,400	194,800	424,200

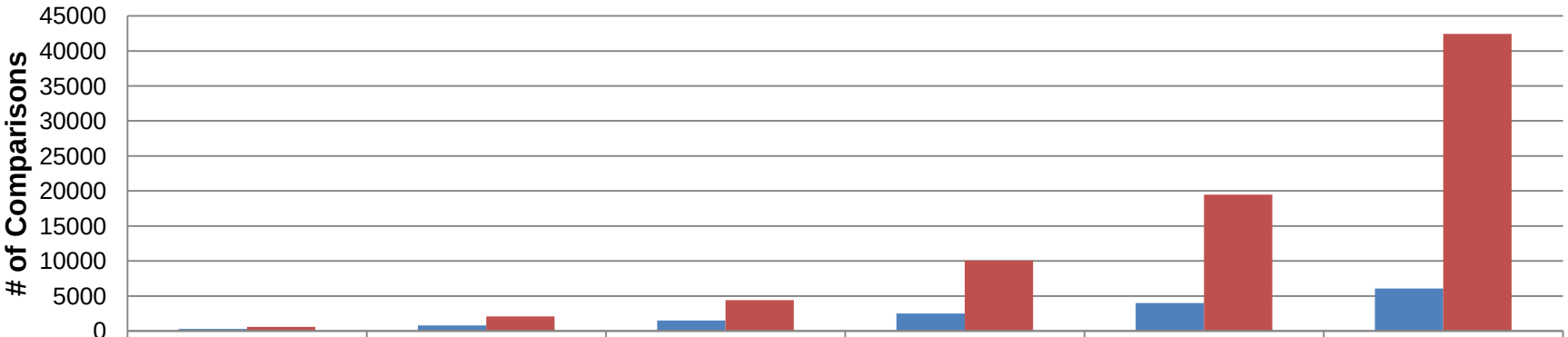
In cGA, the vector update parameter (n) may be considered to be comparable to the population size parameter in the GA.

Observation #1: Iterations in cGA are not Generations in GA.

Considering that n in cGA is population size in GA, we define M2 as:
For cGA = 2 * n (considering that 2 individuals are evaluated per iteration).
For GA = population size * number of generations (each individual is evaluated on each generation).

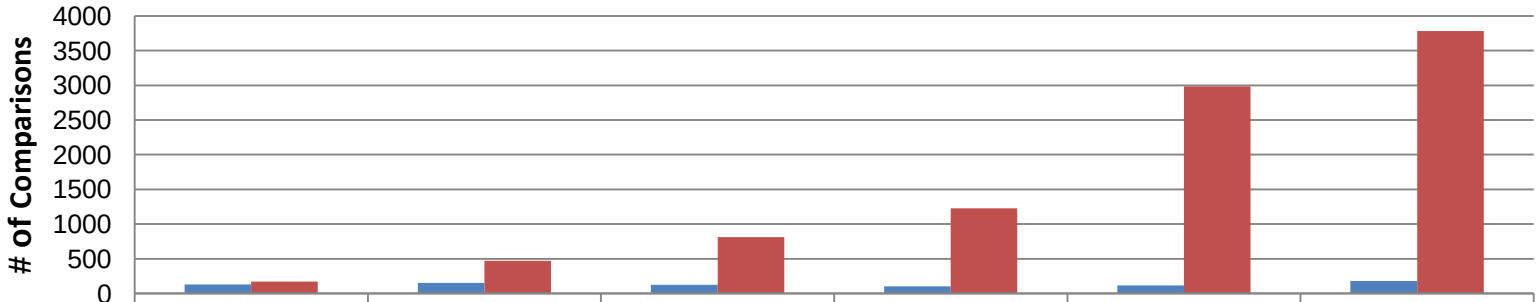
M2. Individuals Comparisons

Average Comparisons and Problem Size (Population 100)



	16	32	64	128	256	512
cGA	310,200	802,000	1514,600	2513,600	3984,400	6050,400
GA	570,000	2080,000	4420,000	10040,000	19480,000	42420,000

