A PSEUDO CODE

A.1 Genetic Algorithm

Algorithm 1 Genetic Algorithm Variant Used in Our Study

Input: pop_size, default_size, tournament_size
Output: best_configuration

```
pop ← list()
offsprings ← list()
for i in range pop_size do
    temp ← initialise individual with default value
    if i ≥ default_size then
        temp ← mutate(temp)
    end if
    offsprings[i] ← temp
end for
evaluate(offsprings)
pop ← offsprings
while budget not exhausted do
    offsprings[0: pop_size] ← tournament_selection(pop)
    for i in range (1, pop_size, 2) do
        if random[0, 1] ≤ cxpb_chrm then
            offsprings[i-1], offsprings[i] ← crossover(offsprings[i-1], offsprings[i])
        end if
    end for
    for i in range pop_size do
        if random[0, 1] ≤ mutpb_chrm then
            offsprings[i] ← mutate(offsprings[i])
        end if
    end for
    evaluate(offsprings)
    pop ← offsprings
end while
```

Algorithm 2 Uniform Crossover Operator

Input: ind1 and ind2
Output: ind1 and ind2

```
max_length ← max(ind1, ind2)
for i in range max_length do
    if random[0, 1] > cxpb_gene then
        continue
    end if
    if gene is categorical parameter or GI edit then
        ind1[i], ind2[i] ← ind2[i], ind1[i]
    else if ind1[i] <= ind2[i] then
        ind1[i] ← random[ind1[i], ind2[i]]
        ind2[i] ← random[ind1[i], ind2[i]]
    else
        ind1[i] ← random[ind2[i], ind1[i]]
        ind2[i] ← random[ind2[i], ind1[i]]
    end if
end for
return ind1, ind2
```

Algorithm 3 Mutation Strategy Based on Our Genetic Algorithm

Input: ind
Output: ind

```
for i, gene in range enumerate(ind.get_genes()) do
    if random[0, 1] > mutpb_gene then
        continue
    end if
    if gene is categorical or numerical parameter then
        genes[i] ← random(genes[i])
    else if gene is GI edit then
        delete genes[i]
    end if
end for
if GI in possible_edits and random[0, 1] ≤ statement_insertion_prob then
    ind.append(new statement_edit)
end if
return ind
```

A.2 Local Search

Algorithm 4 First Improvement [Blot and Petke 2022]

```
best ← empty_patch
while budget not exhausted do
    if random[0, 1] < delete_prob then
        mutant ← remove one random edit.
    else
        mutant ← add one new edit.
    end if
    if f(mutant) ≤ f(best) then
        best ← mutant
    end if
end while
return best
```

B EXPERIMENT RESULTS

Table 3: Number of parameter and statement edit(s) for each k-fold of each algorithm, for joint search space. These are from the training phase.

	LS		GA	
k	parameter	statement	parameter	statement
1	3	6	6	0
2	4	3	13	3
3	3	7	10	2
4	3	5	6	2
5	7	6	8	0
6	0	7	9	1
7	5	6	10	5
8	2	12	12	2
9	3	11	8	0
10	6	6	13	1
avg	3.6	6.9	9.5	1.6

Table 4: Final ranking from the Test phase. We select the single best variant from each experiment. The ranking excludes AC + GI with GA, because no generated variants generalised to the test instances.

Rank	Technique	Speed Up
1	GI with LS	18.05%
2	AC with LS	17.75%
3	GI with GA	16.12%
4	AC with GA	14.32%
5	AC + GI with LS	9.88%

Table 5: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Algorithm Configuration with First Improvement Local Search. The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-57.91%	-33.24%	N/A
2	-61.81%	-22.44%	-9.48%
3	-46.97%	-14.69%	+1.58%
4	-35.92%	+18.91%	N/A
5	-50.29%	-15.63%	N/A
6	-18.36%	+33.37%	N/A
7	-43.21%	-17.71%	-11.78%
8	-53.59%	-26.51%	+6.63%
9	-54.56%	+2.99%	N/A
10	-37.02%	-24.66%	**-17.75%**

