Evolutionary Computation

Assignment 3: PWT Problem

Puzhi Yao (a1205593)
Zhuoying Li (a1675725)
Jingwen Wei (a1671836)
Xueyang Wang (a1690260)

June 4, 2017

1 Architecture and Changes

Our program is based on the JMetal4.5.2. The architecture of JMetal is shown in the Figure 1.

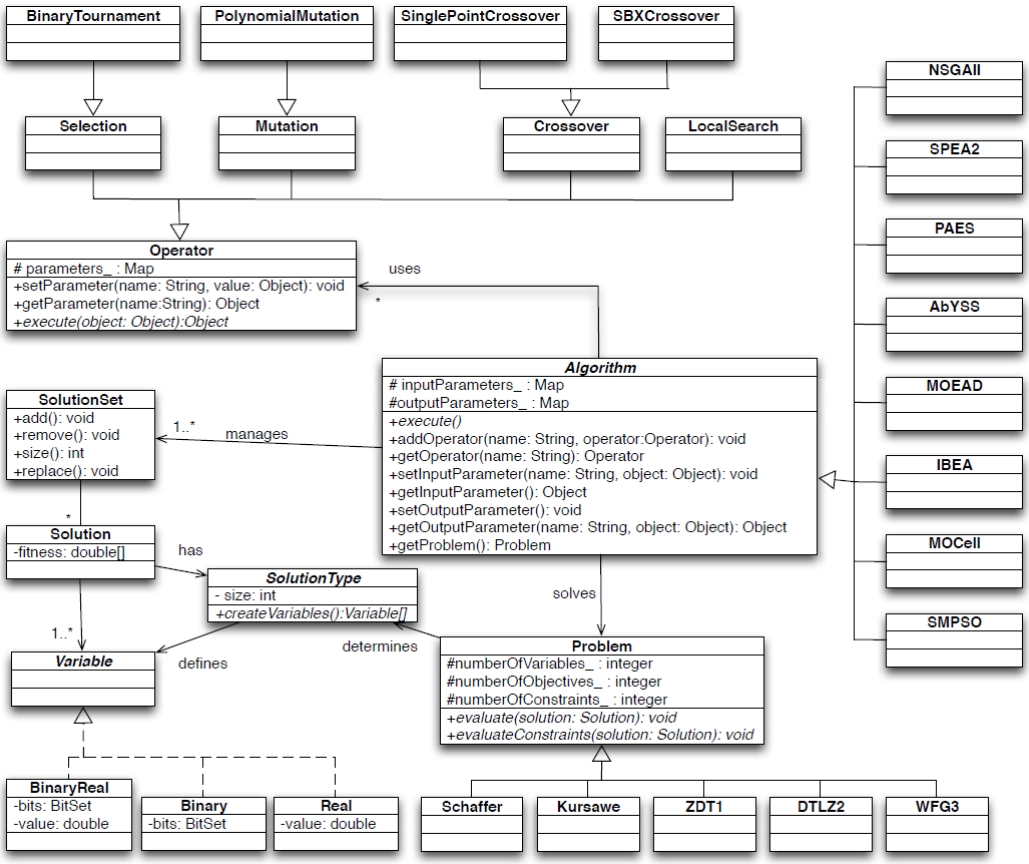

Figure 1: Architecture of JMetal

We haven't make any changes on the overall architecture but we have designed and implement some components in the architecture.

1.1 Solution type

The only variables that will applied to the algorithm is the packing plan, and the packing plan is only a series of zeros and ones, so we choose Binary Solution Type and Binary data type to store our solutions and packing plan.

1.2 Problem

We have implement a new problem extends the problem class called PWT. In JMetal 4.5.2, there is no problem type that can suit PWT problem, so we designed a new problem type that can keep the constant parameters of PWT problems such as the number of cities, the number of items and so on. Except the function that can evaluate a solution and evaluate a constraint, PWT problem also have the function that can read from the TTP input file and fixed tour file to initial the problem. In order to make the dynamic items available, we also design several function in PWT problem that can validate and update the packing plan of a solution. The architecture of the PWT problem is shown as Figure 2.

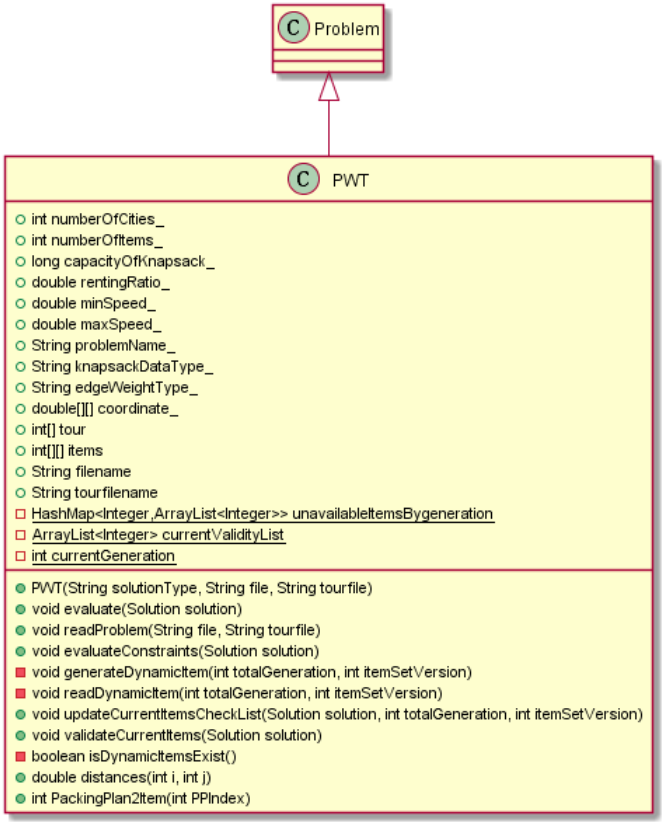

Figure 2: PWT Architecture

1.3 Comparator

We designed a new comparator: Priority Comparator which consider the constraint first then final profit and at last for the weight (Algorithm 1). The default comparator of JMetal is Dominance Comparator which consider all the objective value at the same level. Priority Comparator can make the selection

more efficiency.

Algorithm 1: Priority Comparator

Input: solution1, solution2

```
if solution1.profit > solution2.profit then
    return solution1 < solution2
end
if solution1.profit < solution2.profit then
    return solution1 > solution2
end
if solution1.weight > solution2.weight then
    return solution1 > solution2
end
if solution1.weight < solution2.weight then
    return solution1 < solution2
end
```

1.4 Util

We also designed some classes that can automatically running the program with a configuration file and generate the result in a csv format which can be analysis easily.

2 Operators for Exercise 2

We make experiment on different combination of crossovers and mutations including the operators that already in the JMetal library and designed by ourselves. The operators in JMetal which used in comparison are: HUXCrossover, SinglePointCrossover, BitFlipMutation. Group Point Crossover and One Bit Flip Mutation are designed for Exercise3 and will be introduced in Exercise3's document, but we also test them in Exercise 2. Here are the operators that we designed for Exercise 2:

2.1 Crossover

2.1.1 Arithmetic Crossover

Arithmetic crossover compare the binary packing plan in parents, only if at the same position both bits are true the offspring bit at this position are set to true (Algorithm 2).

Algorithm 2: Arithmetic Crossover

Input: parent1,parent2

Output: offspring

```
for i ← 0 to n do
    if parent1 and parent2 's packing plan at i are both true then
        offspring packing plan at i ← true;
    end
end
return offspring;
```

2.1.2 Reduce Surrogate Crossover

Reduce Surrogate Crossover are based on the Single Point Crossover, first the offspring copy the parent packing plan then record the index(and bits) which is false in at least one parent's packing plan. At last choose a random point and do swap the gene using the recording. This operator will perform well when

most bits in the packing plan are true (algorithm 3).

Algorithm 3: Reduce Surrogate Crossover

Input: parent1,parent2
Output: offspring
for $i \leftarrow 0$ ***to*** n **do**
  if *parent1 and parent2 's packing plan at i have at least one false* **then**
    Index ← i Bits ← bits of parent 2
  end
end
offspring ← parent1 **for** $i \leftarrow$ *random* ***to*** n **do**
  if *i in index* **then**
    offspring ← Bits[i]
  end
end
return *offspring*;

2.1.3 Uniform Crossover

Uniform Crossover first set a rate, then copy parent 1 to the offspring, each bit in the gene have the rate to change into the value at the same position of the gene of parent2 (Algorithm 4).

Algorithm 4: Uniform Crossover

Input: parent1,parent2,probability
Output: offspring
offspring ← parent1 **for** $i \leftarrow 0$ ***to*** n **do**
  if *random* $<$ *probability* **then**
    offspring packing plan at i ← parent2 packing plan at i;
  end
end
return *offspring*;

2.2 Mutation

2.2.1 Dynamic Bit Flip Mutation

Dynamic Bit Flip mutation is based on Bit Flip mutation but the probability of mutation are separate to two: probability from false to true and probability from true to false. Both two probability are set to the default at first but if the current packing plan is overweight, the mutation rate from false to true are set to zero. This mutation will make sure the mutation is good when packing plan is overweight

(Algorithm 5).

Algorithm 5: Dynamic Bit Flip Mutation

```
Input: offspring,probability
Output: offspring
probability01 ← probability probability10 ← probability if overweight then
    probability10 ← 0
end
for i ← 0 to n do
    if packing plan at i = true then
        if random < probability10 then
            Flip the bit at i
        end
    end
    else
        if random < probability01 then
            Flip the bit at i
        end
    end
end
return offspring;
```

2.2.2 Greedy Mutation

Greedy Mutation first calculate if each item are taken the final profit will rise or fall. If rise, it still has the probability to mutate from false to true if not is will set to false (Algorithm 6). (Penalty function can be found in the Assignment 2 in myOperator.java)

Algorithm 6: Greedy Mutation

```
Input: offspring,probability
Output: offspring
penalty ← penaltyfunction(packing plan)
for i ← 0 to n do
    if packing plan at i = true then
        if penalty is rise then
            if random < probability then
                flip the bit
            end
        end
        else
            bit ← false
        end
    end
    else
        if penalty is rise then
            if random < probability then
                flip the bit
            end
        end
    end
end
return offspring;
```

2.2.3 Interchange Mutation

Interchange mutation first random a value that determine the run times of mutation, then find two random position and swap the value (Algorithm 7).

Algorithm 7: Interchange Mutation

```
Input: offspring,probability
Output: offspring
while random < probability do
    i ← random position;
    j ← random position;
    swap the value of the packing plan of offspring at i, j;
end
return offspring;
```

2.2.4 Reversing Mutation

Reversing Mutation flip two bit before a random point and run random times base on the probability (Algorithm 8).

Algorithm 8: Reversing Mutation

```
Input: offspring,probability
Output: offspring
while random < probability do
    i ← random position;
    flip the packing plan of offspring at i+1;
    flip the packing plan of offspring at i-1;
end
return offspring;
```

3 Result

6 crossover and 6 mutation combination are test using three algorithm with 9 instances. Testing parameters used in the test:

- The number of generation are set to 1000, 10000, 50000 and 100000.
- Each combination are running for 10 times

Figure 3-11 are showing the best three average scaled error in 9 instances. Overall, IBEA algorithm outperform the other two algorithm. Figure 3-5 shows that the One Bit Flip Mutation and Single Point Crossover are outperform others when the instance is in a low size. Figure 4,6,9 shows that the Dynamic Bit Flip Mutation can be use regardless of the size of instance but still have an acceptable result. Figure4,6,7 shows that the HUXCrossover is better use in medium size instance. Figure 8, 10, 11 shows that the Arithmetic and Single Point Crossover are perform better in large size instance. Figure 9,10,11 shows that the Reversing Mutation perform better in large size instance. The details performance of combinations are shown in the appendix. In conclusion, Single Point Crossover and Dynamic Bit Flip Mutation are the best which can be use broadly regardless of the size of the instance and have a good perform.