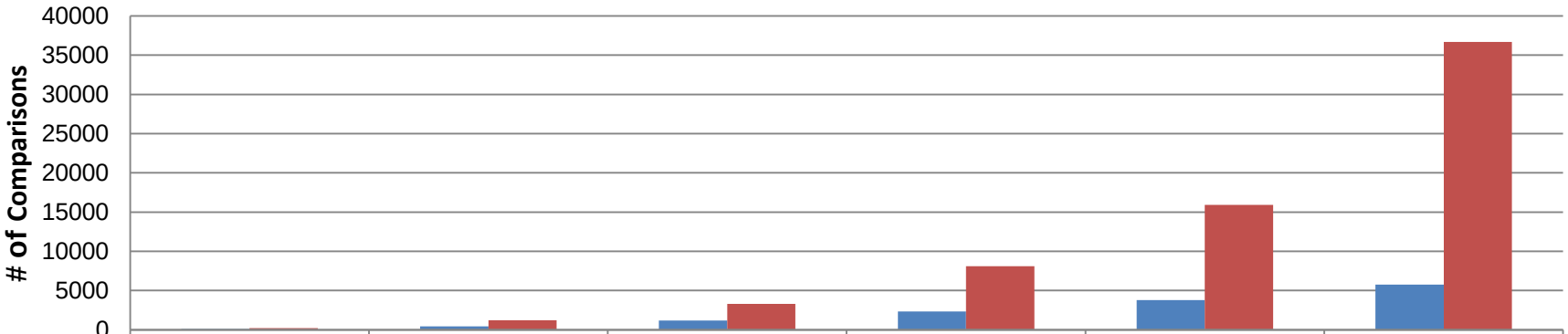
Standard Deviation and Problem Size (Population 100)



	16	32	64	128	256	512
cGA	129,822	151,886	126,115	101,792	113,199	178,781
GA	170,294	470,933	813,497	1225,833	2982,467	3781,769

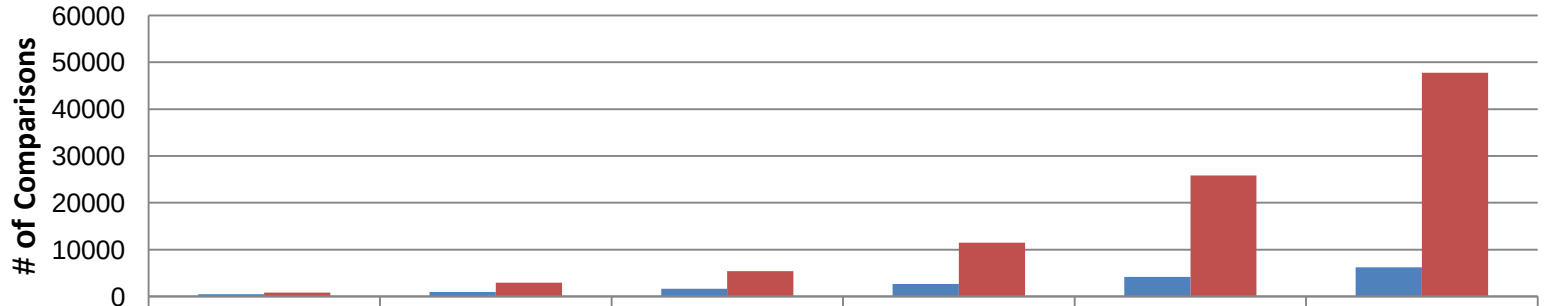
M2. Individuals Comparisons

Minimum Comparisons and Problem Size (Population 100)



	16	32	64	128	256	512
cGA	102,000	446,000	1188,000	2338,000	3764,000	5768,000
GA	200,000	1200,000	3300,000	8100,000	15900,000	36700,000

Maximum Iterations and Problem Size (Population 100)



