

EA for constrained optimization

EOA - Homework 2.

David Koleckar

CTU, FEL, winter 21/22

Modules

main.py

Runs all the algorithms for all the objective functions, collects statistics and visualizations.

objective_functions.py

Implemented objective functions from [3], namely *g05*, *g06*, *g08*, *g11*, *g24*.

Provides an interface for working with listed functions.

```
class ObjectiveFunction:
    def __init__(self, id):
        self.name = id
        self.fitness_func = g<id>
        self.feasibility_func = g<id>_feasibility
        self.variables_size = g<id>_variables_size
        self.objectives_size = g<id>_objectives_size
        self.opt = g05_f_opt
```

Code snippet 1.: the objective_functions.py interface

for each of the objective functions there is in *objective_functions.py* a:

- function returning value for each of the objectives – fitness and constraints,
- function returning True, if the solution is feasible
- number of objectives and variables
- known optimum

Epsilon for numerical precision is set to $\epsilon = 0.00001$.

function_tools.py

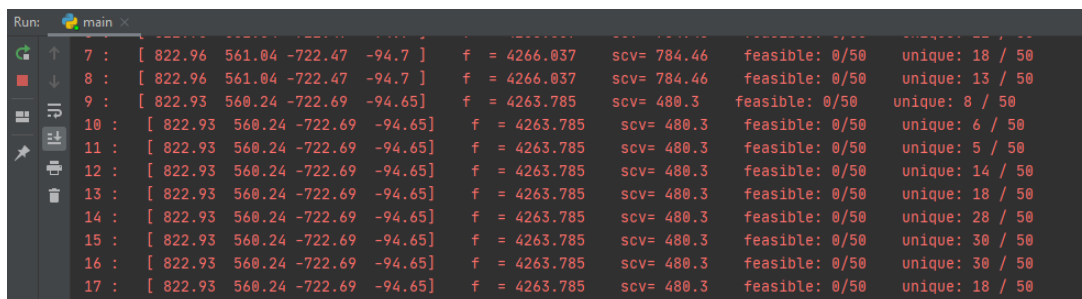
All helper functions and non-specific ea functions common to all the used ea pipelines (e.g. *init_population()*, *breed()*, ...).

operators_real_encoding.py

Collection of various crossover, mutation functions.

ea_nsga2.py, *ea_spea2.py*, *ea_stochastic_ranking.py*

Implementing the specific ea algorithms. Can be run as a script, if params[“verbose”] - printing some basic population statistics [Image 1].



7 :	[822.96 561.04 -722.47 -94.7]	f = 4266.037	scv= 784.46	feasible: 0/50	unique: 18 / 50
8 :	[822.96 561.04 -722.47 -94.7]	f = 4266.037	scv= 784.46	feasible: 0/50	unique: 13 / 50
9 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 8 / 50
10 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 6 / 50
11 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 5 / 50
12 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 14 / 50
13 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 18 / 50
14 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 28 / 50
15 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 30 / 50
16 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 30 / 50
17 :	[822.93 560.24 -722.69 -94.65]	f = 4263.785	scv= 480.3	feasible: 0/50	unique: 18 / 50

Image 1: Information(coordinates, fitness, scv, etc.) printed to console during chosen algorithm run.

Parameters

```
objective_function = ObjectiveFunction(objective_function_name)
params = {
    'fitness_func': objective_function.fitness_func,
    'feasibility_func': objective_function.feasibility_func,
    'objectives_size': 2, # objective_function.objectives_size
    'variables_size': objective_function.variables_size,
    'evaluation_mode': "single_objective", # "multi_objective"
    'solution_bounds': [-1000, 1000],
    'population_size': 50,
    'offspring_size': 150,
    'parents_size': 40,
    'generations': 200,
    'initialize_solution_func': init_population,
    'cmp_mode': "constraint_domination", # constraint_domination / "binary_tournament"
    'replacement_strategy': "generational",
    'tournament_size': 15,
    'crossover': xover_arithmetic,
    'mutation': mutation_cauchy,
    'mutation_prob': 0.35,
    'verbose': True,
}
```

Code snippet 2.: parameters for the ea.

These parameters were fixed and used for all of the test runs, except *g05*, where *generations*=2000. *objectives_size* is set to =2, it is possible to set it to =*objective_function.objectives_size*, to consider other objectives and not only the *sum of constraints violations*. If the *objectives_size* considers all the objectives, the *evaluation_mode* should be set to “*multi_objective*”.

Results

The following plots of the algorithms run were generated with [Code snippet 2] parameters. For information the full number of objectives is noted, but the algorithms were considering only 2 – fitness and sum of all constraints violations (the other “objectives”).

Note: the plots for Spea2 are not present, since the implementation of algorithm doesn’t converge (due to possible bug). Also the plots for multi-objective version of Nsga2 are not shown, because the algorithm didn’t converge neither, getting stuck in one solution (*unique_specimen*=1 from the whole *population_size*), while still not being feasible.

On the left: the plots for best fitness in current population (archive – for spea2) are given. Best means smallest fitness value, all functions are minimization problems. If non of the current solutions is feasible, the fitness value of solution with the smallest *sum of constraint violation* is given. Only values $\pm \frac{1}{4}$ from the optimum are plotted.

On the right: unique and feasible solutions in current population.