Table 6: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Genetic Improvement with First Improvement Local Search. The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-52.16%	-13.11%	-16.62%
2	-28.12%	+7.96%	N/A
3	-32.57%	-10.44%	-13.26%
4	-13.02%	+10.75%	N/A
5	-31.67%	-8.63%	**-18.05%**
6	-47.81%	-0.92%	N/A
7	-7.33%	N/A	N/A
8	-42.47%	N/A	N/A
9	-51.73%	-22.18%	+5.63%
10	-19.54%	+16.69%	N/A

Table 7: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for joint search space with First Improvement Local Search. The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-18.08%	N/A	N/A
2	-51.11%	+25.80%	N/A
3	-35.14%	-13.94%	-5.64%
4	-67.92%	+0.71%	N/A
5	-30.57%	N/A	N/A
6	-35.42%	+20.32%	N/A
7	-10.61%	+27.34%	N/A
8	-50.72%	-26.94%	-6.30%
9	-11.98%	N/A	N/A
10	-26.41%	-12.24%	**-9.88%**

Table 8: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Algorithm Configuration with Genetic Algorithm (population 10). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-48.07%	+8.61%	N/A
2	-42.54%	-11.27%	N/A
3	-46.27%	-12.78%	+0.55%
4	-23.53%	+44.56%	N/A
5	-40.62%	-19.23%	-13.31%
6	-28.75%	-5.67%	-4.15%
7	-46.56%	-9.17%	N/A
8	-40.37%	-24.67%	**-14.32%**
9	-59.88%	-19.62%	-3.75%
10	-32.00%	+9.31%	N/A

Table 9: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Genetic Improvement with Genetic Algorithm (population 10). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-37.85%	+15.34%	N/A
2	-51.85%	-19.20%	-9.64%
3	-37.60%	+0.20%	N/A
4	-42.97%	+3.22%	N/A
5	-32.76%	-23.90%	N/A
6	-33.10%	-9.09%	-8.11%
7	-16.75%	+9.31%	N/A
8	-41.80%	-12.66%	**-16.12%**
9	-40.54%	-5.92%	+8.13%
10	-19.72%	+16.32%	N/A

Table 10: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for joint search space with Genetic Algorithm (population 10). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-45.97%	+5.98%	N/A
2	-50.49%	-8.62%	N/A
3	-32.40%	+20.04%	N/A
4	-20.61%	+31.67%	N/A
5	-29.23%	+13.48%	N/A
6	-44.98%	-19.15%	+2.38%
7	-66.82%	-6.17%	N/A
8	-38.92%	-4.16%	N/A
9	-53.60%	+3.11%	N/A
10	-35.49%	+39.78%	N/A

Table 11: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Algorithm Configuration with Genetic Algorithm (population 100). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-45.52%	-2.49%	**-18.61%**
2	-40.37%	+15.97%	N/A
3	-35.24%	+3.99%	N/A
4	-51.01%	-8.27%	-17.83%
5	-36.92%	-20.75%	+2.57%
6	-29.07%	-12.90%	-2.19%
7	-40.78%	-4.87%	-0.54%
8	-47.68%	-30.42%	+0.68%
9	-54.70%	-14.25%	+7.47%
10	-27.34%	-20.26%	-5.29%

Table 12: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for Genetic Improvement with Genetic Algorithm (population 100). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-36.05%	+30.52%	N/A
2	-54.17%	+12.57%	N/A
3	-39.88%	-4.69%	-3.65%
4	-2.69%	-9.42%	-8.11%
5	-7.94%	+45.31%	N/A
6	-36.49%	-13.95%	**-19.06%**
7	-1.28%	N/A	N/A
8	-23.88%	N/A	N/A
9	-52.51%	+66.69%	N/A
10	-19.33%	+10.65%	N/A

Table 13: % runtime improvement of MiniSAT variant found after each experimental stage for each k-fold, for joint search space with Genetic Algorithm (population 100). The negative and positive sign means speed up and slow down, respectively, in terms of CPU instructions.

k	Train	Validate	Test
1	-48.02%	+8.75%	N/A
2	-41.03%	+3.76%	N/A
3	-43.78%	-10.08%	-2.54%
4	-20.61%	-7.44%	+7.44%
5	-37.22%	+6.80%	N/A
6	-26.77%	-6.36%	**-5.59%**
7	-31.42%	+24.62%	N/A
8	-38.53%	+20.84%	N/A
9	-48.88%	-31.45%	+4.08%
10	-31.68%	-14.81%	-2.60%

Table 14: The number of different types of GI edits that First Improvement Local Search generated, during training phase.

