CS550 Distributed Systems and Cloud Computing

Parallel Genetic Algorithm Using PySpark for Knapsack Problem

Furkan Cantürk

15.01.2021

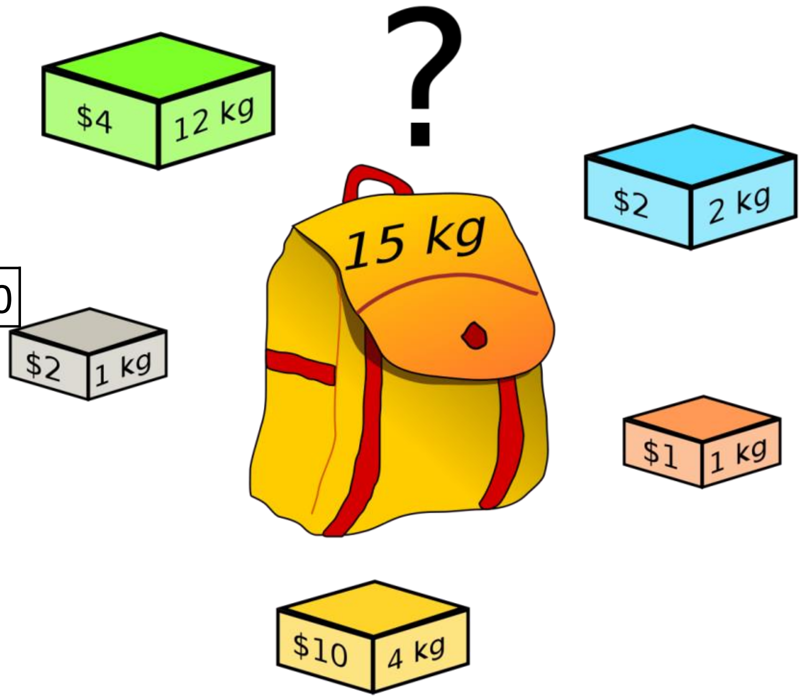
Knapsack Problem

Item, *i*	0	1	2	3	4	5	6	7
Weight (kg), w_i	2	2	3	3	4	4	5	5
Profit ($), p_i	9	10	12	20	15	18	13	14

Solution Representation: A binary string
Value at index i is 1 if item i is in knapscak, otherwise 0

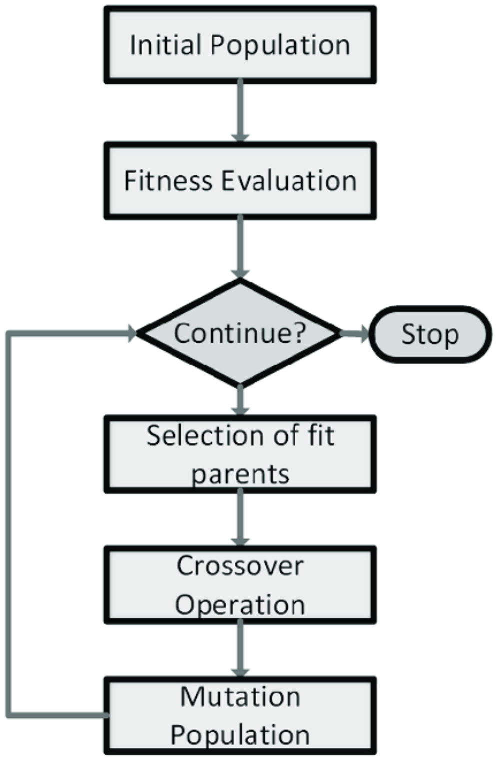
1	1	1	1	1	0	0	0

Total Weight: 14
Total Profit: 66



Genetic Algorithm (GA)

GA Flow



GA Notions

Individual: Solution

Chromosome: Solution representation

Gene: Smallest solution piece

Fitness function: Solution quality

Population-Generation: A set of solutions

Parents: Two solutions to produce new solutions (offspring)

Next Generation: New population generated by current population

Crossover: Exchanging genes between 2+ chromosomes

Mutation: Changing a gene / multiple genes in a chromosome

A Parallel Genetic Algorithm (PGA)

1. Split population into n subpopulations (partitions)
2. For each population apply a same or different GA
3. Fittest individual(s) of each subpopulation is cloned (migrated) to other subpopulations at a some frequency
    - This operation requires synchronization at a given frequency between RDD partitions!!!