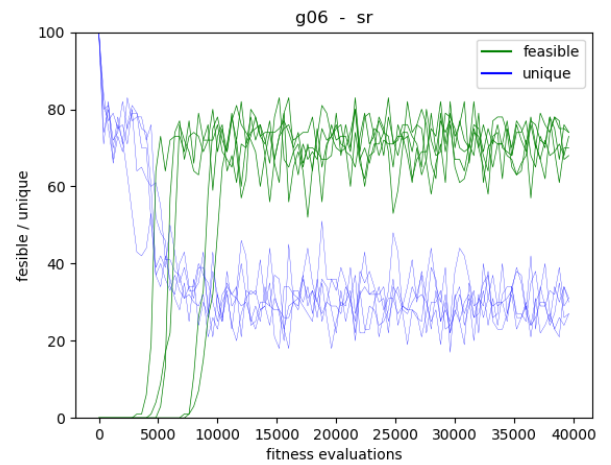
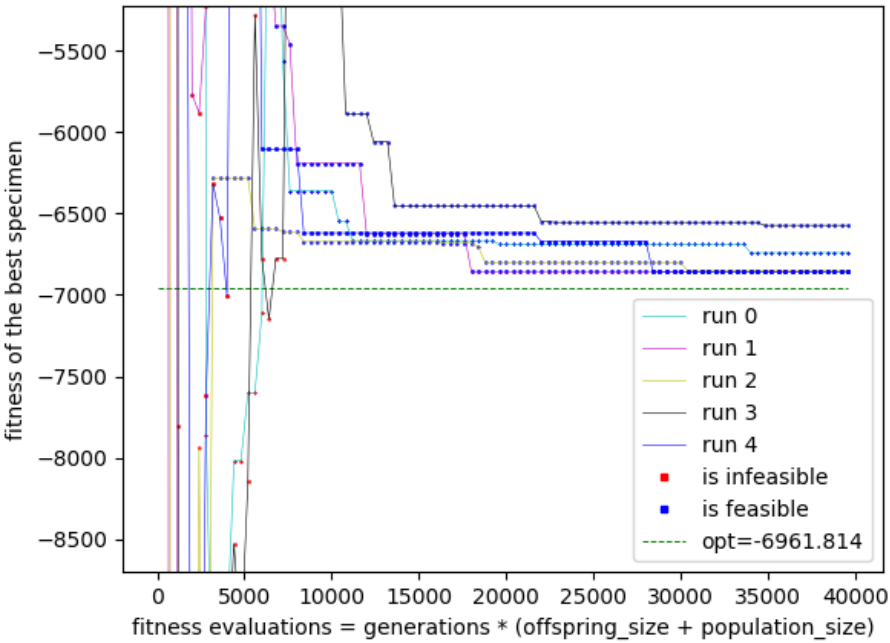
Note: y-axis should be better called “specimen in population”.

The optimum gap is calculated as an average of the fitness of the best specimen from the last population across the multiple runs.

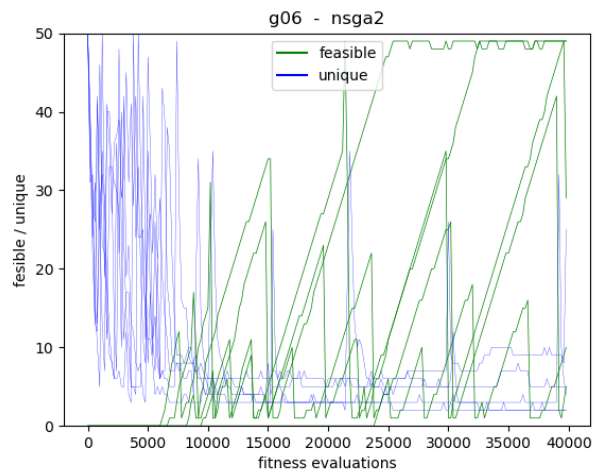
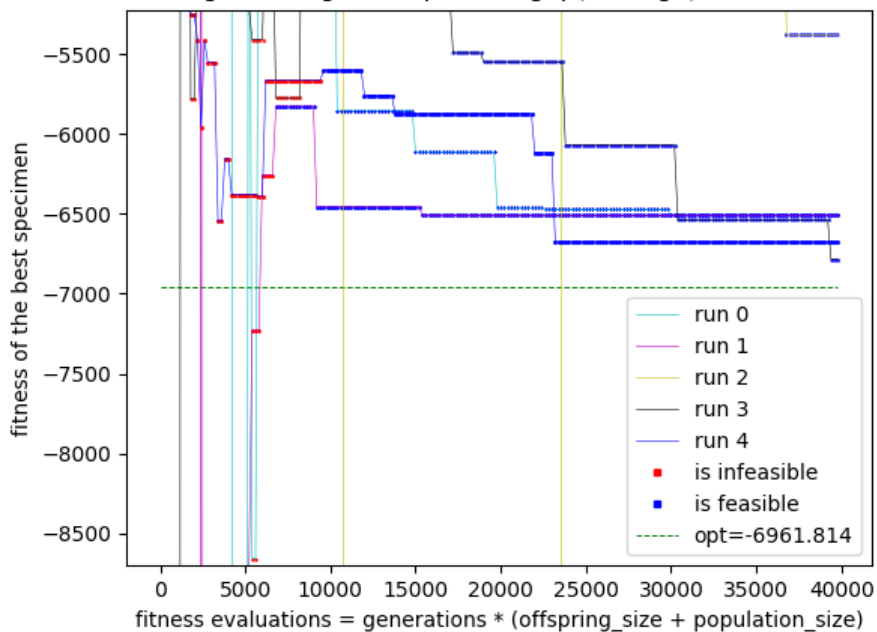
g06

variables = 2, # objectives = 7, (function definition given in [3])