Search Space	k	Insertion	Deletion	Replacement
Joint	1	2	3	1
	2	0	3	0
	3	1	6	0
	4	2	0	3
	5	1	4	1
	6	1	3	3
	7	1	3	2
	8	3	7	2
	9	3	5	3
	10	1	3	2
GI	1	7	5	2
	2	2	9	1
	3	1	9	4
	4	1	7	2
	5	0	9	2
	6	1	11	4
	7	1	8	0
	8	1	10	3
	9	3	4	2
	10	0	1	3

Table 15: The number of different types of GI edits that the Genetic Algorithm generated, during training phase.

Search Space	k	Insertion	Deletion	Replacement
Joint	1	0	0	0
	2	0	2	1
	3	0	0	2
	4	0	1	1
	5	0	0	0
	6	0	1	0
	7	1	4	0
	8	2	0	0
	9	0	0	0
	10	0	1	0
GI	1	0	5	4
	2	1	7	3
	3	0	4	1
	4	1	4	2
	5	1	5	2
	6	0	1	0
	7	0	7	1
	8	3	3	0
	9	0	6	2
	10	1	2	4

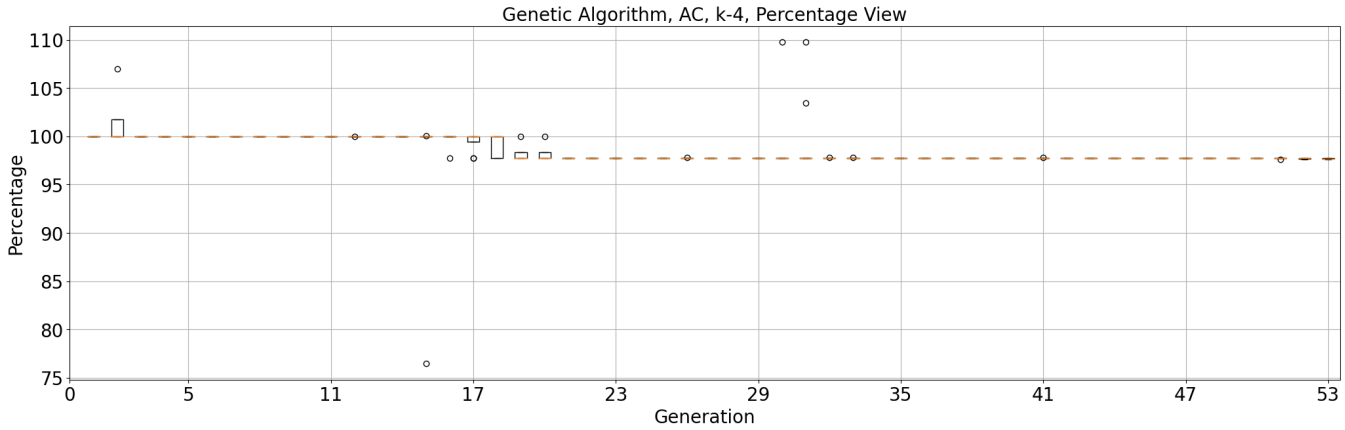

Figure 1: Relative fitness % with respect to original software for each patch generated in K-4 found in the Algorithm Configuration with Genetic Algorithm experiment, during the training phase.