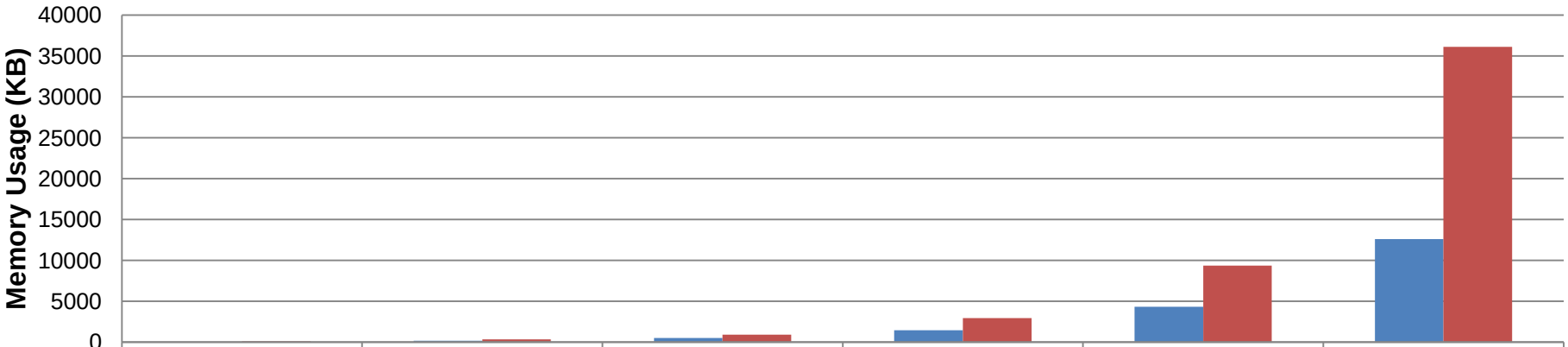
	16	32	64	128	256	512
cGA	486,000	932,000	1620,000	2676,000	4168,000	6244,000
GA	800,000	2900,000	5400,000	11500,000	25800,000	47800,000

M2. Individuals Comparisons

- cGA performed lower number of comparisons than GA (in all experiments)
  - As an example, with 100 individuals the cGA obtained between 45% and 85% less than GA in average.
    - 45.579% less for 16 bits
    - 61.442% less for 32 bits
    - 65.733% less for 64 bits
    - 74.964% less for 128 bits
    - 79.546% less for 256 bits
    - 85.737% less for 512 bits
  - Lower % of number of comparisons with larger number of bits.

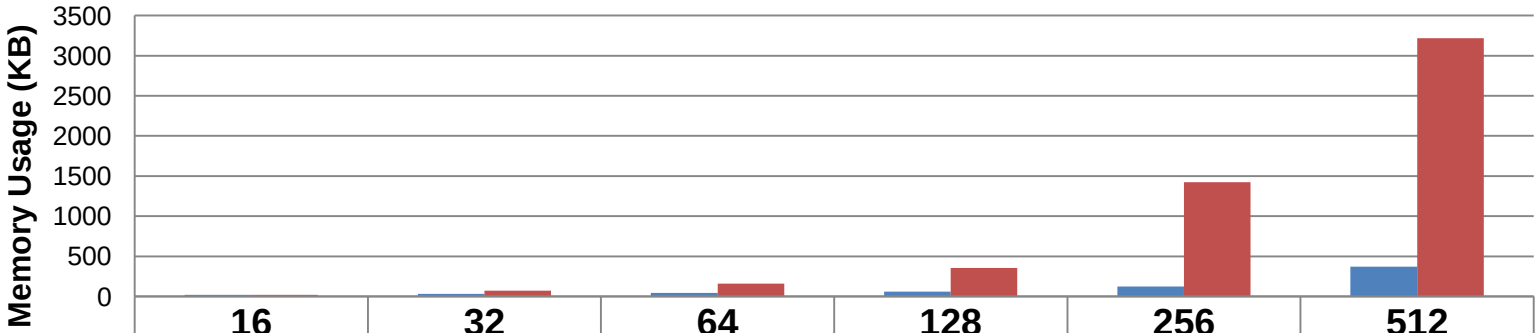
M3. Memory Usage

Average Memory Usage and Problem Size (Population 100)



	16	32	64	128	256	512
cGA	50,026	170,823	507,260	1467,146	4315,743	12603,593
GA	100,412	342,440	907,276	2960,757	9363,041	36120,844