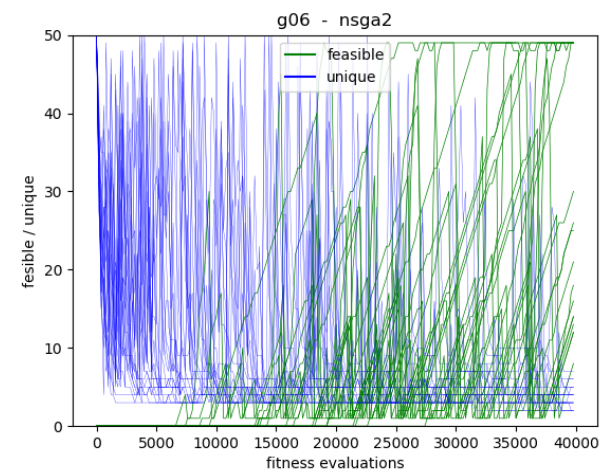
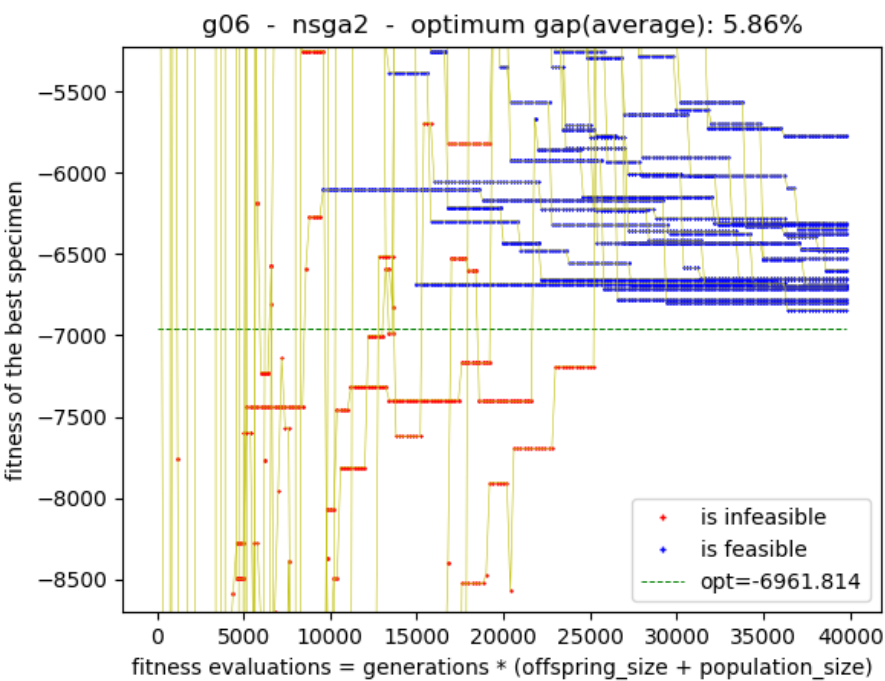
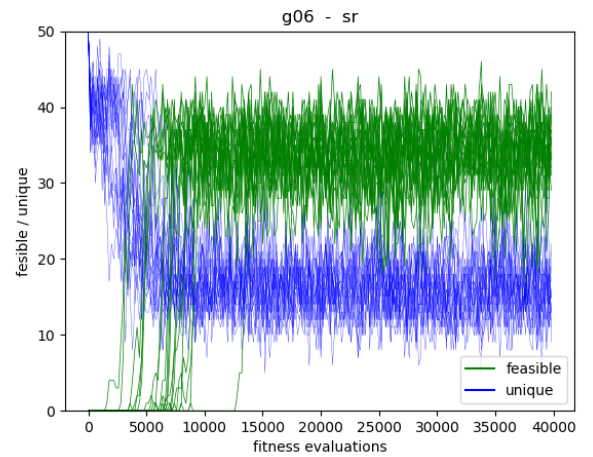
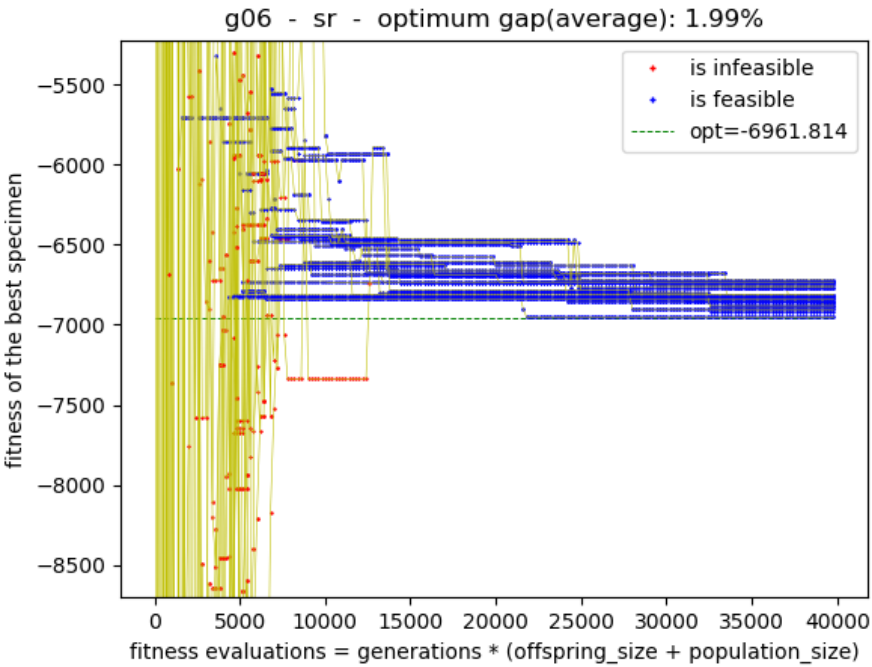
g06 - sr - optimum gap(average): 2.67%



g06 - nsga2 - optimum gap(average): 8.48%



g06 (20 runs)



g08

variables = 2
objectives = 7

min

$$f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

st:

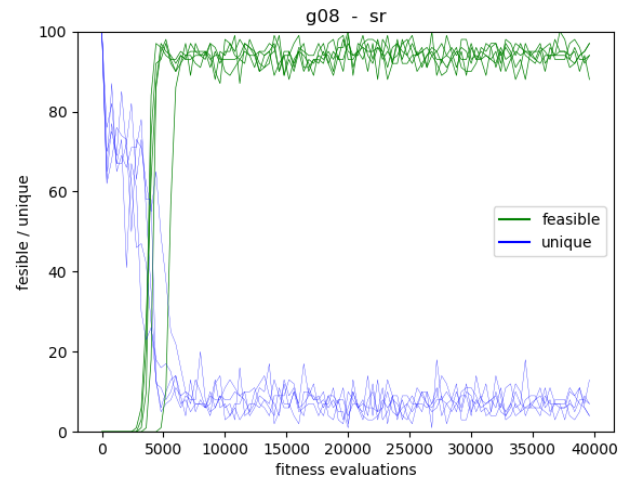
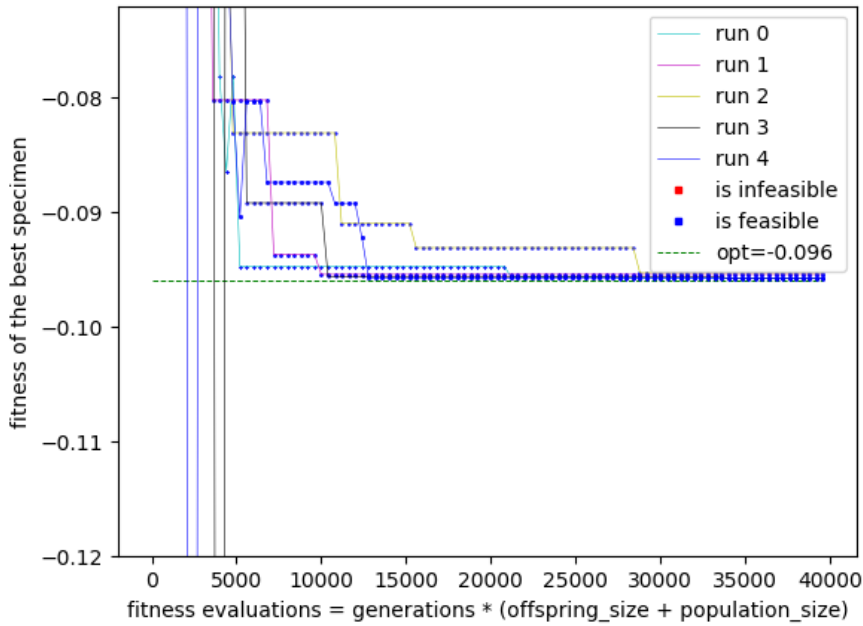
$$g_1(\mathbf{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$

$$g_2(\mathbf{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

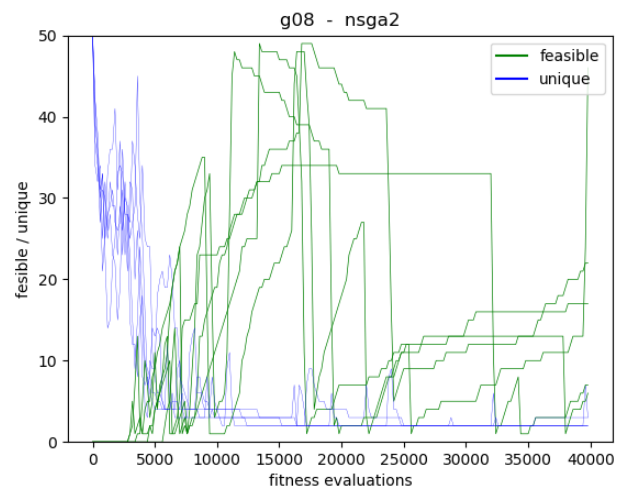
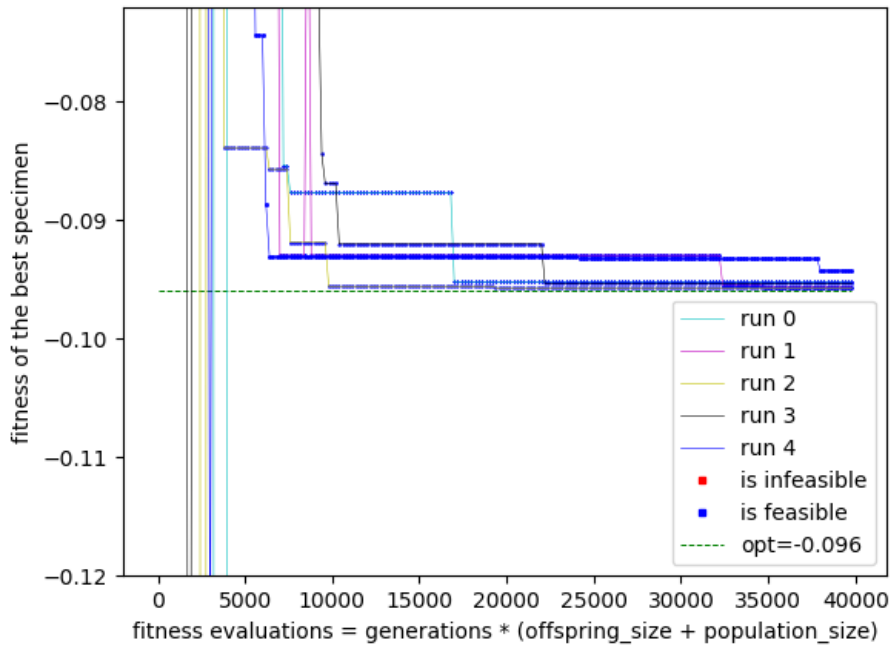
$$13 \leq x_1 \leq 100$$