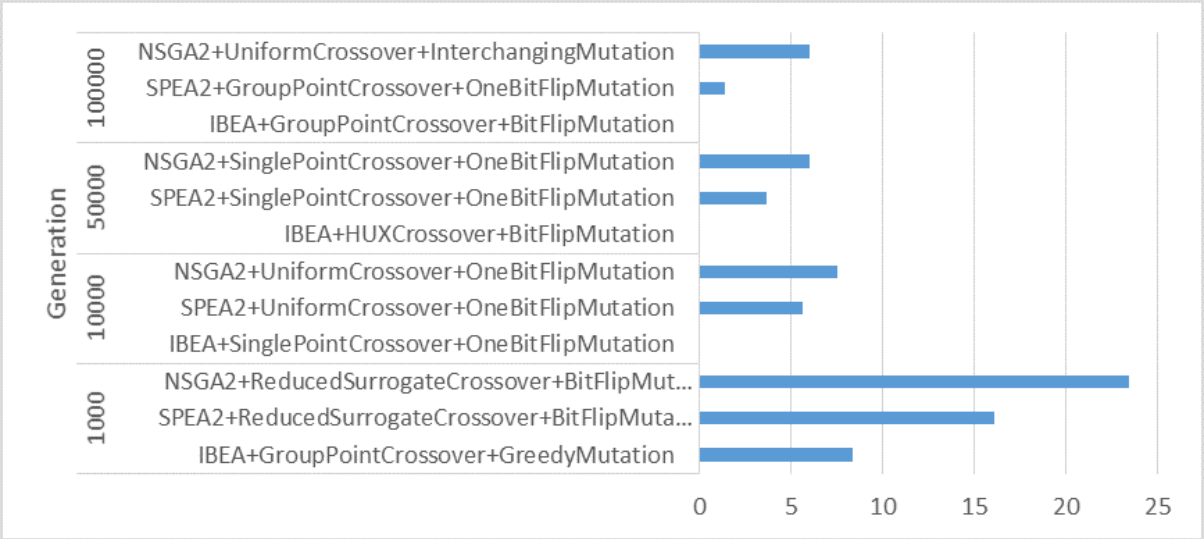

Figure 3: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_01 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

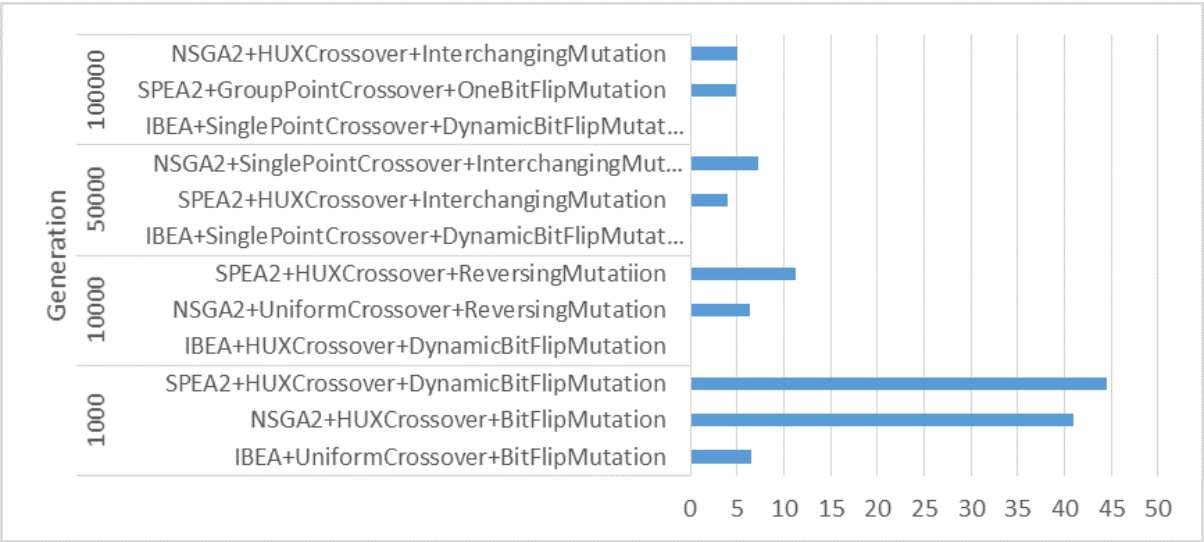

Figure 4: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_06 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

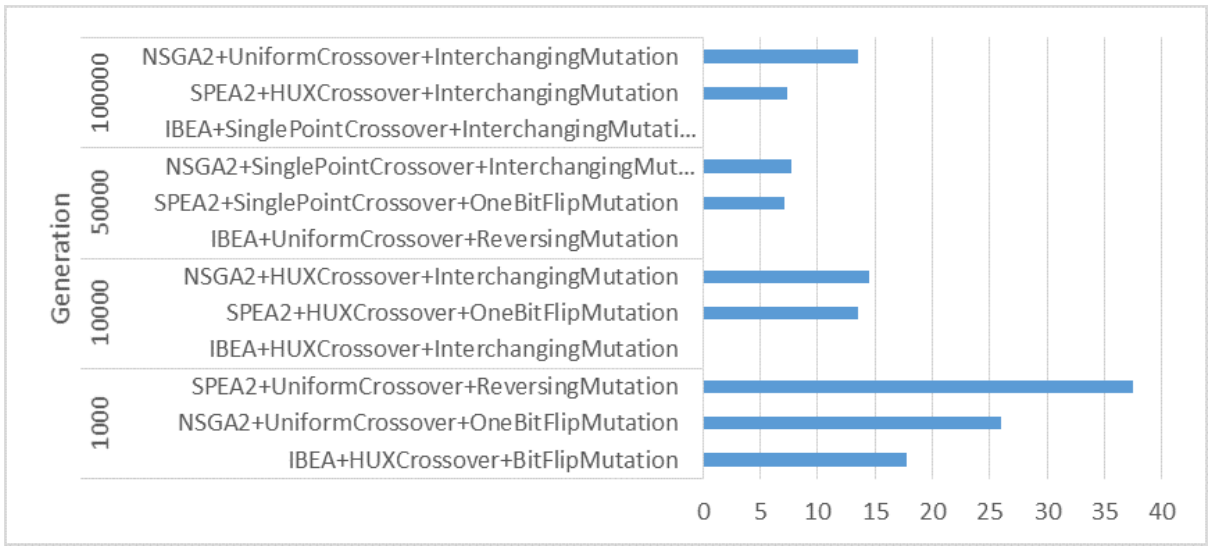

Figure 5: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_10 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

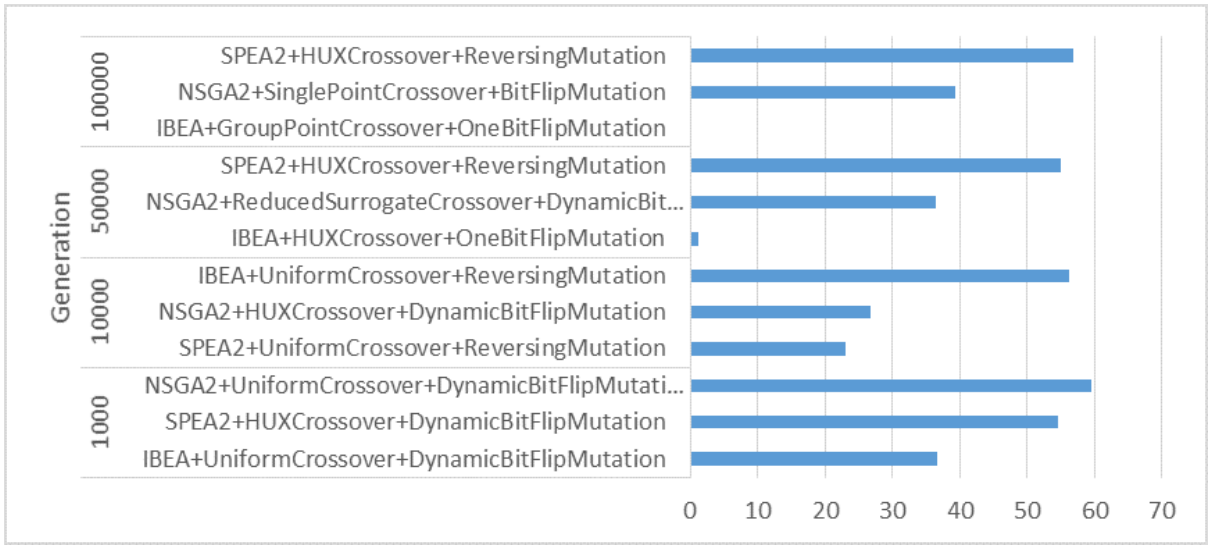

Figure 6: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_01 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

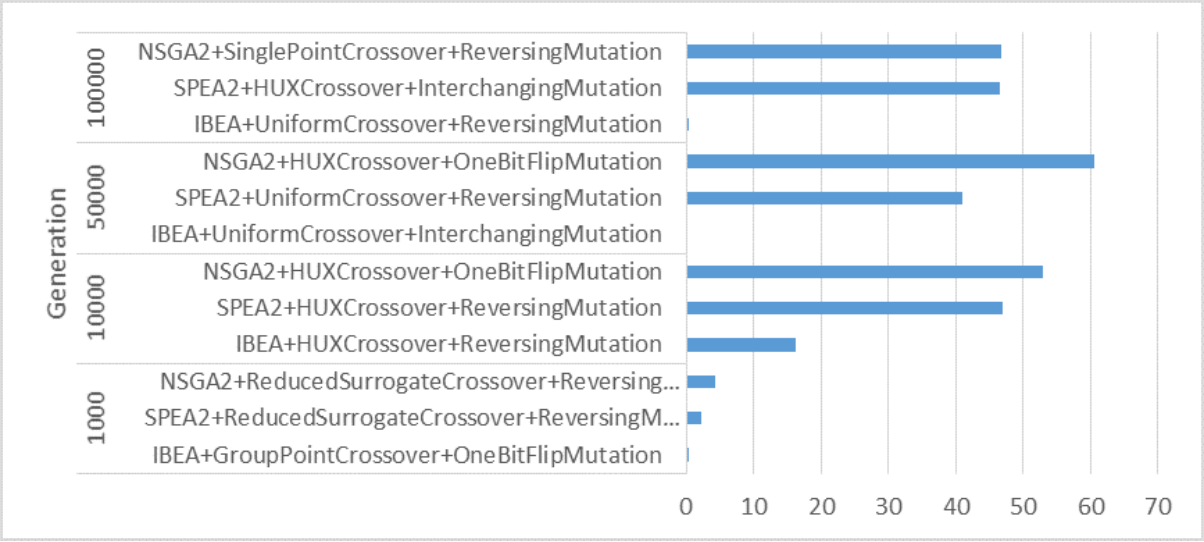

Figure 7: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_06 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

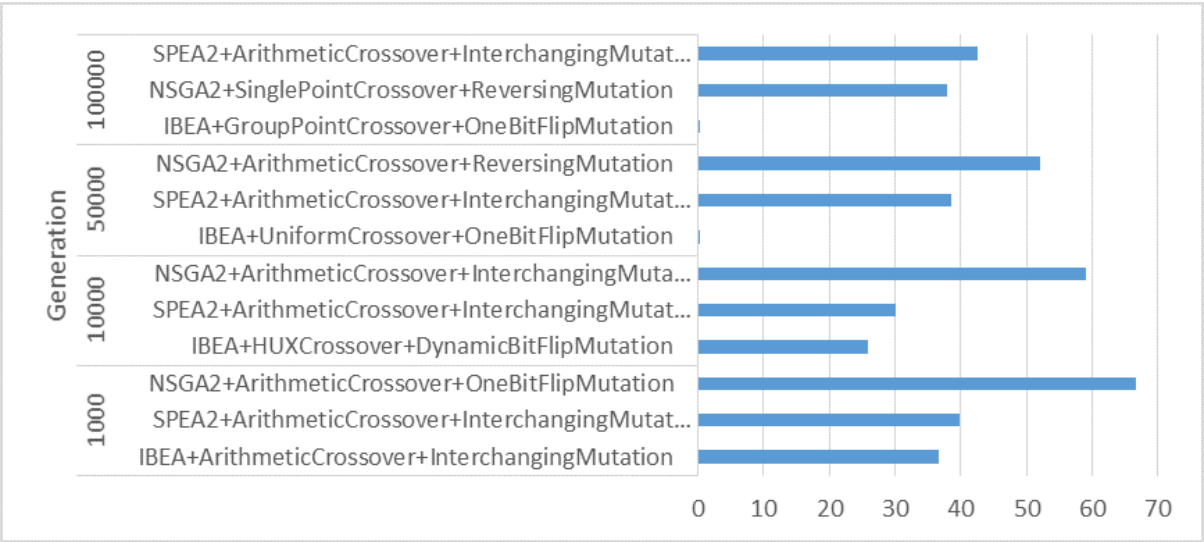

Figure 8: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_10 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