Single Point Crossover

Crossover Point

PARENTS

Parent 1	1	1	1	1	0	0	0	1
Parent 2	1	1	1	1	0	1	0	0

OFFSPRING

Child 1	1	1	1	1	0	1	0	0
Child 2	1	1	1	1	0	0	0	1

I designed Sequantial-GA (SGA) which chooses the crossover point randomly.

I designed Parallel-GA(PGA) which have 4 crossover operators where each one chooses a crossover index randomly in 1st, 2nd, 3rd, and 4rd fold of chromosome respectively.

Tests

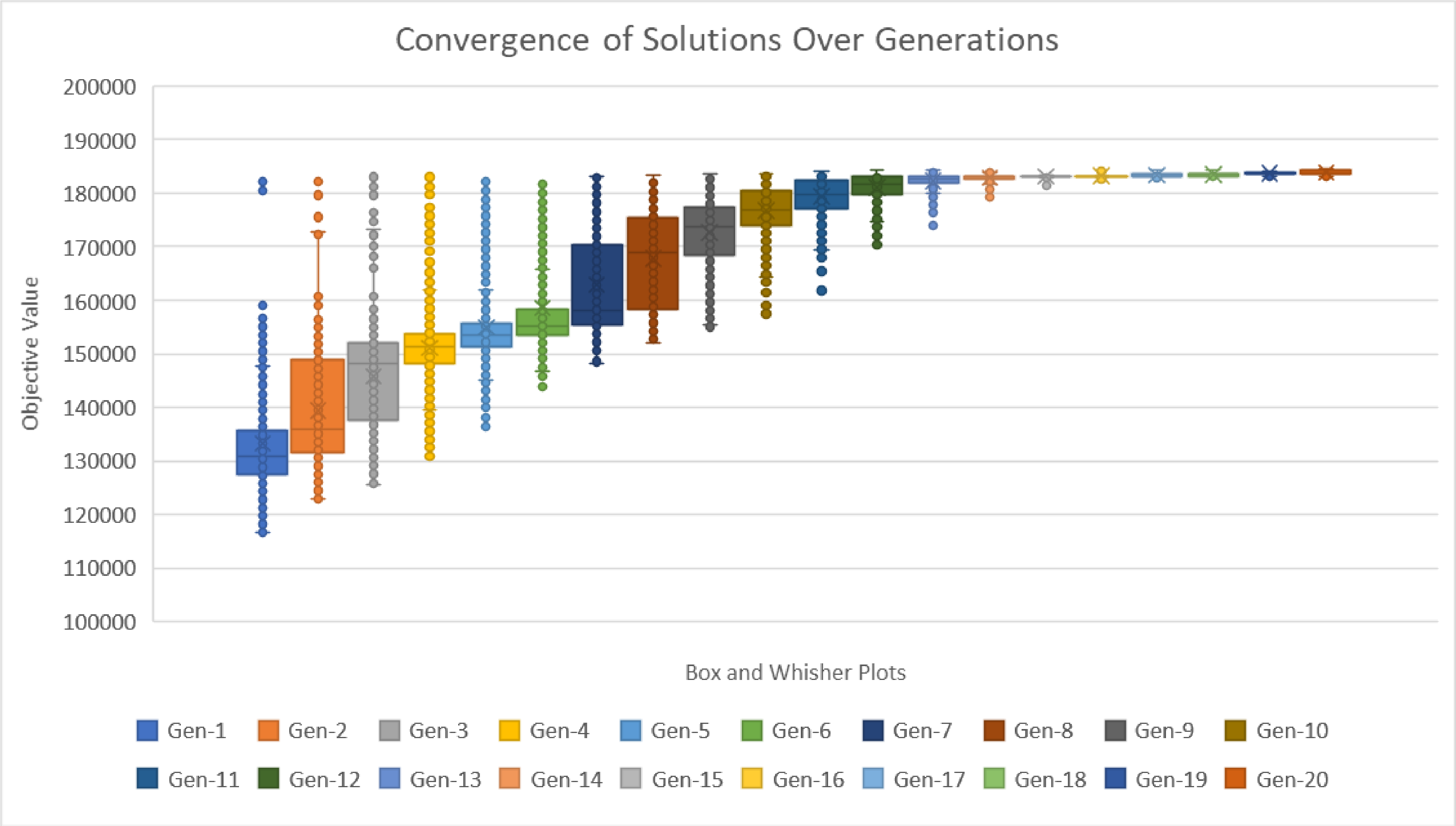
The following test results is valid for the following system requirements:

Intel İ7-1065G7 @1.30 GHz up to 3.90 GHz. (4 Cores – 8 Threads)

8 GB RAM

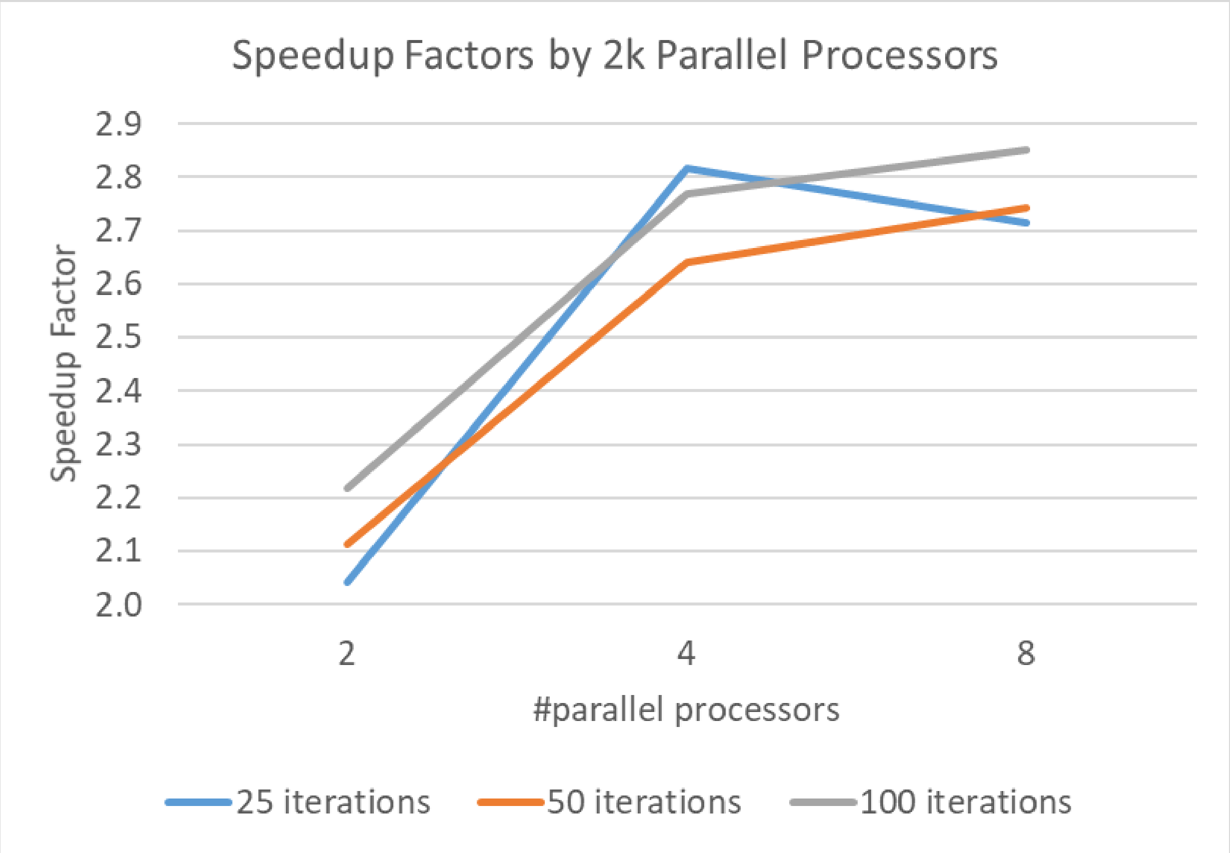
Python 3.8

Pyspark 3.0.1



Solution quality of PGA converges in 16-20 generations for a given 10000-items Knapsack instance.

Speedup Using 2k Parallel Processors



PGA with #generations

	#Parallel Processors		
#Iterations	**2**	**4**	**8**
25	2.0	2.8	2.7
50	2.1	2.6	2.7
100	2.2	2.8	2.9
Avg.	2.12	2.74	2.77

Speedup Factors

Conclusion:
There is no significant difference using 4 and 8 parallel processors for a 4 Core – 8 Thread CPU.