$$0 \leq x_2 \leq 100$$

g08 - sr - optimum gap(average): 0.42%



g08 - nsga2 - optimum gap(average): 0.81%

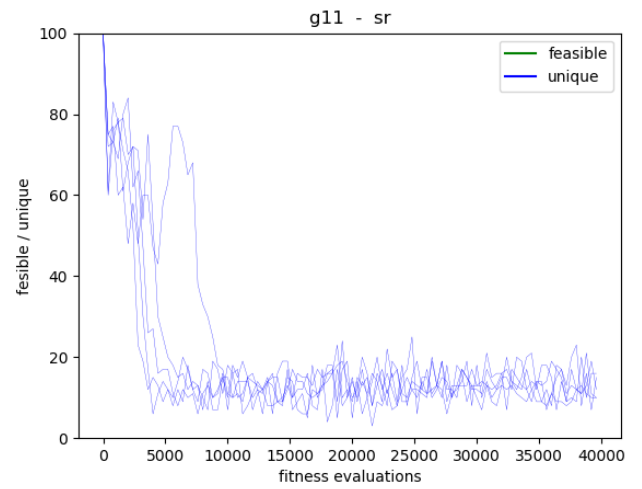
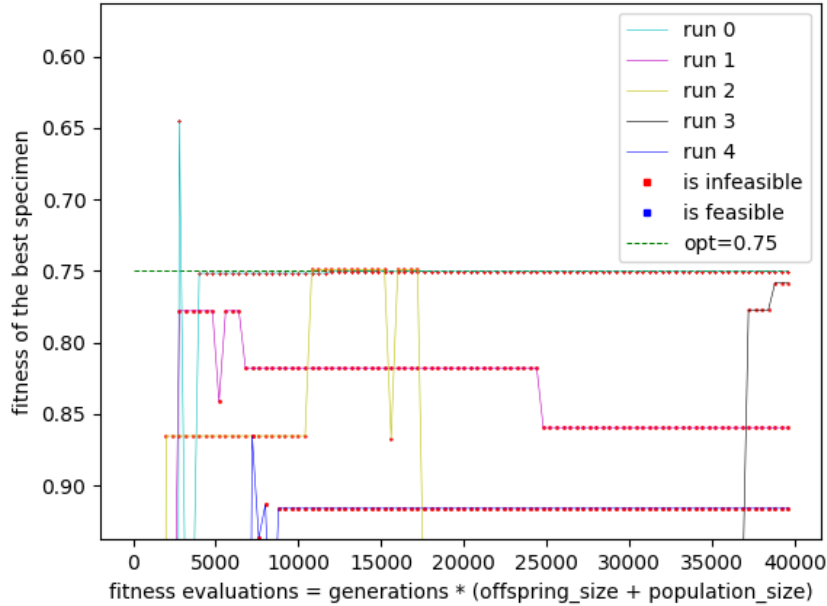


g11

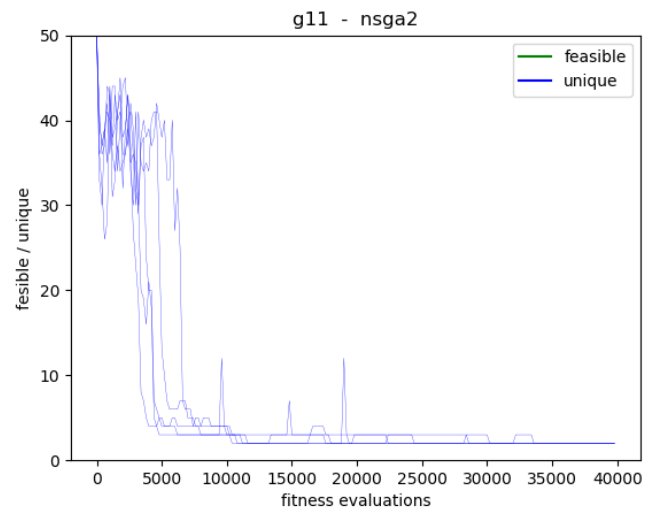
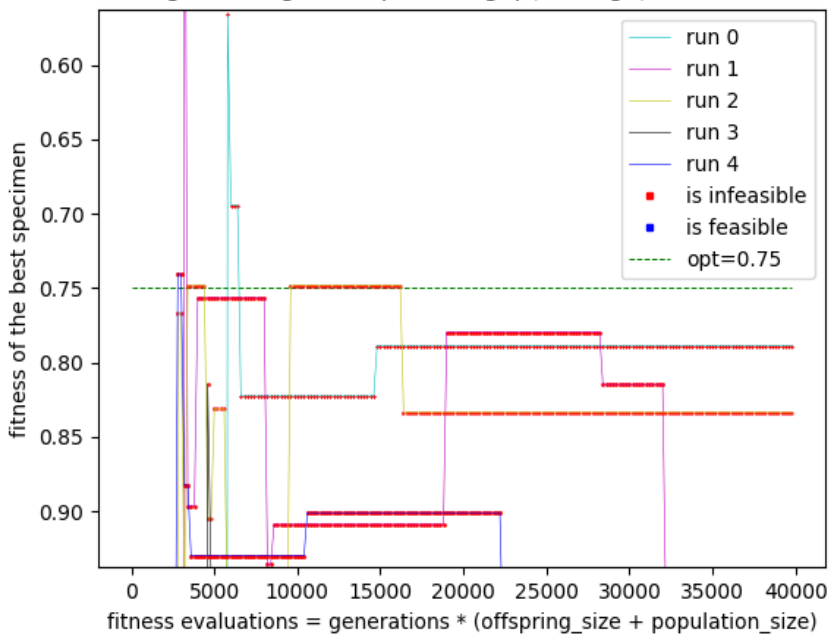
variables = 2
objectives = 6

$$\begin{aligned} \min \quad & f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \\ \text{st:} \quad & h(\mathbf{x}) = x_2 - x_1^2 = 0 \\ & -1 \leq x_1 \leq 1 \\ & -1 \leq x_2 \leq 1 \end{aligned}$$

g11 - sr - optimum gap(average): 14.24%



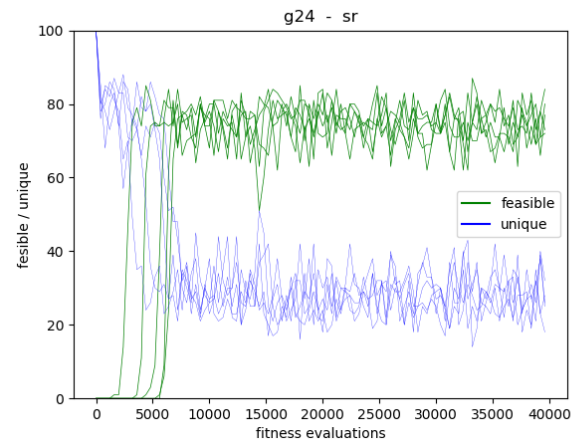
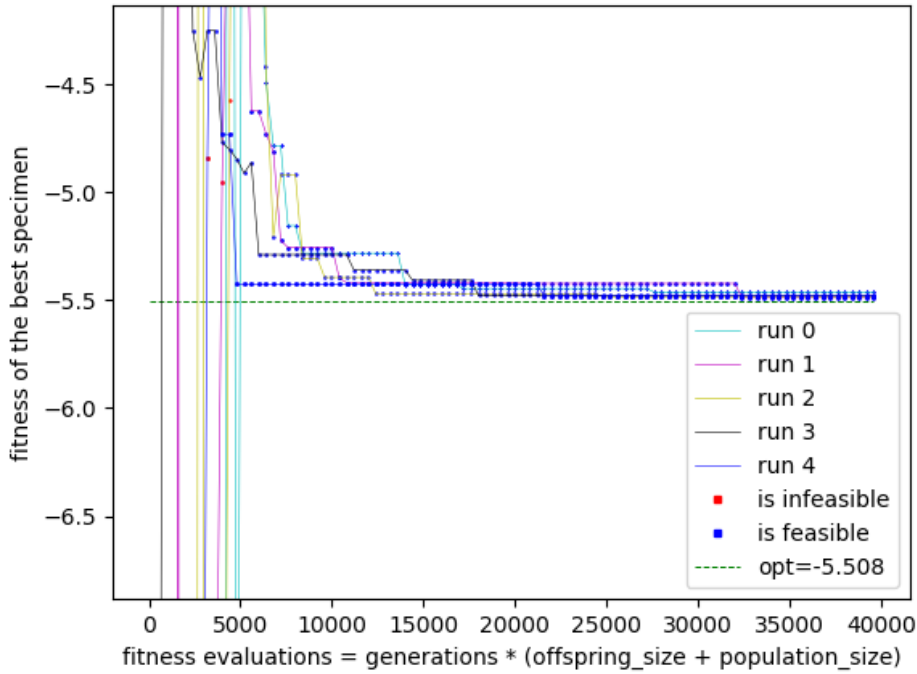
g11 - nsga2 - optimum gap(average): 20.86%