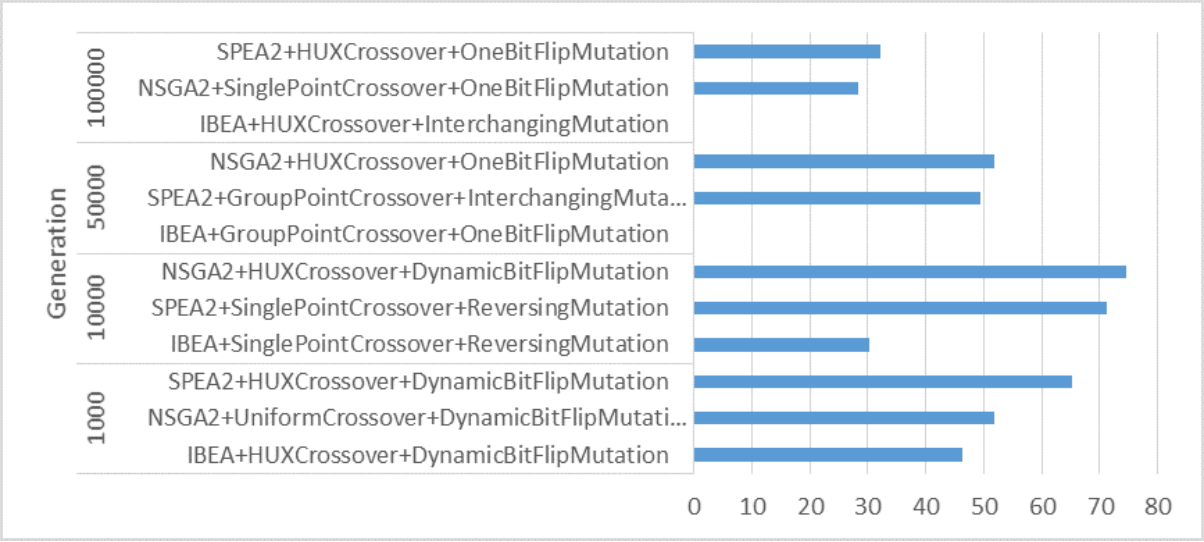

Figure 9: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_01 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

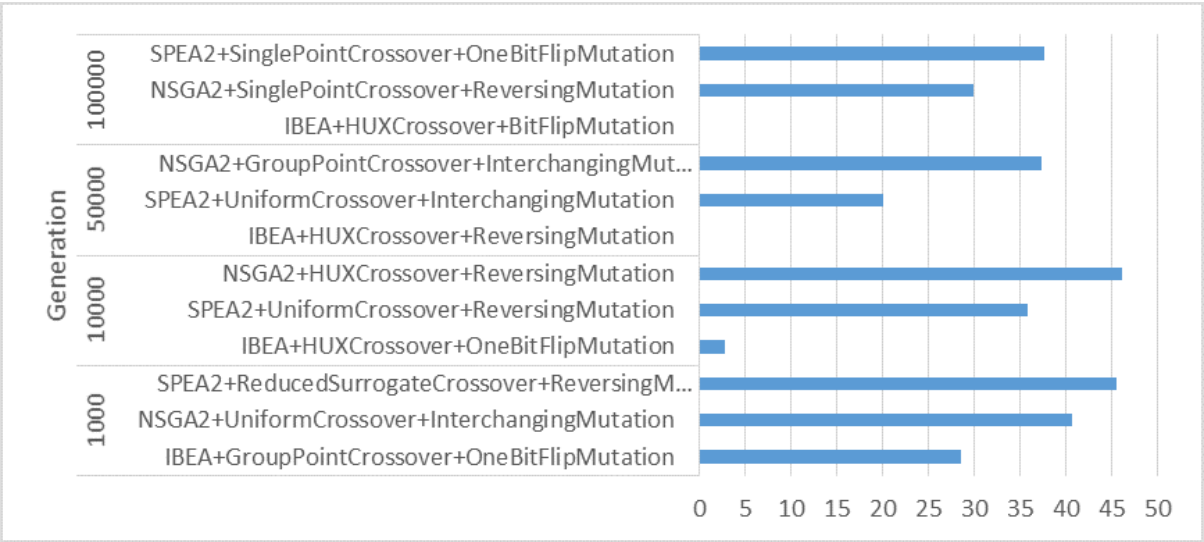

Figure 10: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_06 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

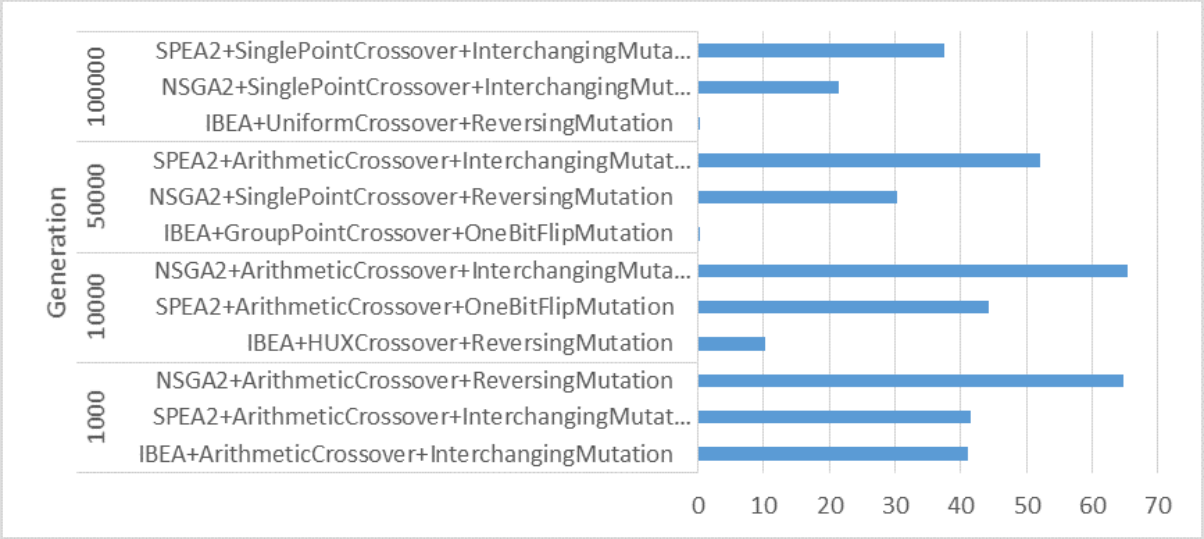

Figure 11: Best three Average scaled Error of NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_10 instances for different generations. The Error is calculated between Pareto Front and algorithm combination results. IBEA combination performs much better in 10000, 50000 and 100000 generations.