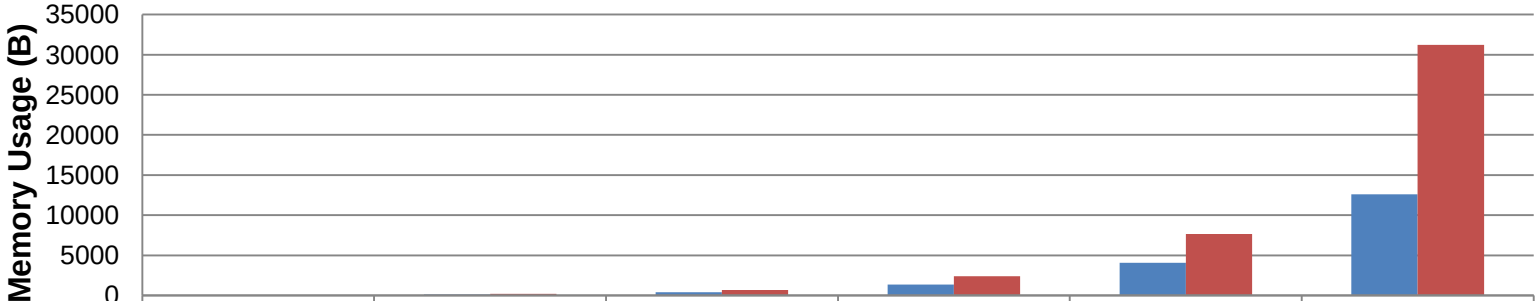
Standard Deviation and Problem Size (Population 100)



	16	32	64	128	256	512
cGA	18,573	31,223	41,689	59,097	122,319	371,967
GA	20,856	69,961	159,592	354,876	1421,416	3216,863

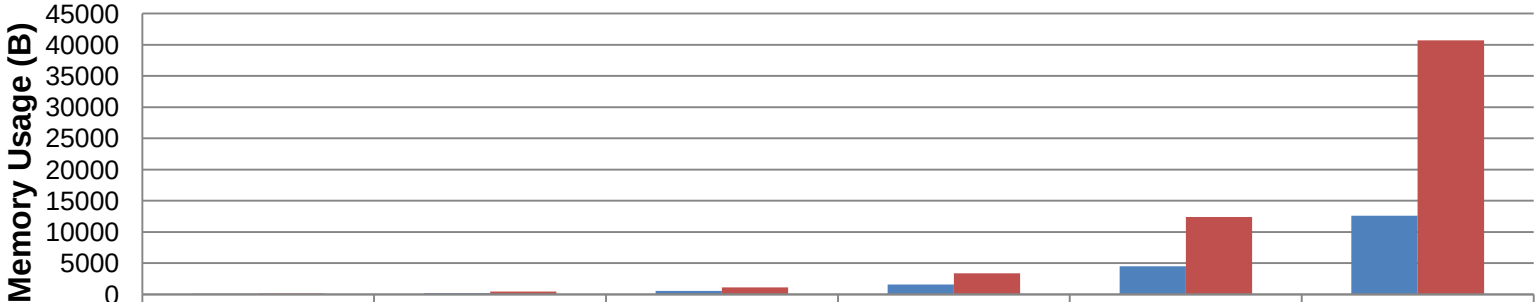
M3. Memory Usage

Minimum Memory Usage and Problem Size (100)



	16	32	64	128	256	512
cGA	20,239	97,642	399,297	1365,198	4077,586	12603,593
GA	55,123	211,895	688,753	2399,810	7645,267	31221,150

Maximum Memory Usage and Problem Size (100)