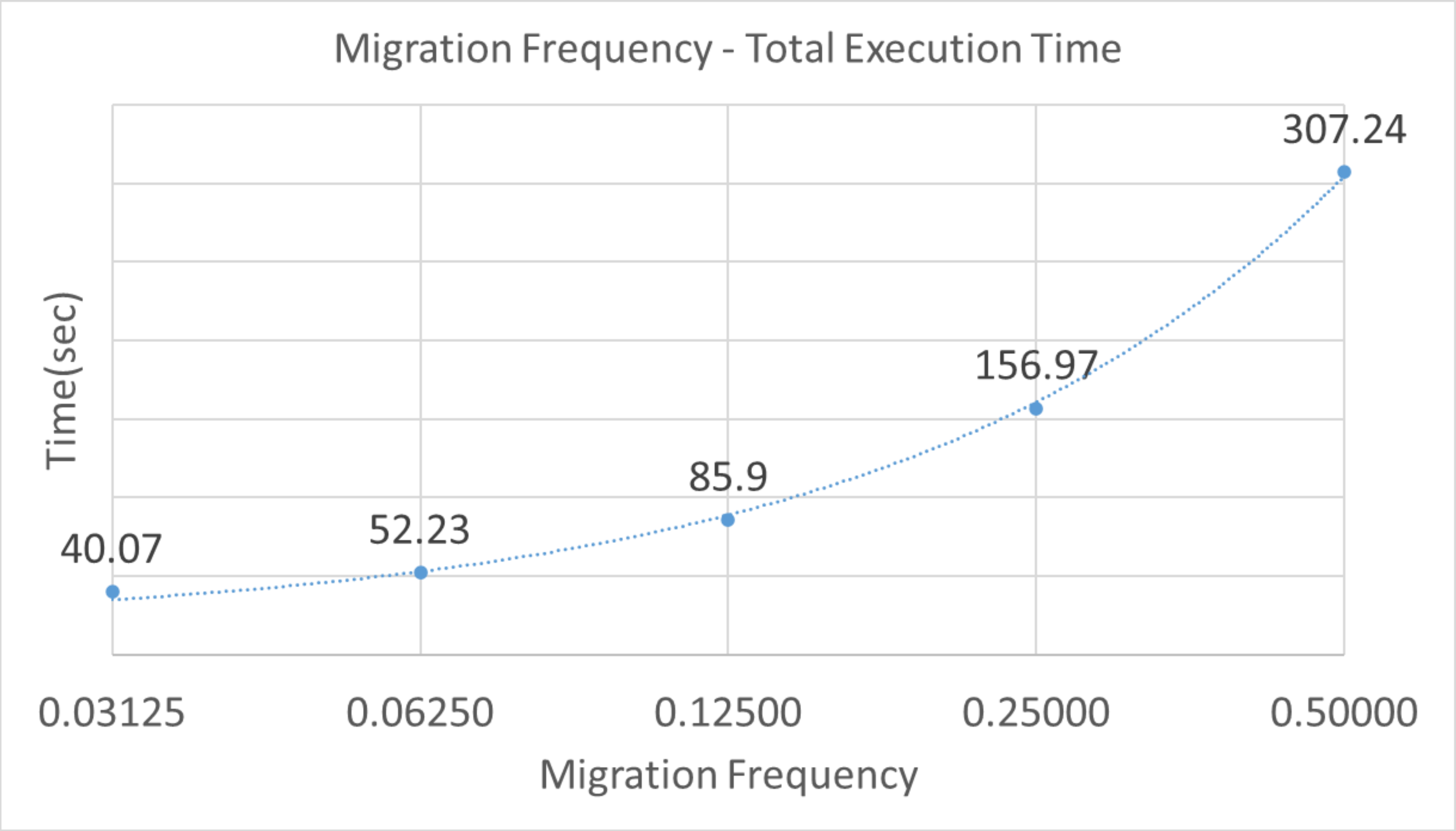
Performance of 2k RDD Partitions

PGA with 25 generations (iterations)

#partitions	Run-1	Run-2	Run-3	Avg.
2	17	17	19	17.6
4	13	12	13	**12.6**
8	17	16	14	15.6
16	13	13	14	13.4
32	17	17	17	17.1

Conclusion:
Best performance is achieved by PGA using 4 partitions when Spark runs with 4 parallel processors