Appendix

- 10000.png

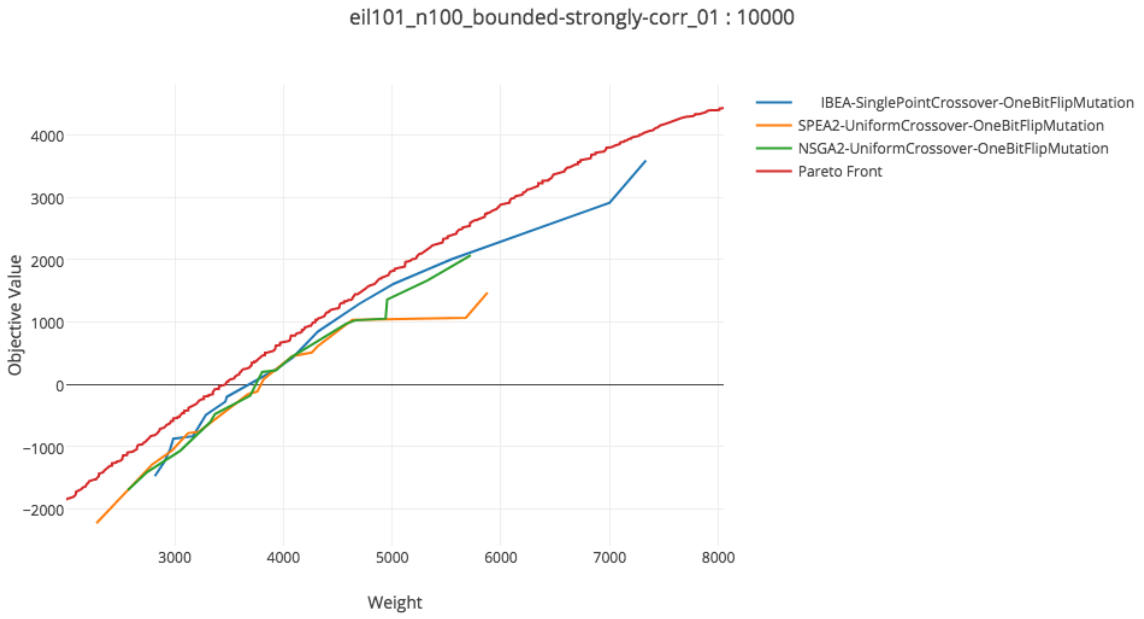

Figure 12: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

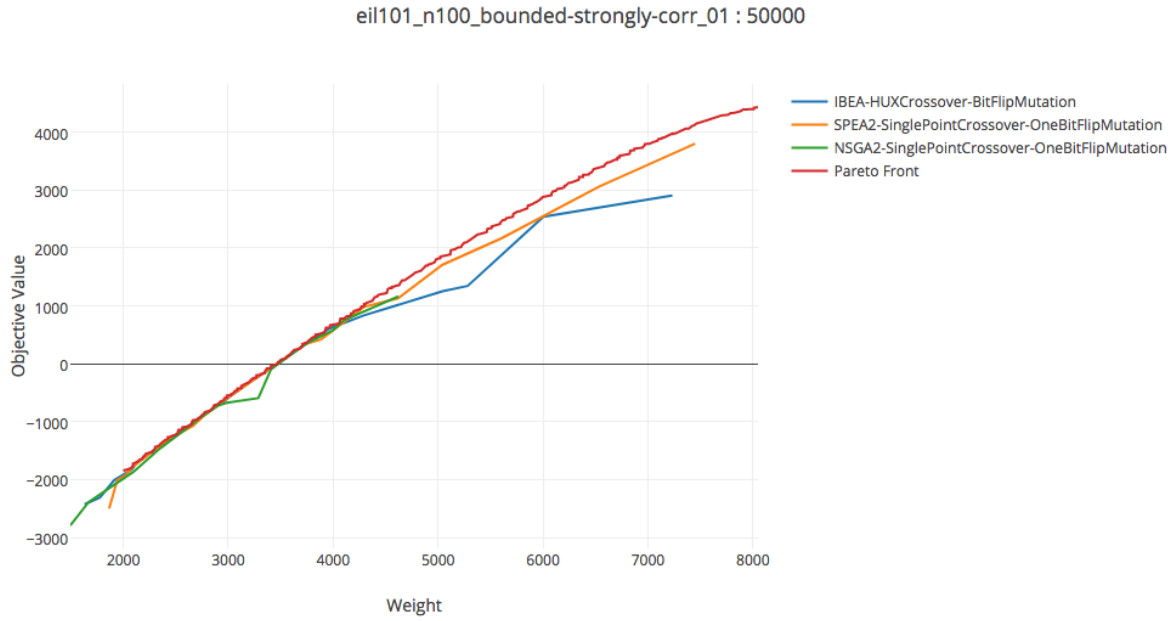

Figure 13: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

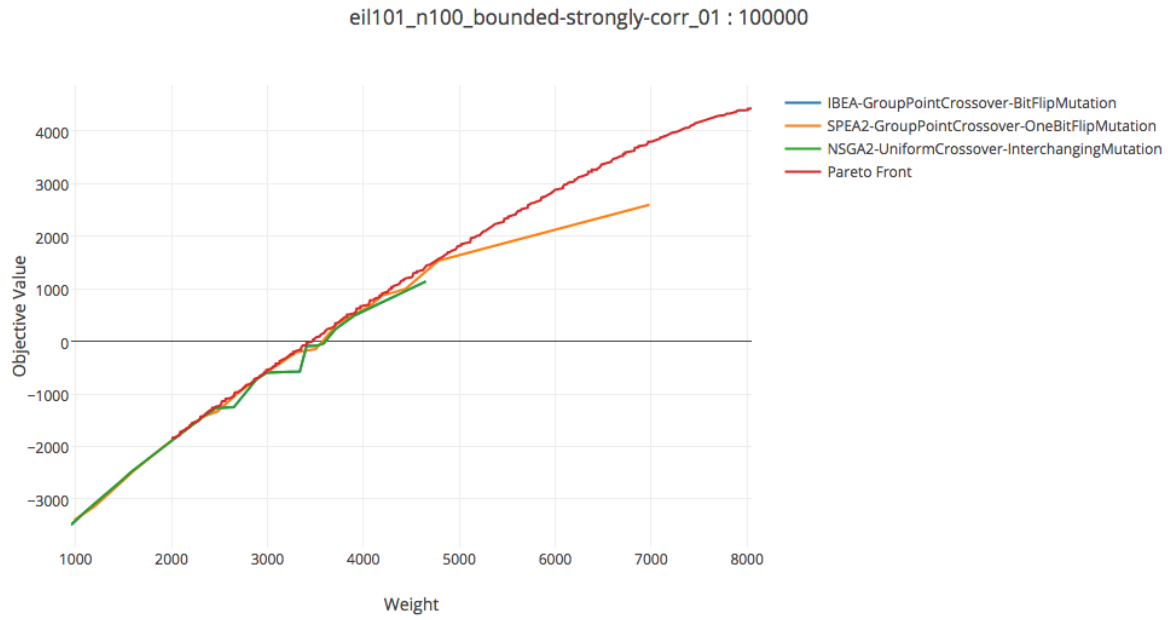

Figure 14: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

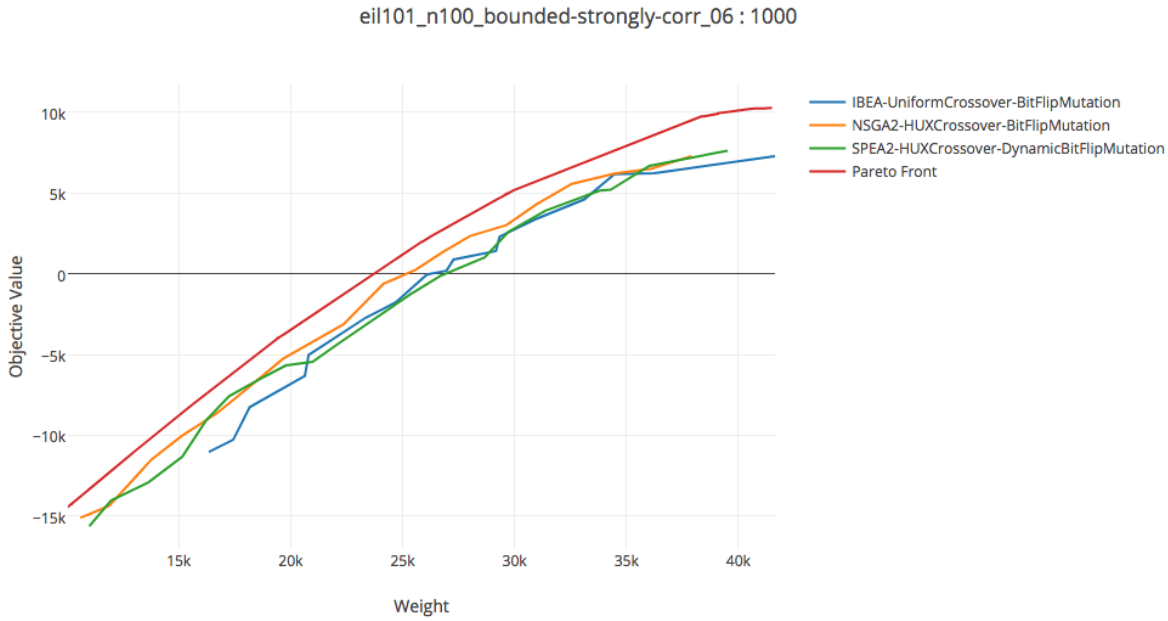

Figure 15: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

Figure 16: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

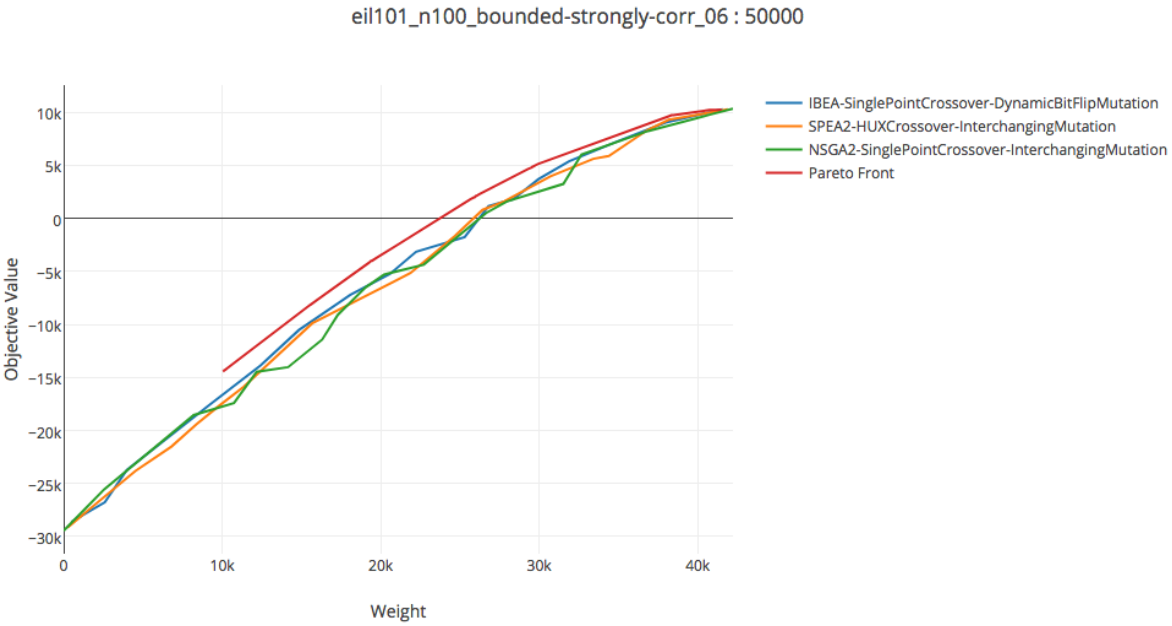

Figure 17: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