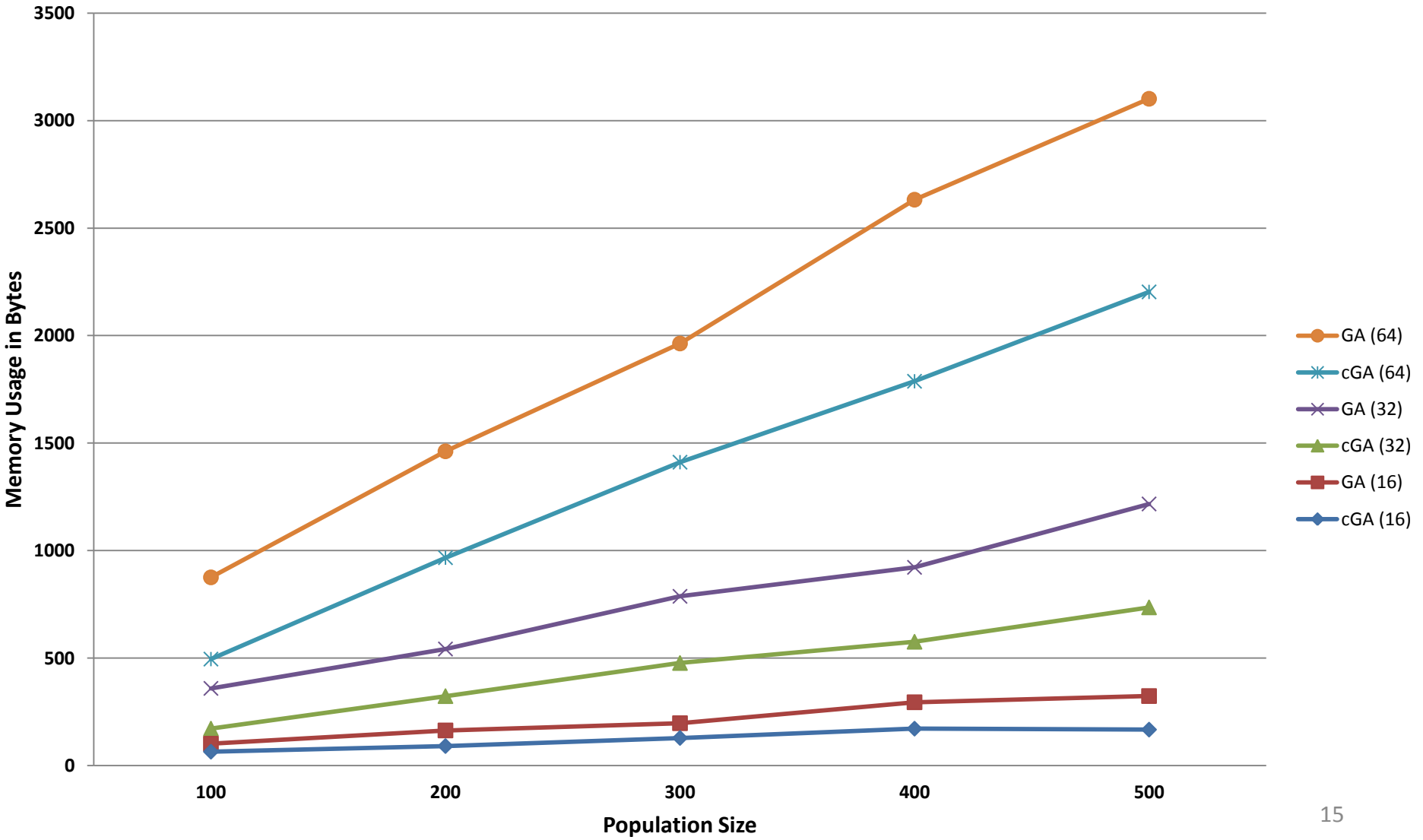


	16	32	64	128	256	512
cGA	75,177	197,547	542,102	1561,430	4514,135	12603,593
GA	128,701	464,612	1099,775	3385,326	12377,798	40698,580