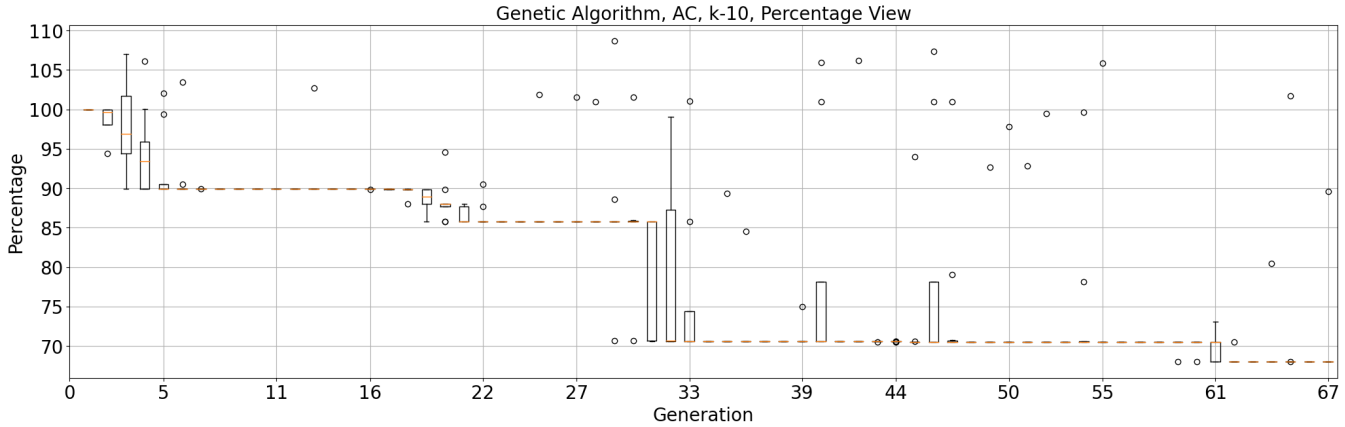

Figure 2: Relative fitness % with respect to original software for each patch generated in K-10 found in the Algorithm Configuration with Genetic Algorithm experiment, during the training phase.

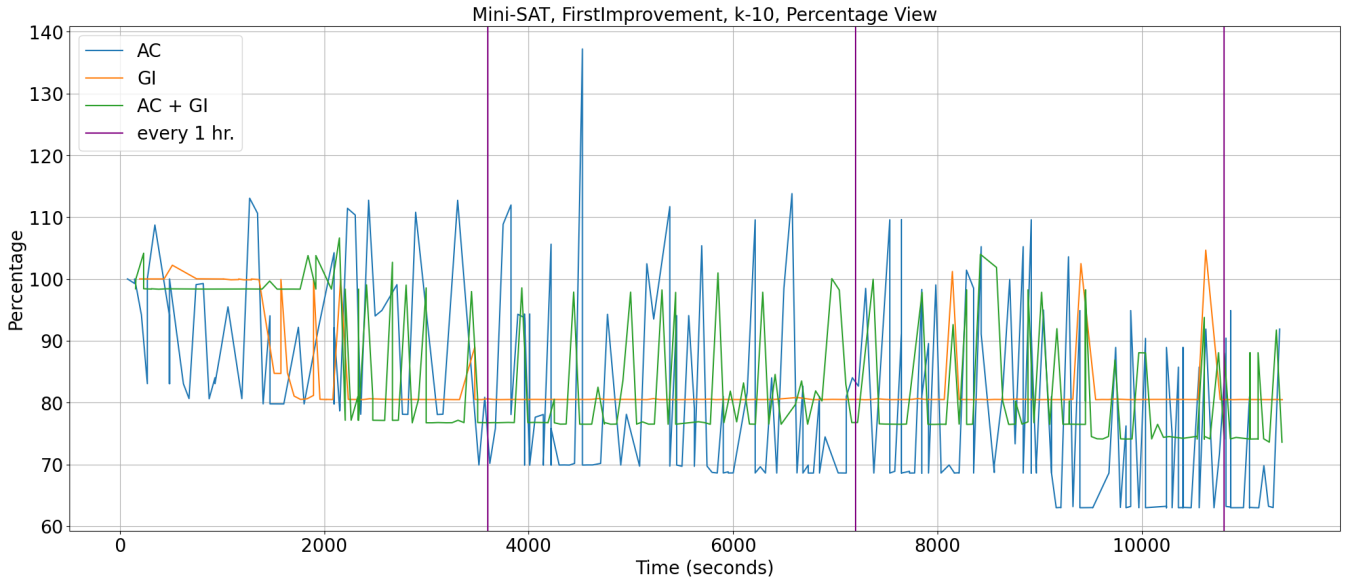

Figure 3: Relative fitness % with respect to original software for each patch generated in K-10 found in the First Improvement Local Search experiment during the training phase.