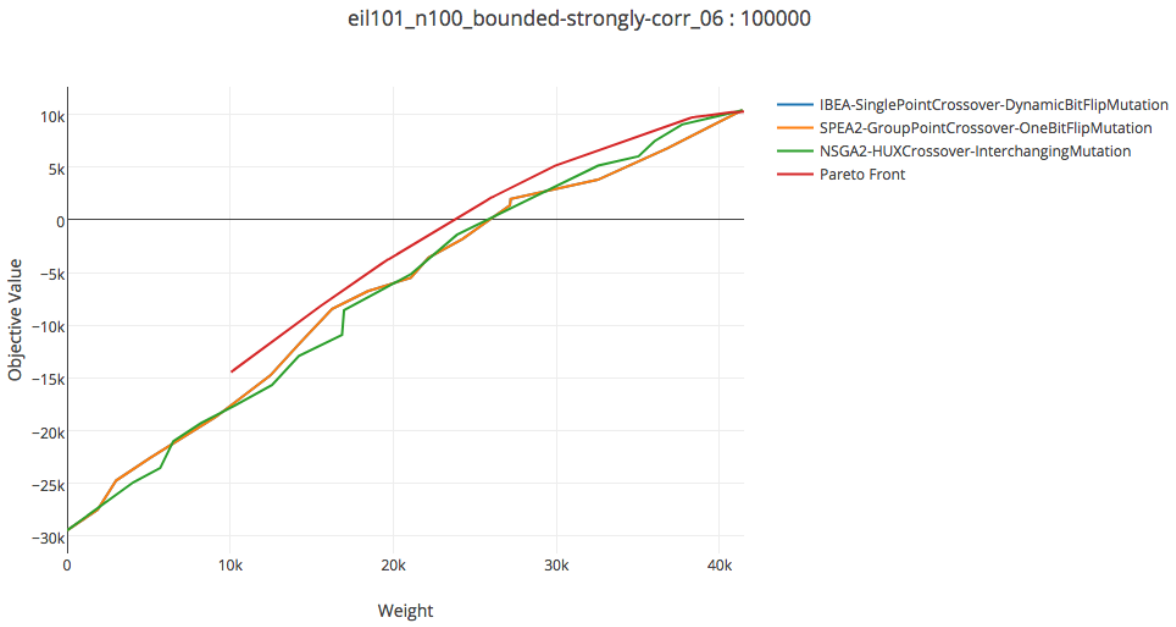

Figure 18: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

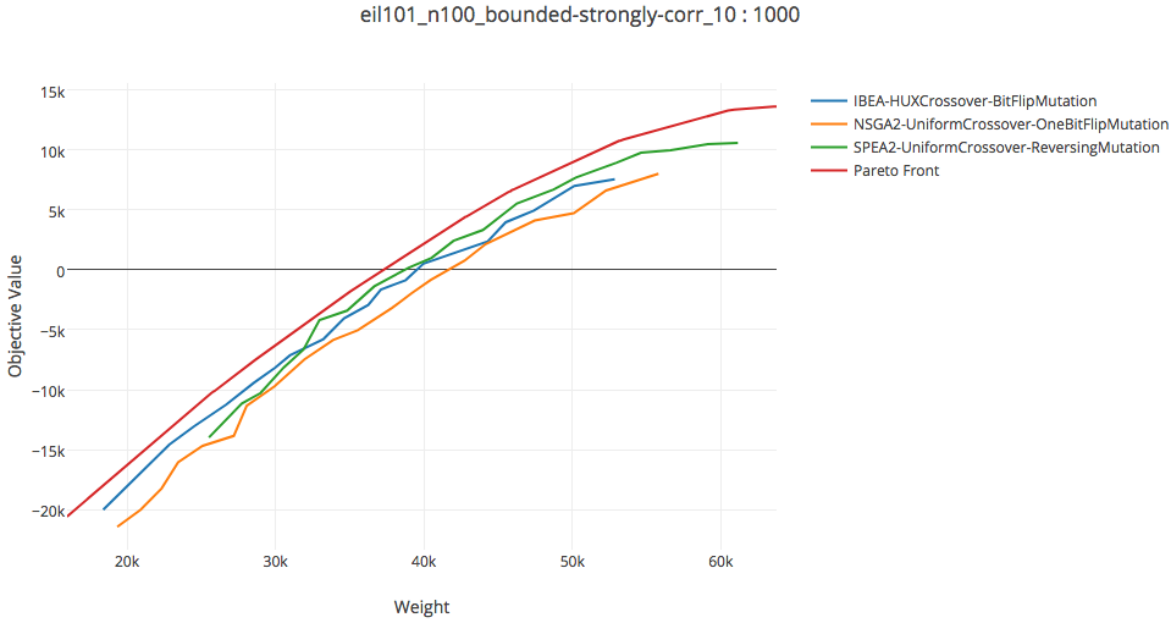

Figure 19: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

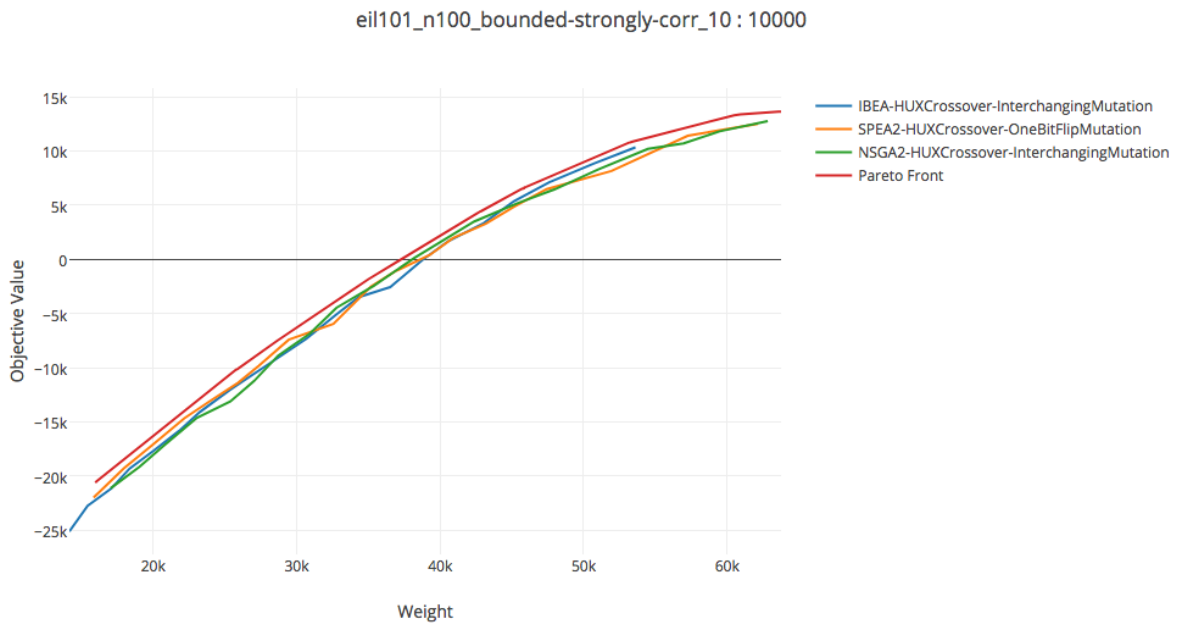

Figure 20: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

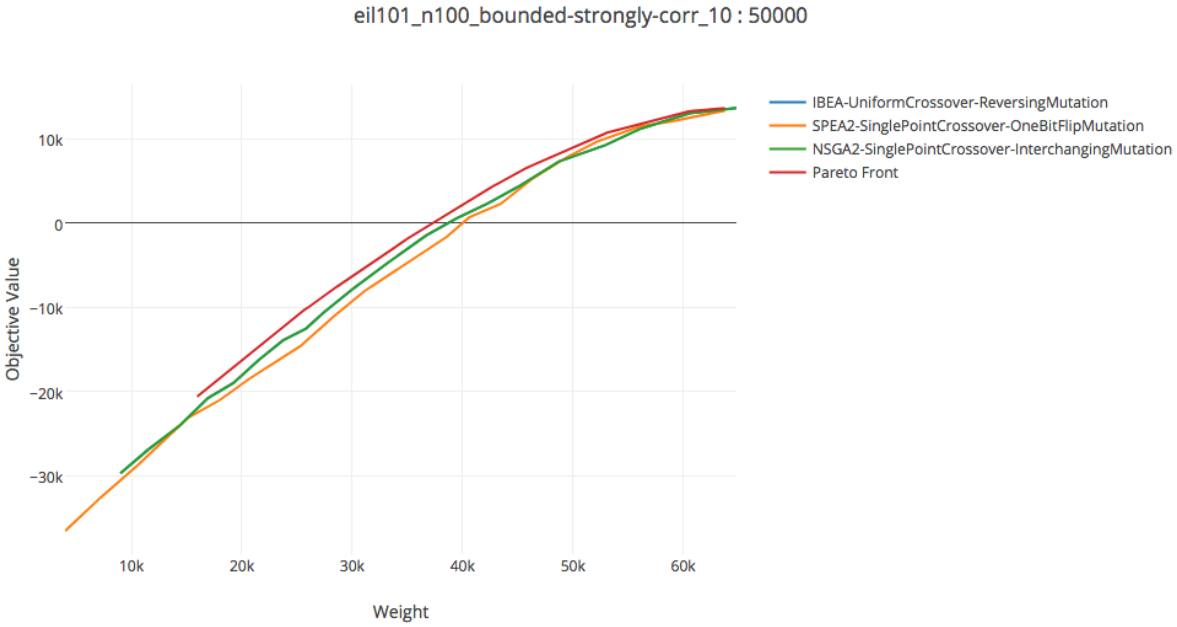

Figure 21: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

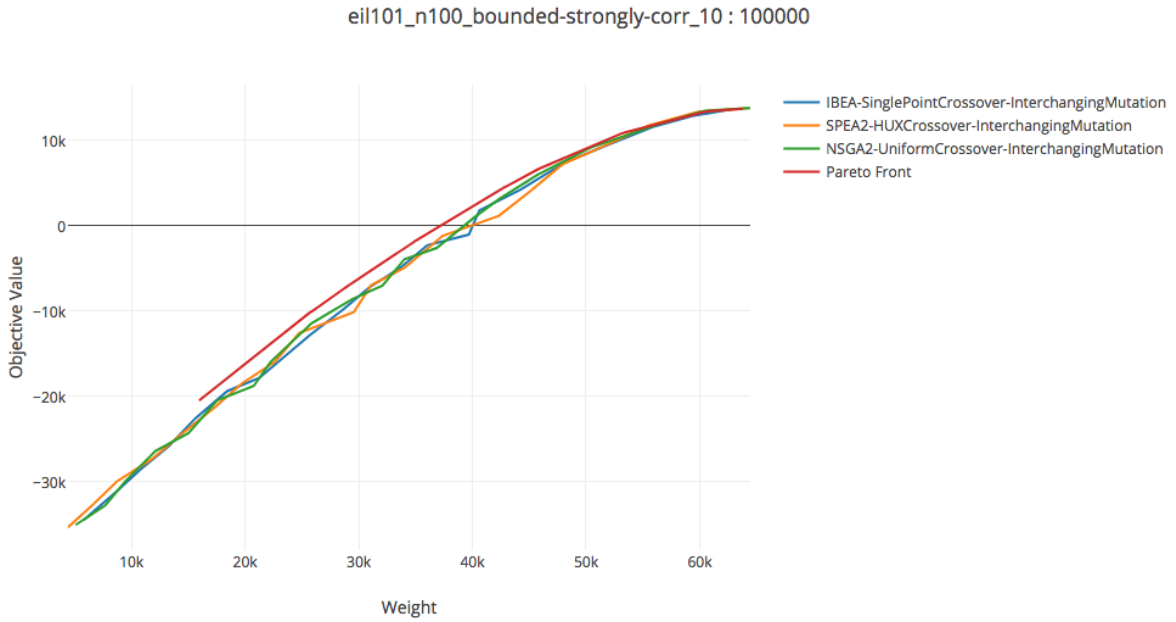

Figure 22: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n100_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 100000 generations.