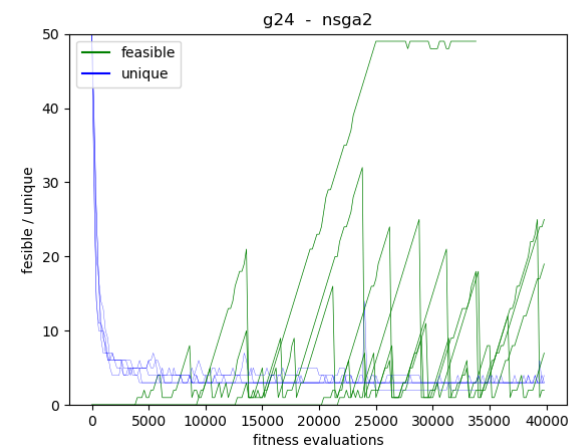
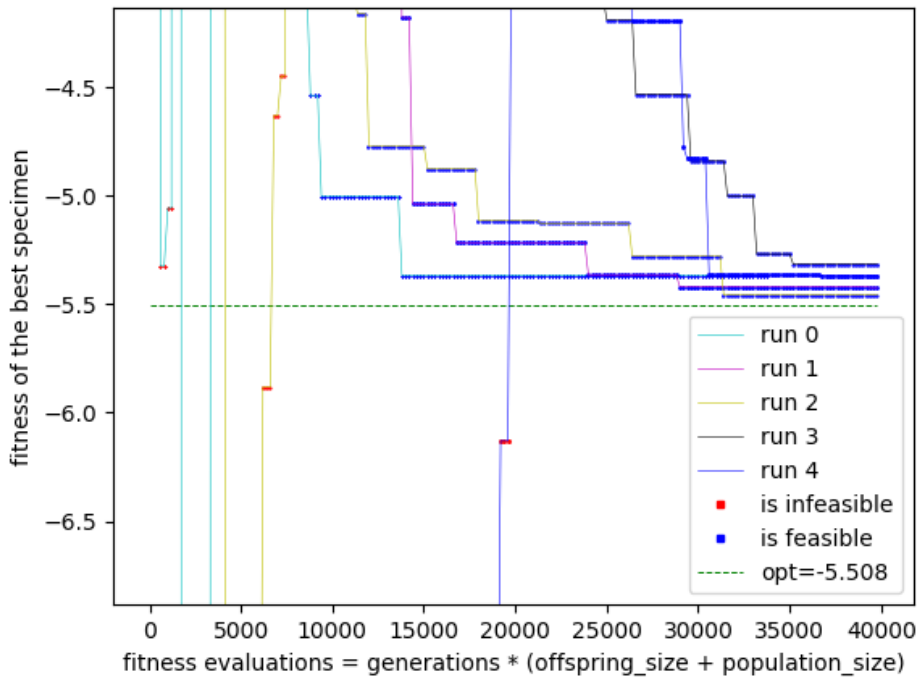
g24

variables = 2, # objectives = 7, (function definition given in [3])

g24 - sr - optimum gap(average): 0.52%



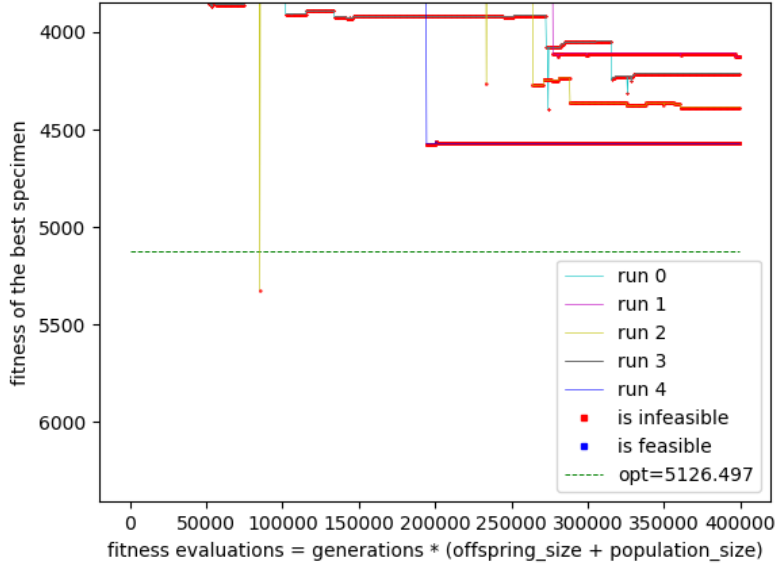
g24 - nsga2 - optimum gap(average): 2.21%