Concurrency – Solution Time Tradeoff

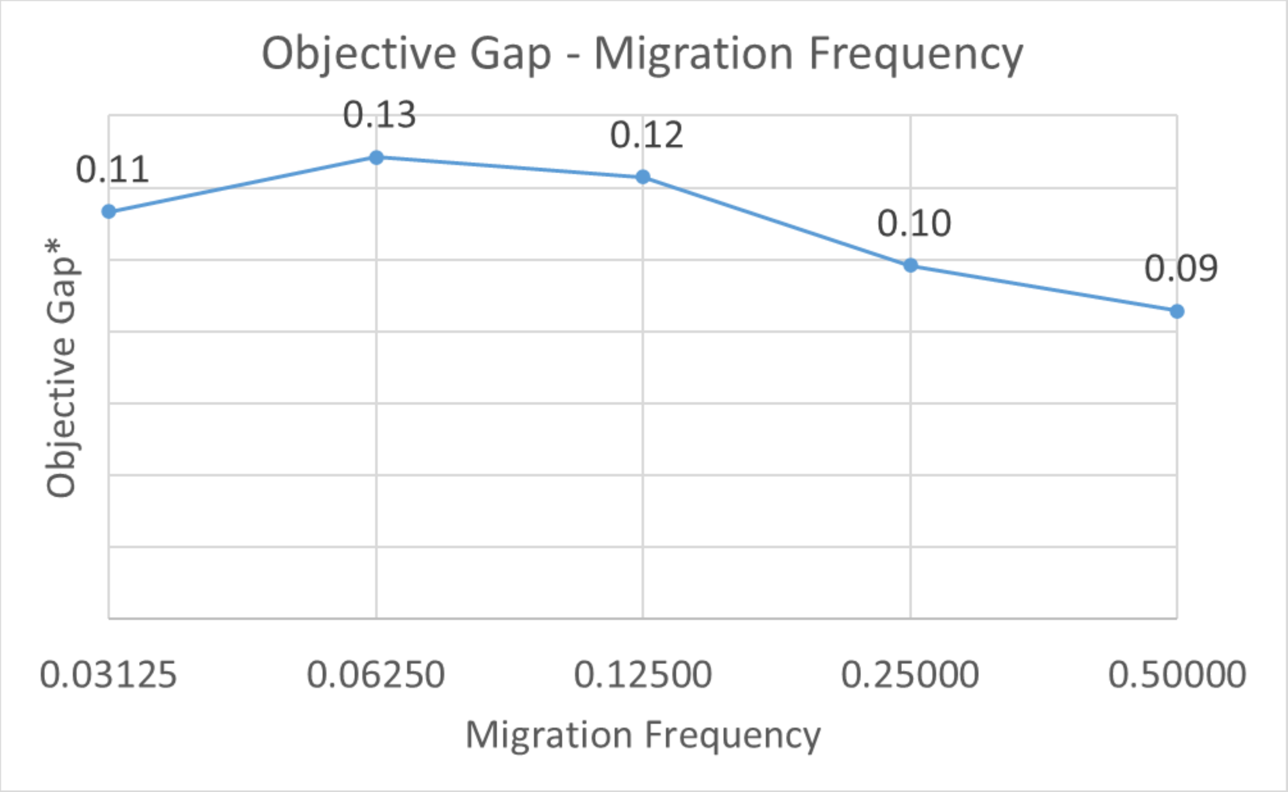


1000 items Knapscak Instance
Total population size = 10000
4 sub-populations
64 generations

Migration Frequency	Execution Time (sec)	Time Increase Rate
0.50000	307.24	1.96
0.25000	156.97	1.83
0.12500	85.9	1.64
0.06250	52.23	1.30
0.03125	40.07	

Conclusion: When migration frequency between parallel partitions is doubled, execution time of PGA increases at rising rates from 1.3 to 1.96.