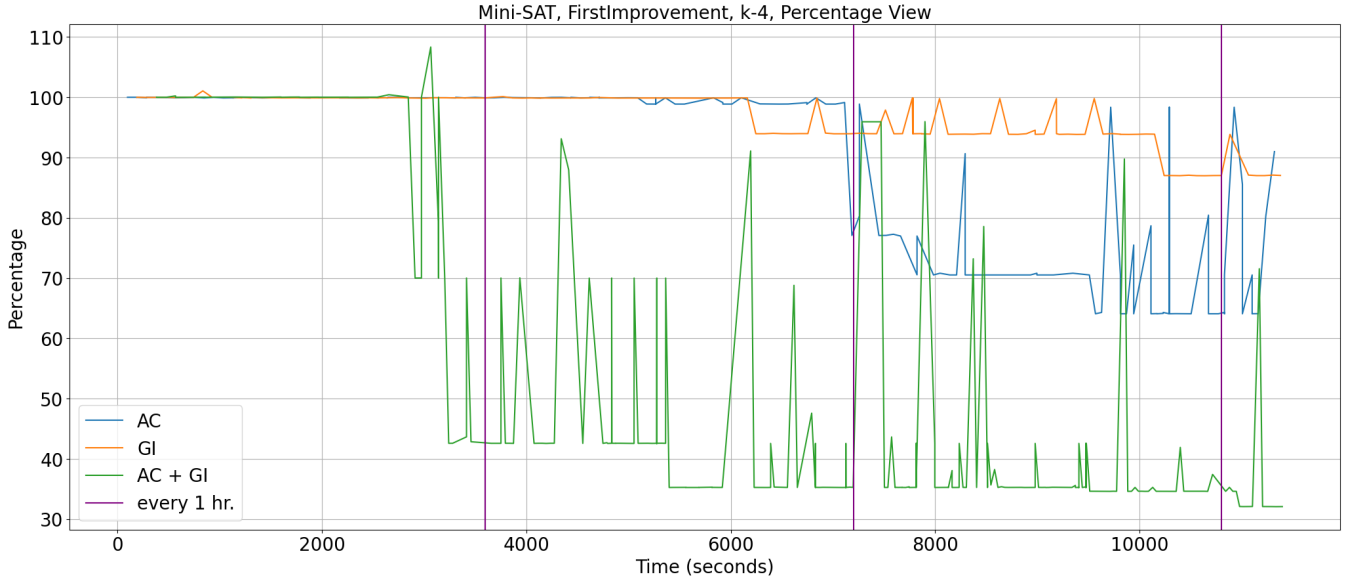

Figure 4: Relative fitness % with respect to original software for each patch generated in K-4 found in the First Improvement Local Search experiment during the training phase.