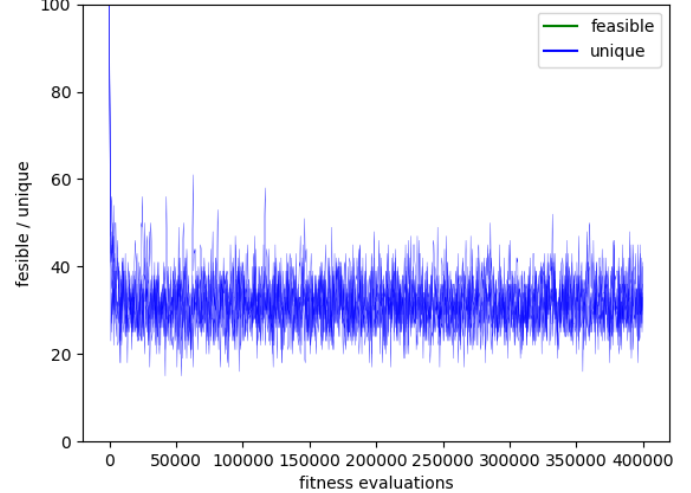
g05

Variables = 4, # Objectives = 14, (function definition given in [3])

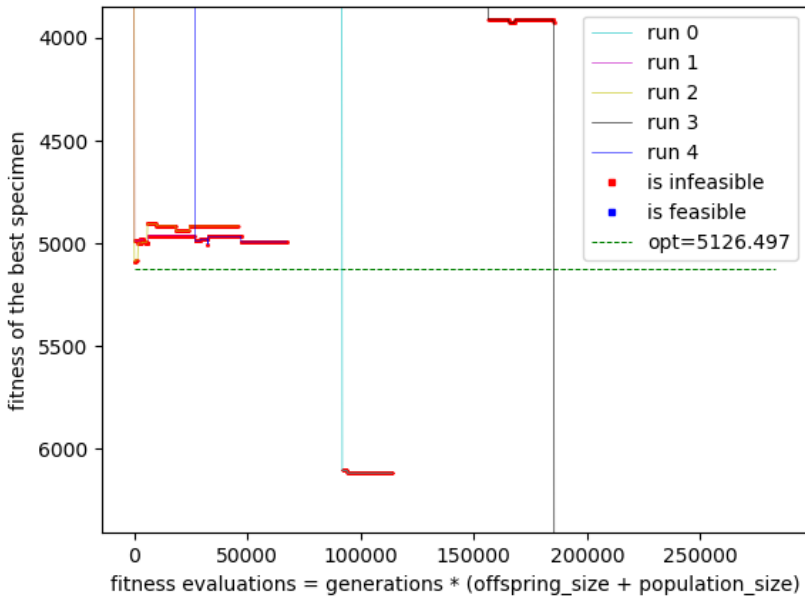
g05 - sr - optimum gap(average): 18.13%



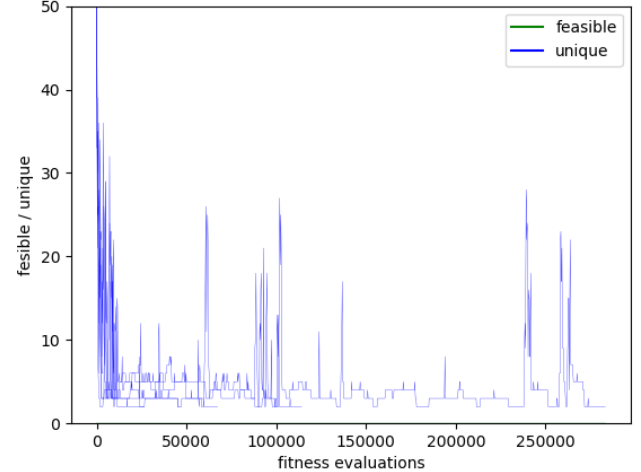
g05 - sr



g05 - nsga2 - optimum gap(average): 8.58%



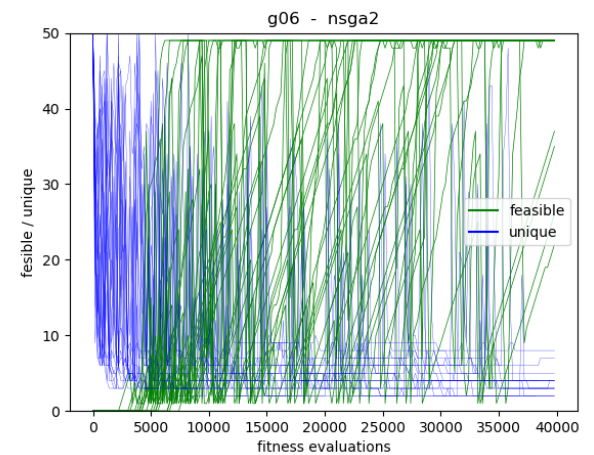
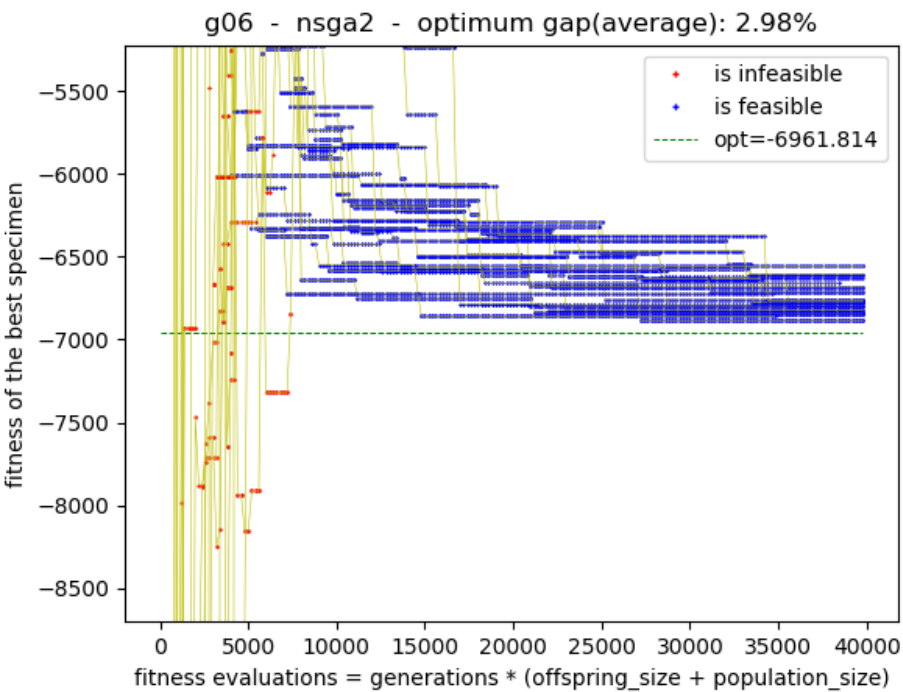
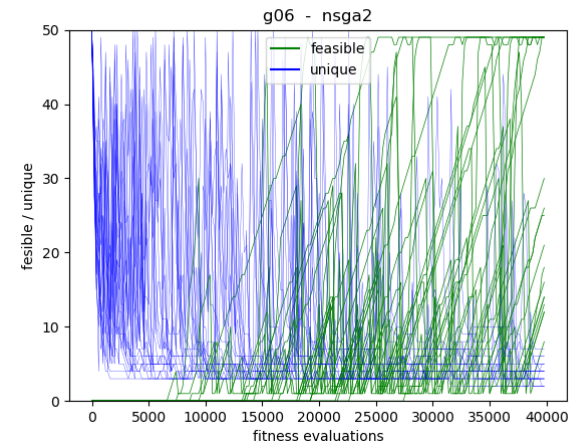
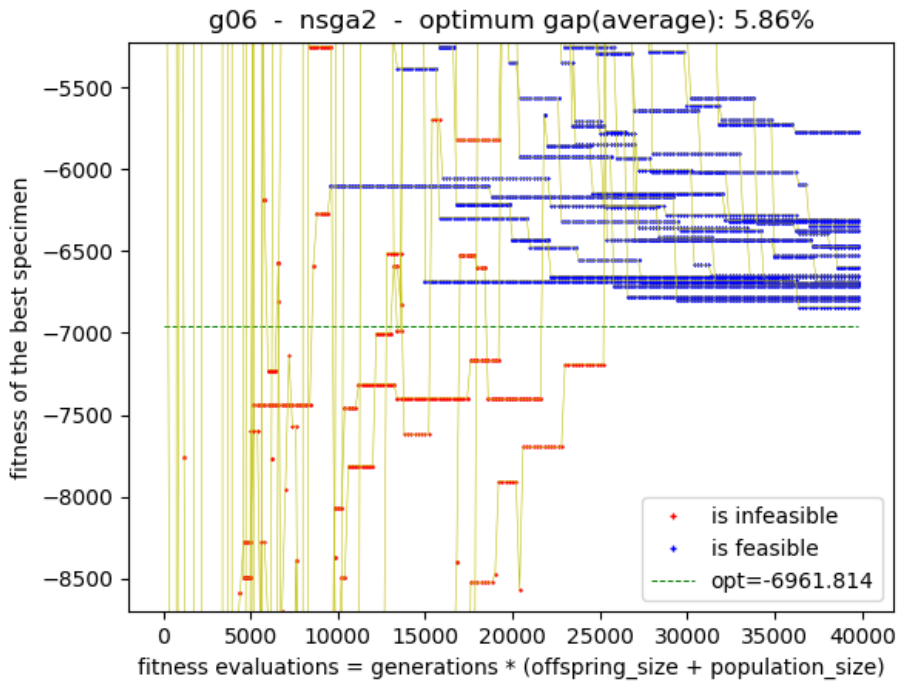
g05 - nsga2



The Nsga2 was in every run killed – due to the policy of 100 non-improving generations. The stochastic ranking was still improving (even if it is not clear from the plot resolution).

NSGA2: binary tournament vs. constraint domination - 20 runs

(constraint domination first, binary tournament as second plot)



Discussion

Stochastic ranking (sr), *NSGA2* and *SPEA2* algorithms were implemented. The results show for given implementation the stochastic ranking ea converges faster towards the global optimum. The *NSGA2* search seems to be more fluctuating.

The fitness was decreasing monotonically for both *sr* and *NSGA2* once the feasible solution was found. The *nsga2* tends to keep smaller portion of feasible solutions in population, contrary to the stochastic ranking. (Such phenomenon could be caused by given implementation?).

The optimality gap seems to decrease with increasing generations, but the ratio of optimality gaps for the *sr* and *nsga2* seems to remain more the same.