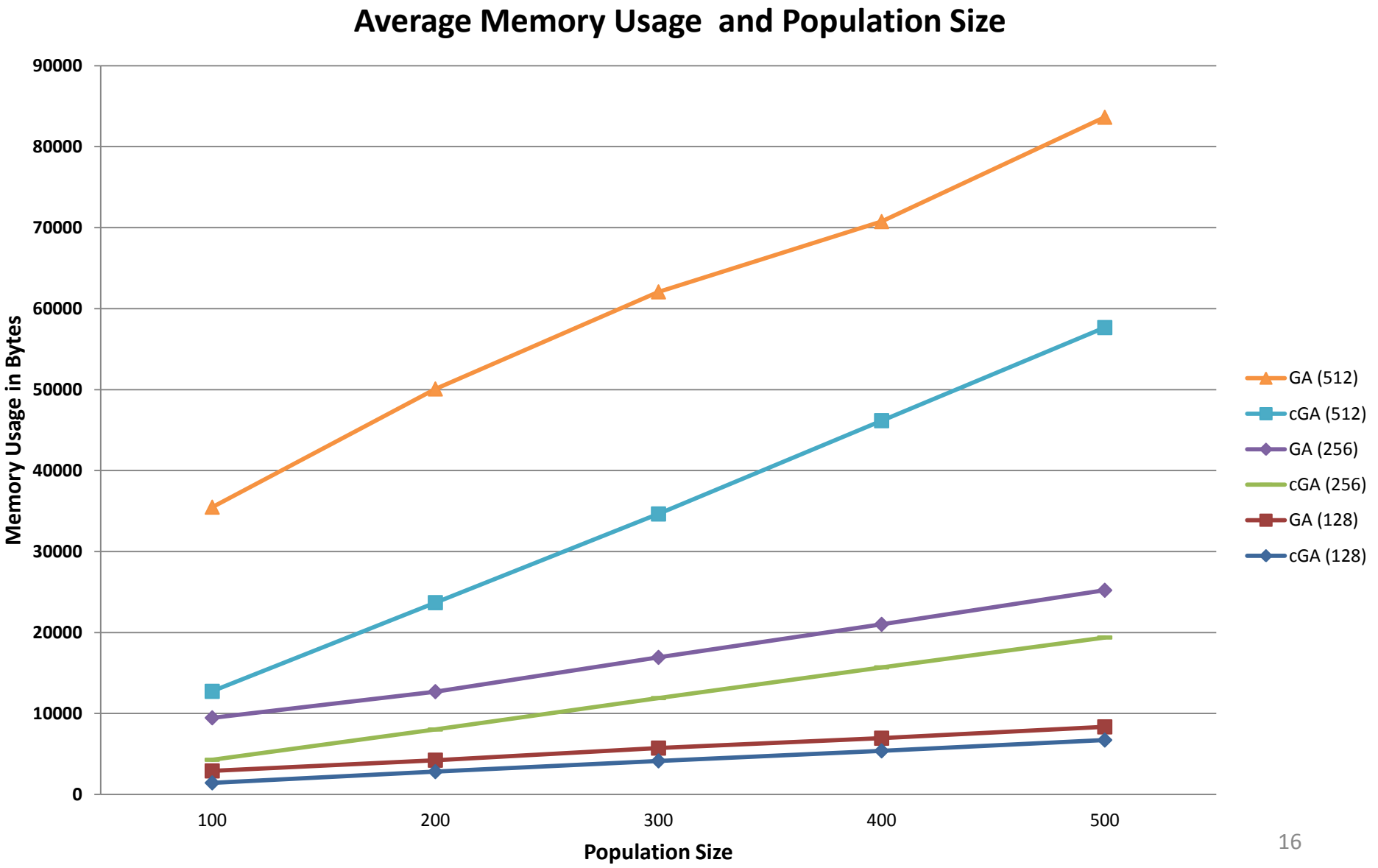
M3. Memory Usage

- cGA performed lower memory usage than GA (in all experiments)
  - As an example, with 100 individuals the cGA obtained between 50% and 65% less than GA in average.
    - 50.180% less for 16 bits
    - 50.116% less for 32 bits
    - 44.090% less for 64 bits
    - 50.447% less for 128 bits
    - 53.907% less for 256 bits
    - 65.107% less for 512 bits
  - Lower % of memory usage with larger number of bits.