- 50000.png

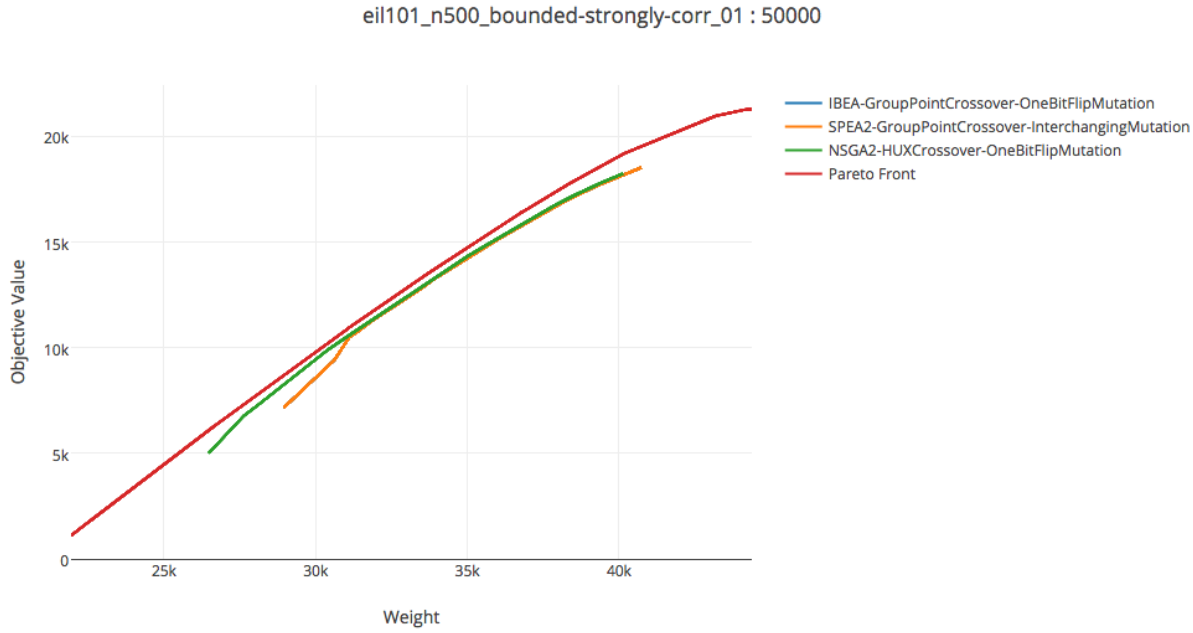

Figure 23: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

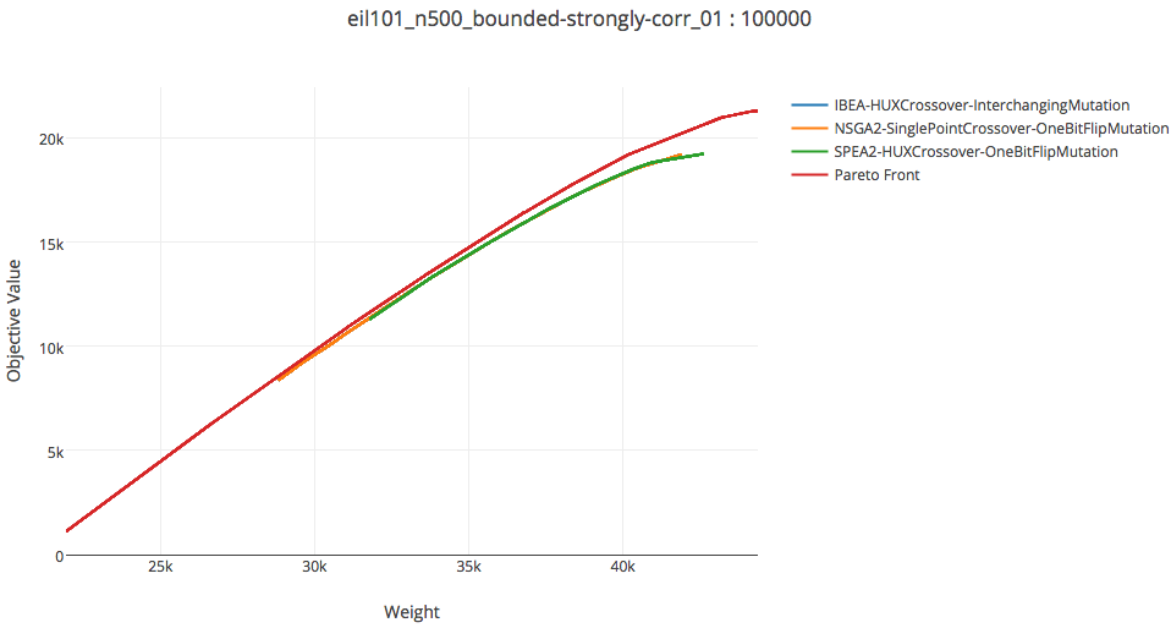

Figure 24: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

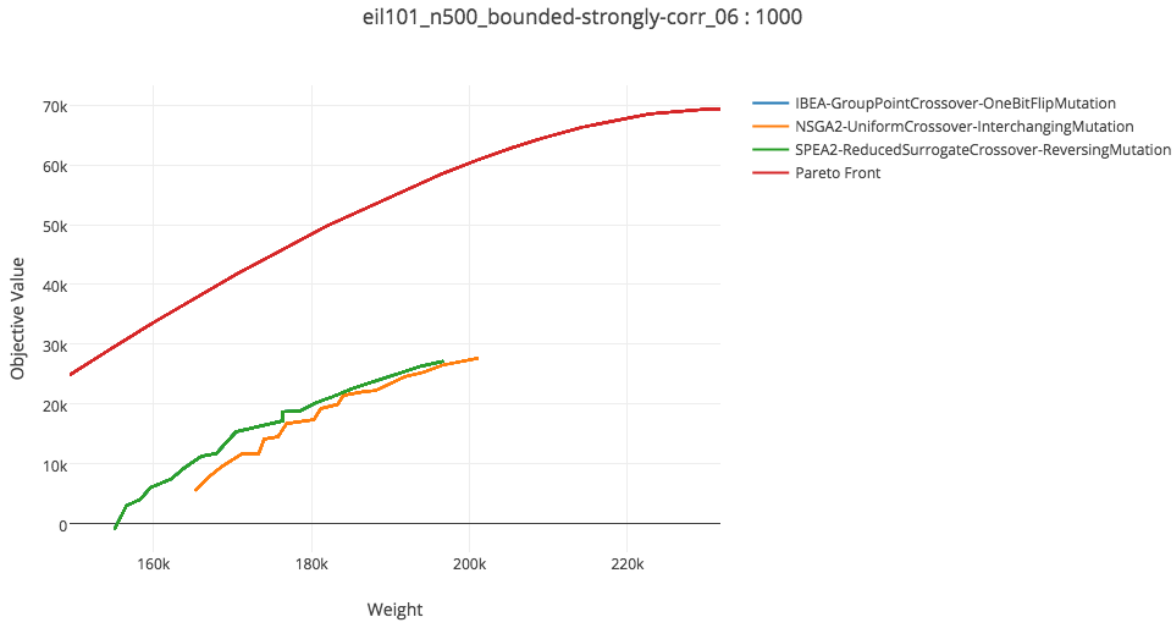

Figure 25: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

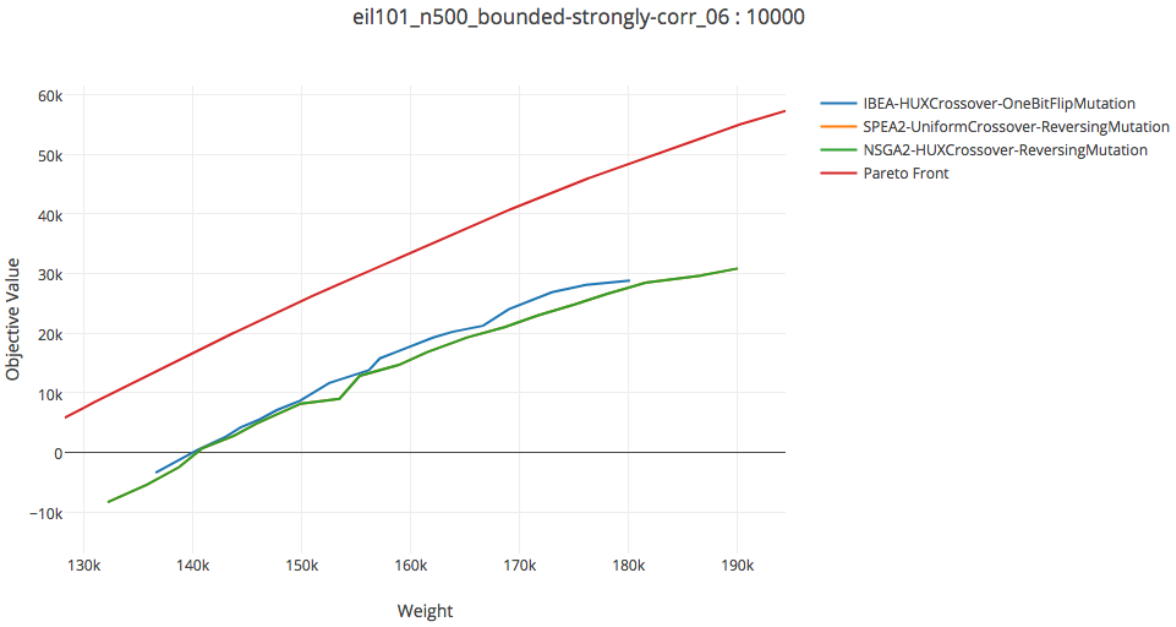

Figure 26: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

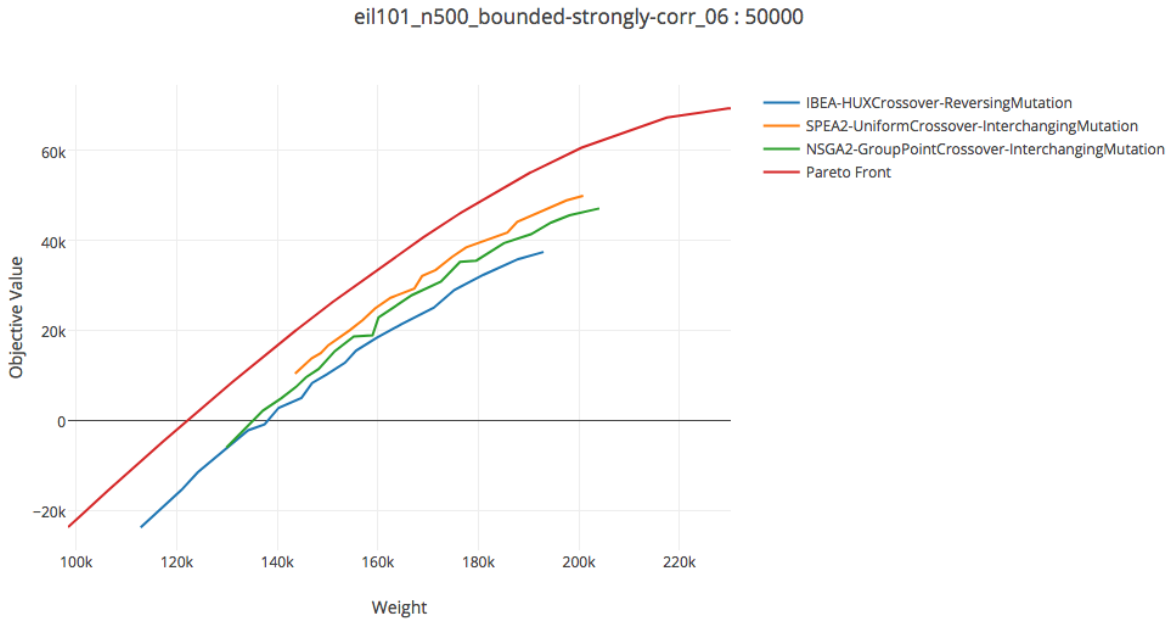

Figure 27: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 1000 generations.