For *g11* both algorithms in given generations limit didn't converged, but it is possible, they would given more generations, the population is "balancing" on the infeasible side of pareto-optimal front. It seems however, the same cannot be said about *g05*.

The operators for parent selection in *NSGA2*, namely the *binary tournament* for *nsga2* seems to be more effective on some functions than using *constraint domination* tournament .

Notes:

1)

Computing of the *crowding distance* is given by [2] as follows:

for specimen *j* the partial crowding distance for objective *i* is given as:

$$\text{partial_crowding_j}[i] = (\text{objective_j}[i + 1] - \text{objective_j}[i - 1]) / (\text{objective_i_max} - \text{objective_i_min})$$

The case for *objective_i_max* == *objective_i_min* however is not discussed in the [2]. This is for example the case when all every solution has violation equal to zero for the same objective.

2) the statistics could be re-run with '*cmp_mode*': '*binary_tournament*', which could improve the optimality gap and convergence for some of the test functions.

References

[1] Zitzler, Eckart; Laumanns, Marco; Thiele, Lothar, (2001), *SPEA2: Improving the strength pareto evolutionary algorithm*

[2] Deb, K., Pratab, A., Agarwal, S., and Meyarivan, T. (2002a). A fast and elitist multi-objective genetic algorithm: *NSGA-II*. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

[3] test functions defintions:

https://cw.fel.cvut.cz/wiki/_media/courses/a0m33eoa/cviceni/2006_problem_definitions_and_evaluation_criteria_for_the_cec_2006_special_session_on_constraint_real-parameter_optimization.pdf