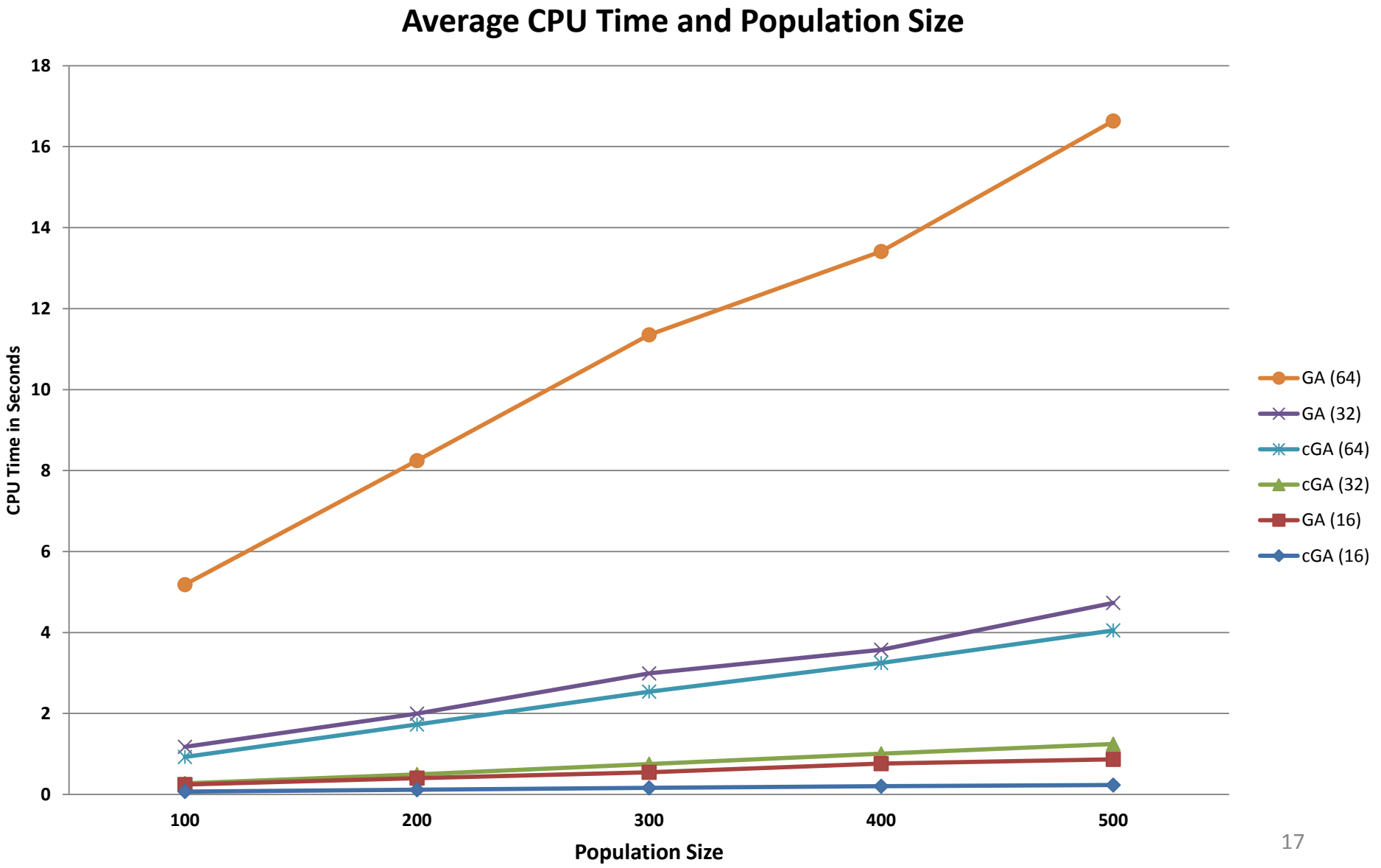
M3. Memory Usage



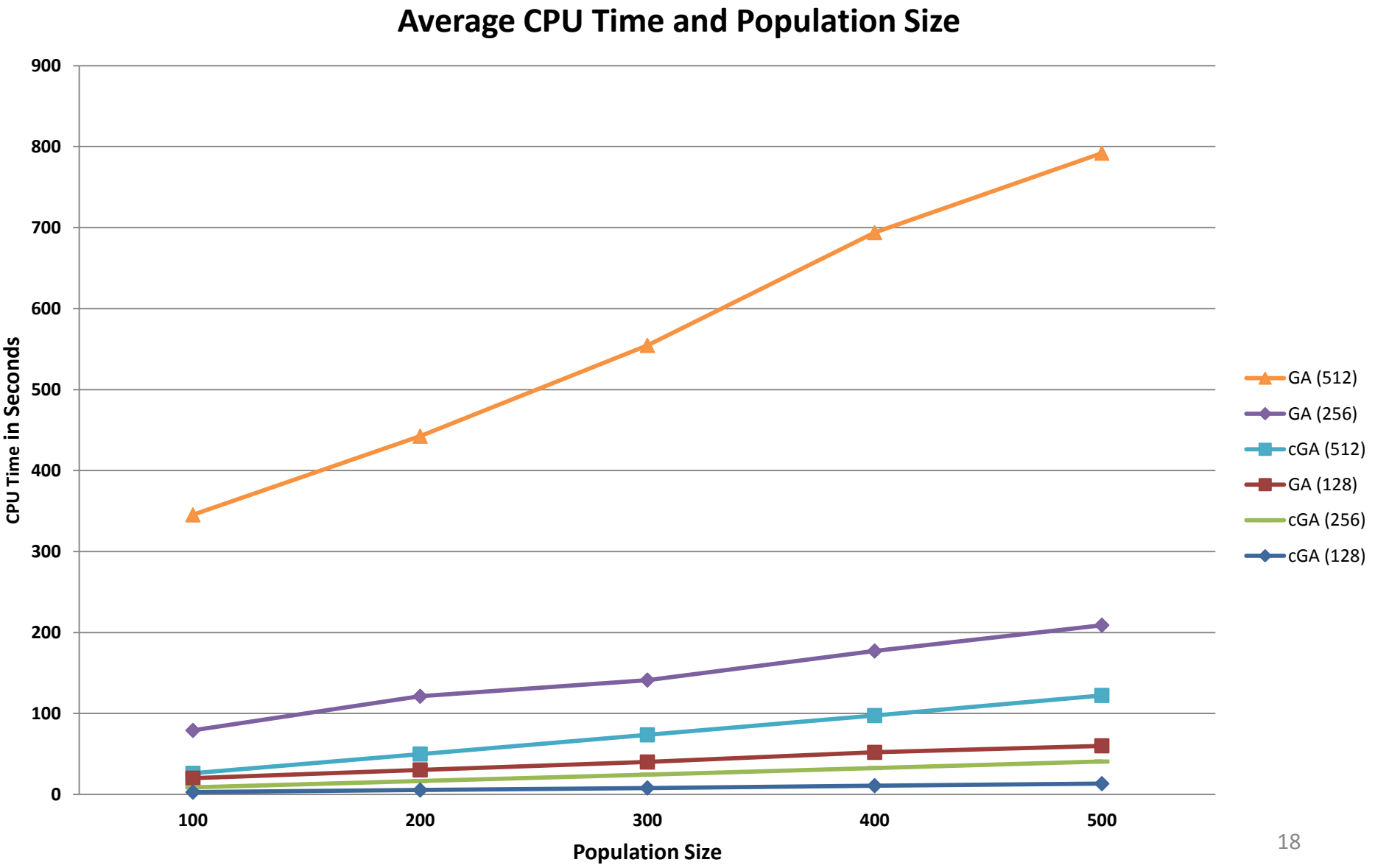
M3. Memory Usage



M4. CPU Time


Average CPU Time and Population Size
CPU Time in Seconds
18
16
14
12
10
8
6
4
2
0
100
200
300
400
500
Population Size
GA (64)
GA (32)
cGA (64)
cGA (32)
GA (16)
cGA (16)

M4. CPU Time


Average CPU Time and Population Size
CPU Time in Seconds
900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
Population Size
GA (512)
GA (256)
cGA (512)
GA (128)
cGA (256)
cGA (128)

cGA is cool, but…

The Dark Side of cGA

cGA (not) Convergence

- Convergence failed (max iterations):
  - Experiment #1: 5/300 (512-100)
  - Experiment #2: 4/300 (512-100,512-200)
  - Experiment #3: 5/400 (256-100,512-100)

- 1.4% of the runs do not converged.

Conclusions

- cGA and GA were evaluated considering the following metrics:
  - M1. Number of Iterations / Generations
  - M2. Number of Comparisons
  - M3. Memory Usage
  - M4. CPU Time
- M1 is not valid, considering that Iterations in cGA are not Generations in GA.
- For M2, cGA performed lower number of individual comparisons than GA (in all experiments).
- For M3, cGA performed lower memory usage than GA (in all experiments).
- For M4, cGA performed lower CPU time than GA (in all experiments).
- cGA convergence failed detected with 1M iterations.

Future Work

- Analyze:
  - number of individual comparisons with different population size.
  - when to stop increasing population size?
  - other classical problems for generalizing the conclusions of this work.
  - are these results valid for other problems?
  - correlation between variables of the problem
  - will variable 1 increase if variable 2 increases?

An Experimental Evaluation for OneMax Problem

cGA versus GA

Fabio López Pires