- 100000.png

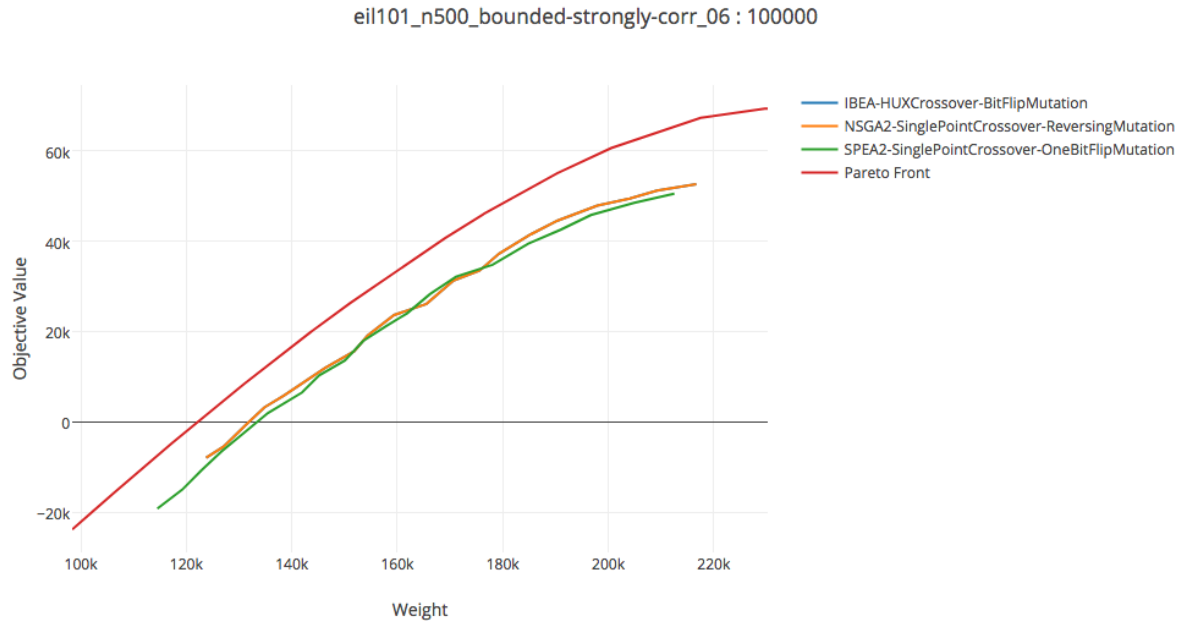

Figure 28: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

Figure 29: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

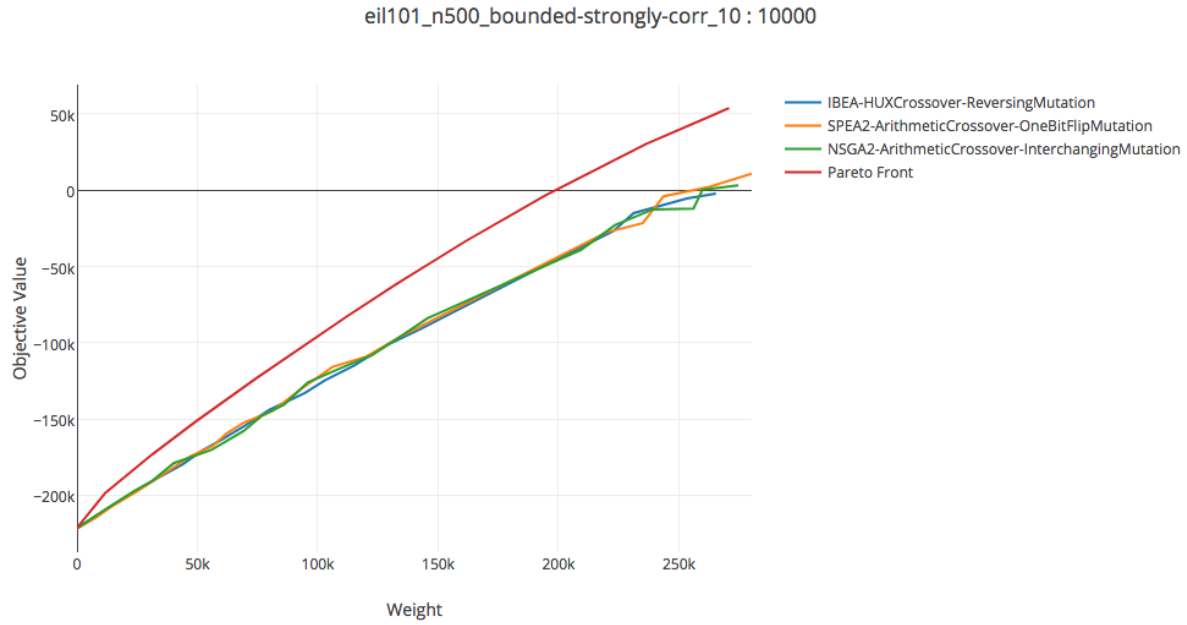

Figure 30: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

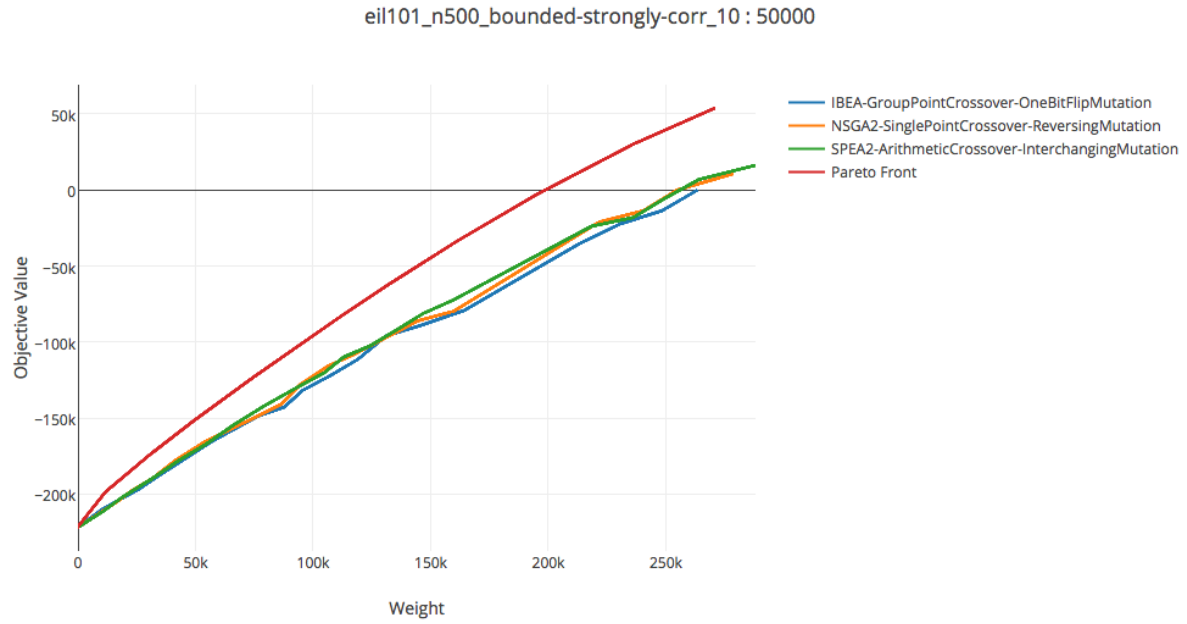

Figure 31: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

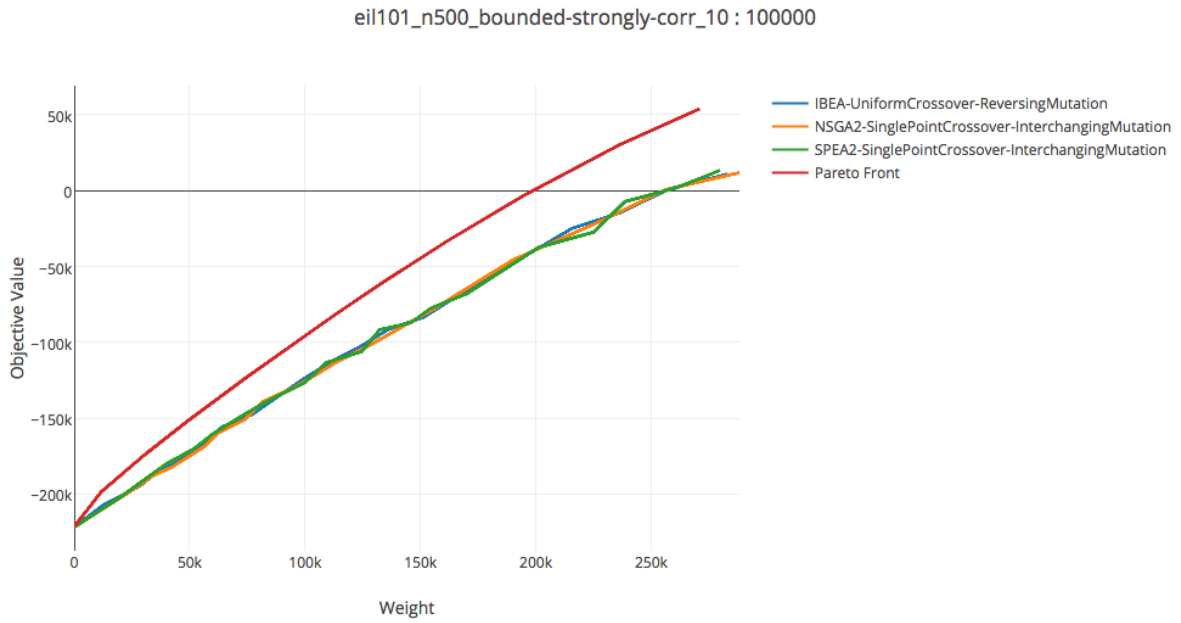

Figure 32: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n500_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 100000 generations.

- 50000.png

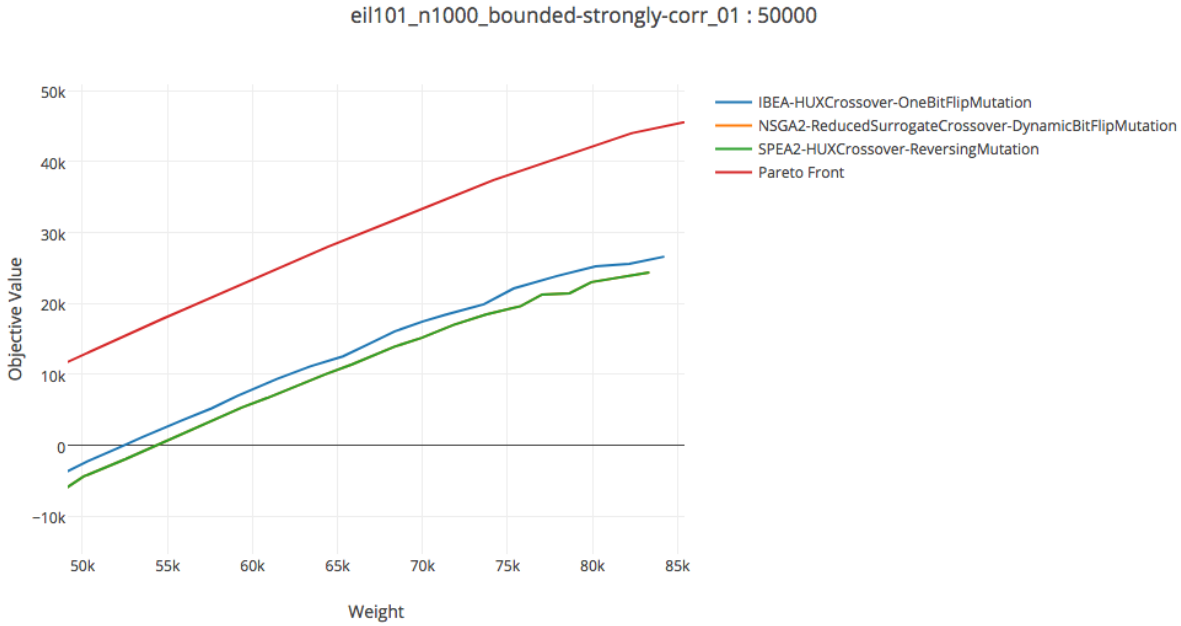

Figure 33: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

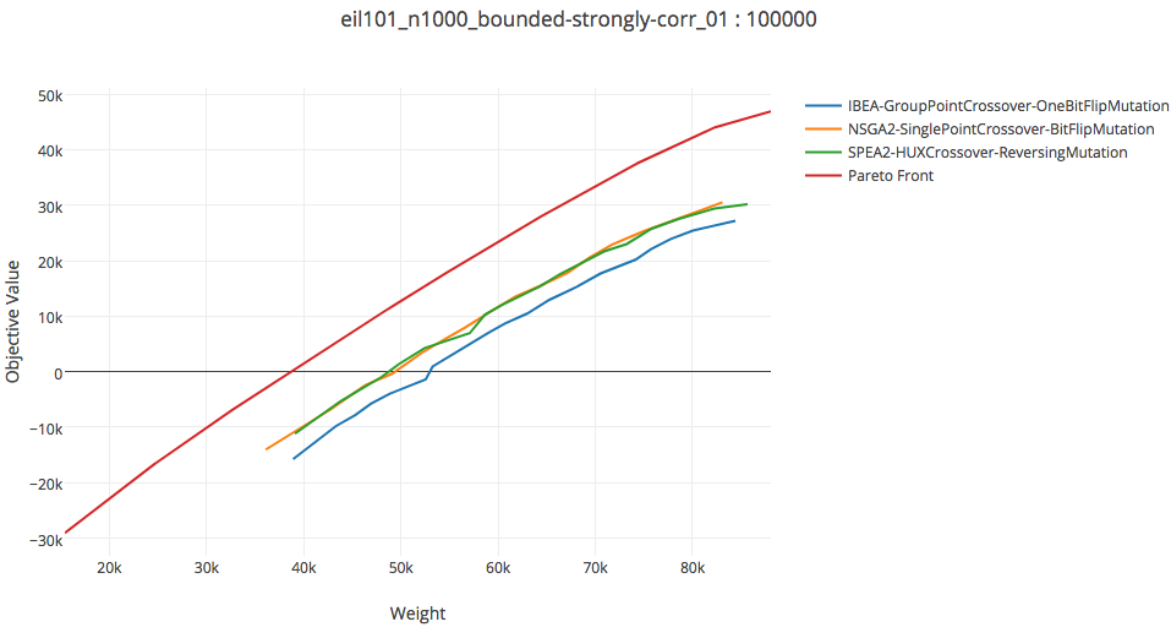

Figure 34: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_01 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