Concurrency – Solution Quality Tradeoff



1000 items Knapscak Instance
Total population size = 10000
4 sub-populations
64 generations

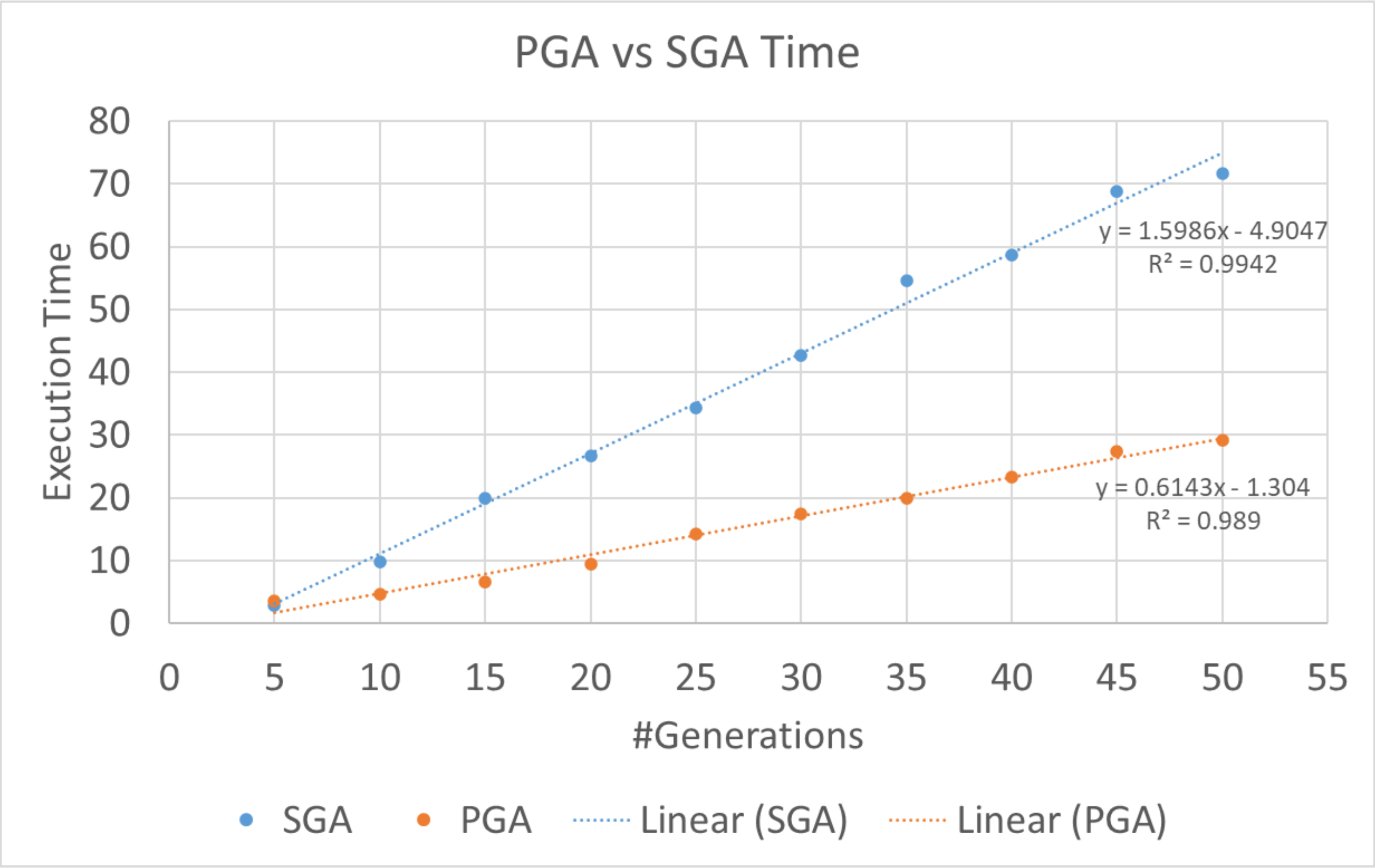
n_generations	migration_freq	time	mean_profit	max_profit	Objective Gap*
64	0.5	307.24	194112	194112	0.0857
64	0.25	156.97	191451	191451	0.0983
64	0.125	85.9	186227	186227	0.1229
64	0.0625	52.23	185028	185028	0.1285
64	0.03125	40.07	186481.7	188272	0.1132

*As I do not have optimal solutions for randomly generated Knapscak instances, I compared PGA solution with the solution provided by a Greedy Heuristic for Knapsack, which has objective value of 212316 for this instance.

Conclusion: More frequent migration between subpopulations improves solution quality of PGA.

Sequential vs Parallel GA

1000 items Knapscak Instance
Total population size = 10000
4 sub-populations
5 to 50 generations



Conclusion: PGA without migration achieves 2.67 times speedup compared to Sequential-GA.