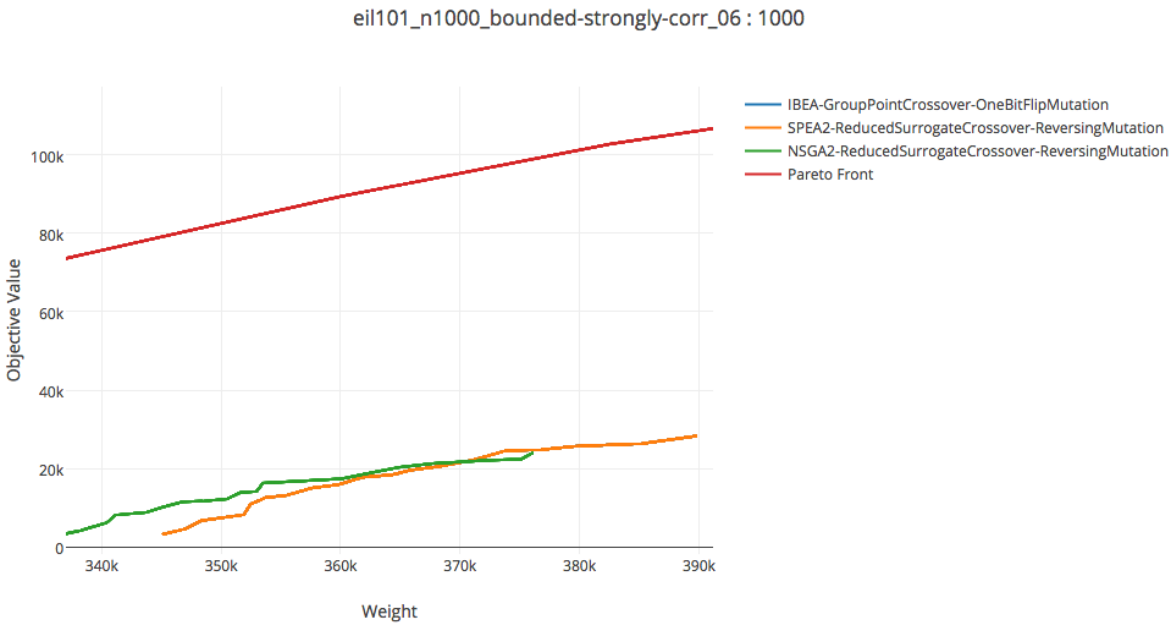

Figure 35: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

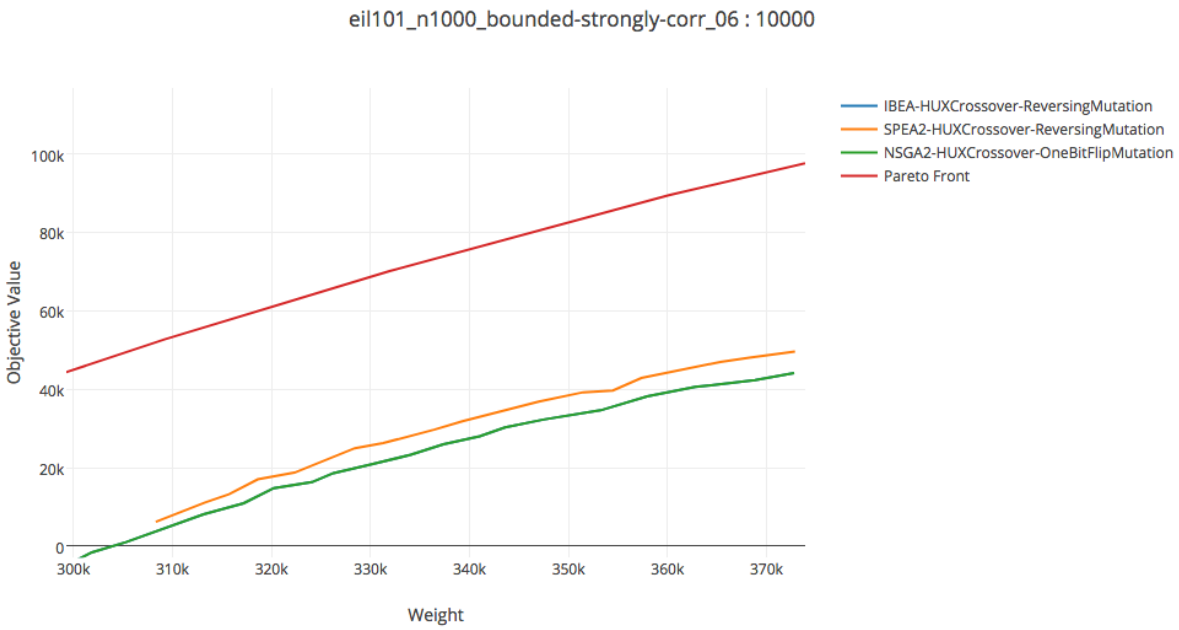

Figure 36: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

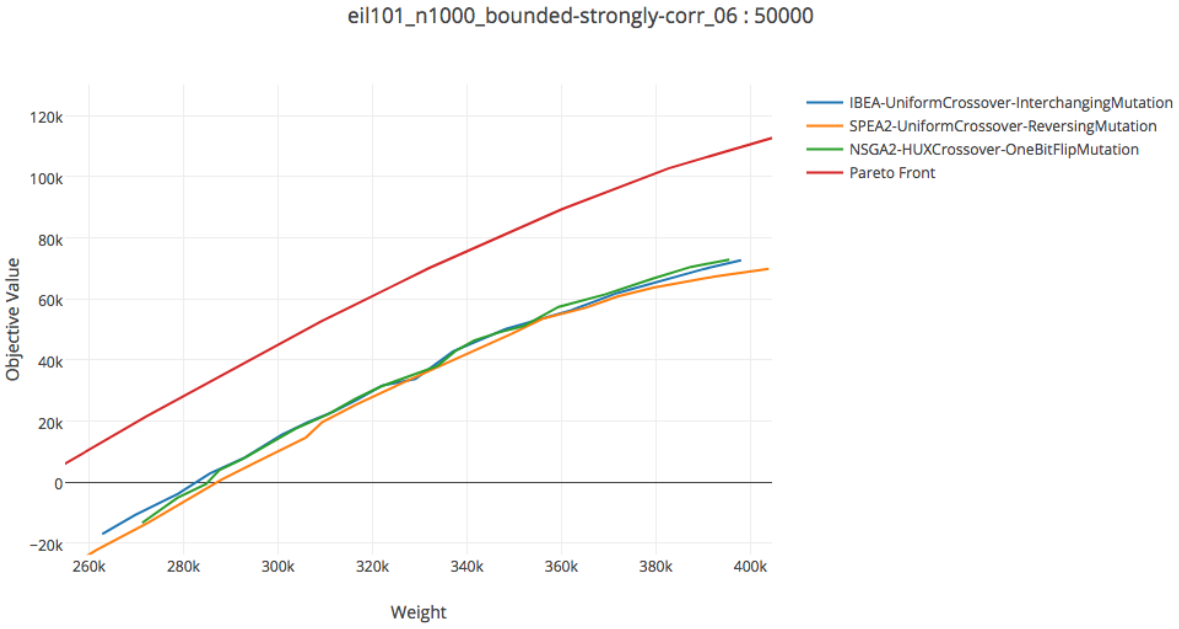

Figure 37: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

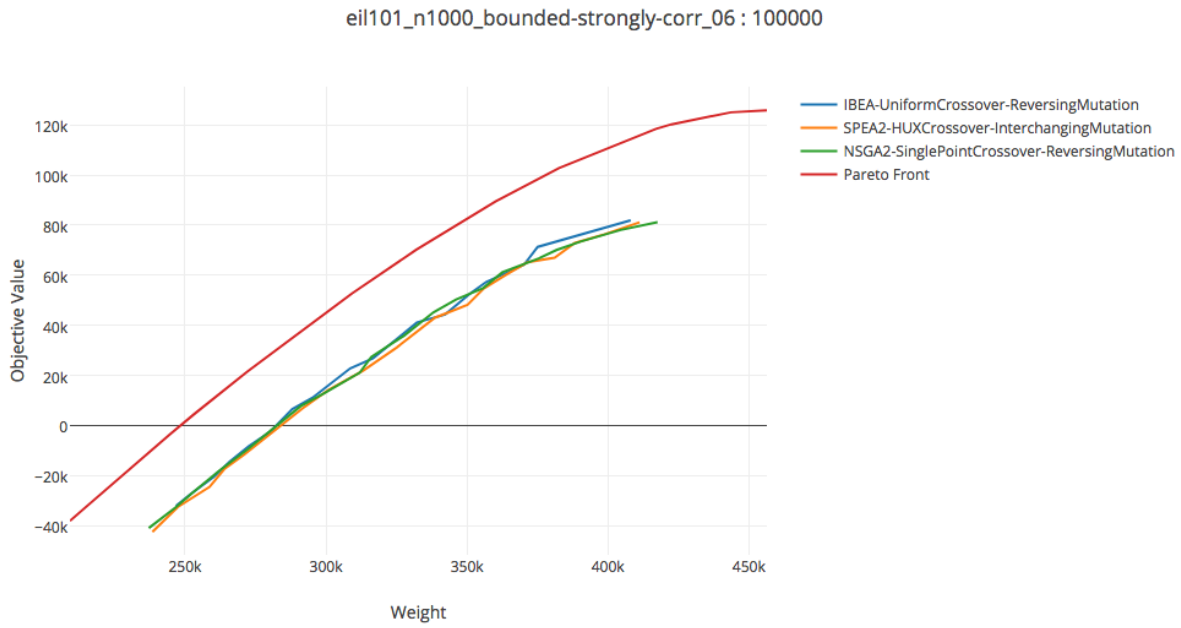

Figure 38: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_06 instances compare to Pareto-optimal front for 100000 generations.

- 1000.png

Figure 39: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 1000 generations.

- 10000.png

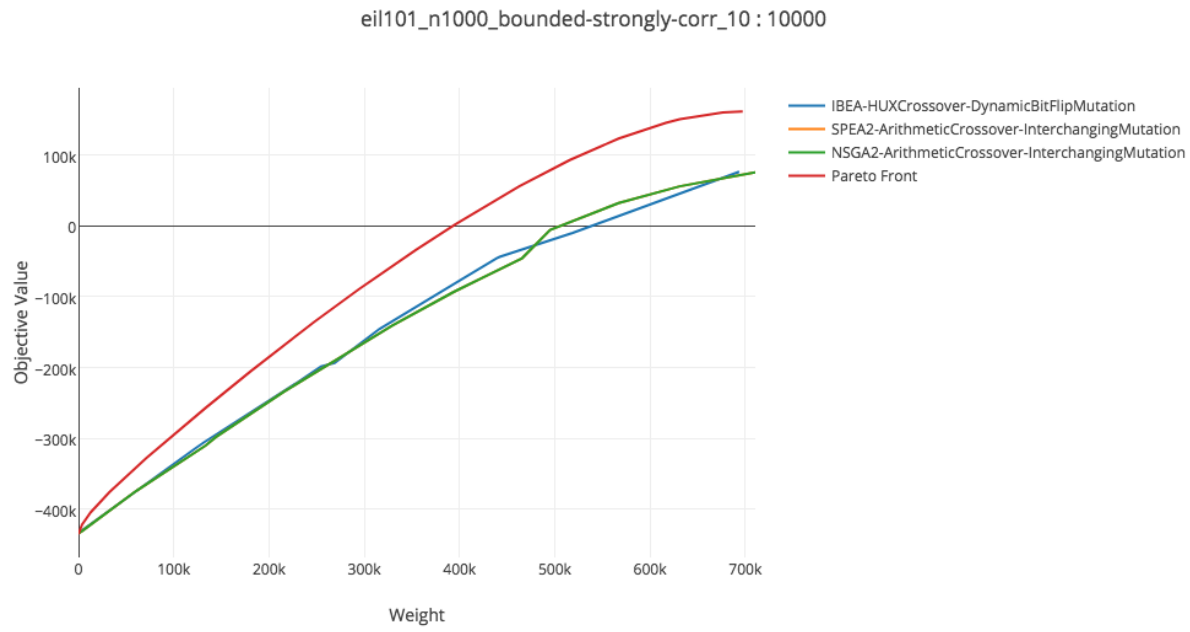

Figure 40: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 10000 generations.

- 50000.png

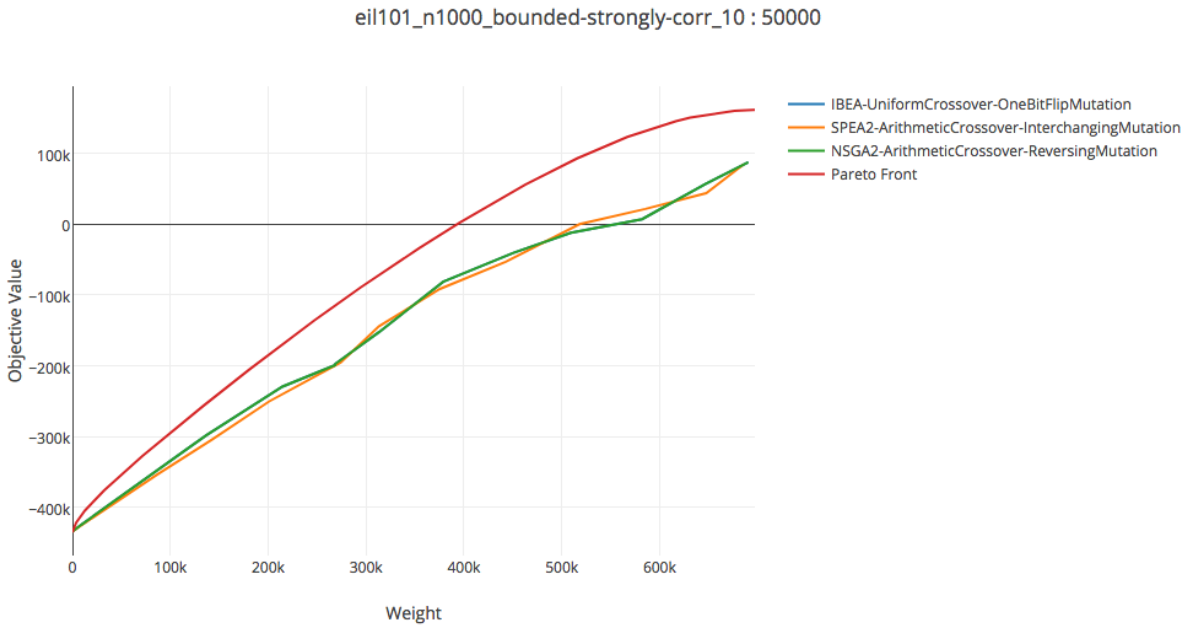

Figure 41: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 50000 generations.

- 100000.png

Figure 42: Performance of best NSGA2, SPEA2 and IBEA algorithm operator combination of eil101_n1000_bounded-strongly-corr_10 instances compare to Pareto-optimal front for 100000 generations.