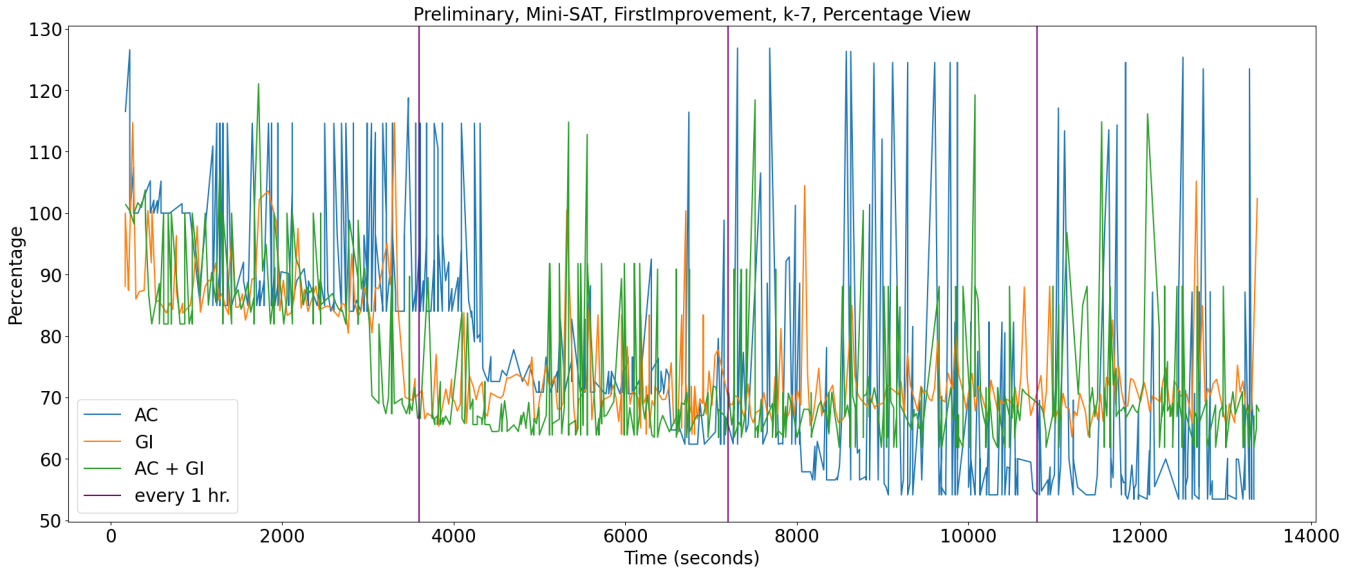

Figure 5: Preliminary result to demonstrate that the new budget provides sufficient time for AC and GI to converge. This graph shows k-7 for all search spaces.

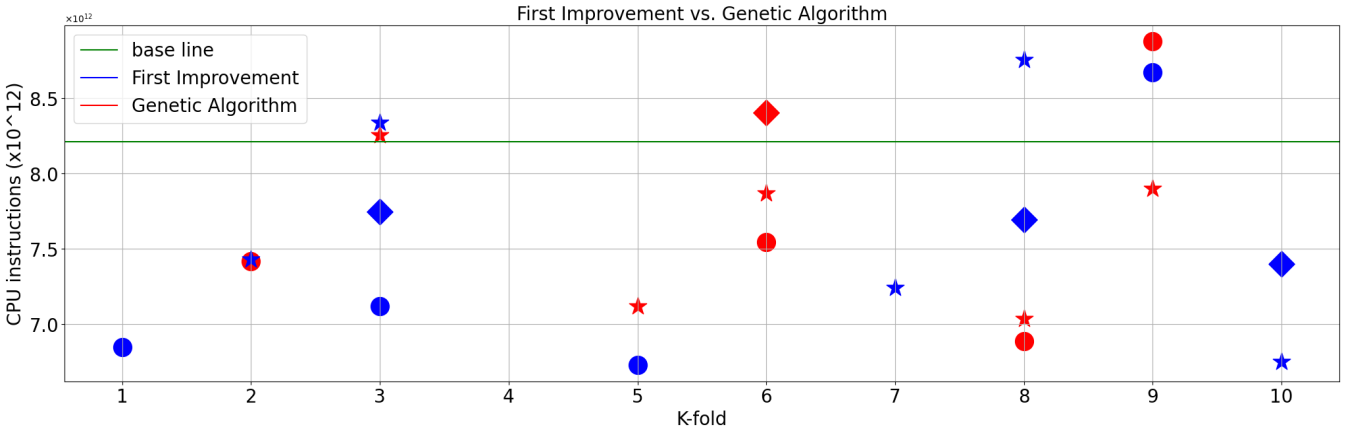

Figure 6: Comparison between Local Search and Genetic Algorithm, from the test phase. [Star: Algorithm Configuration, Circle: Genetic Improvement on Statement, and Diamond: Joint search space]

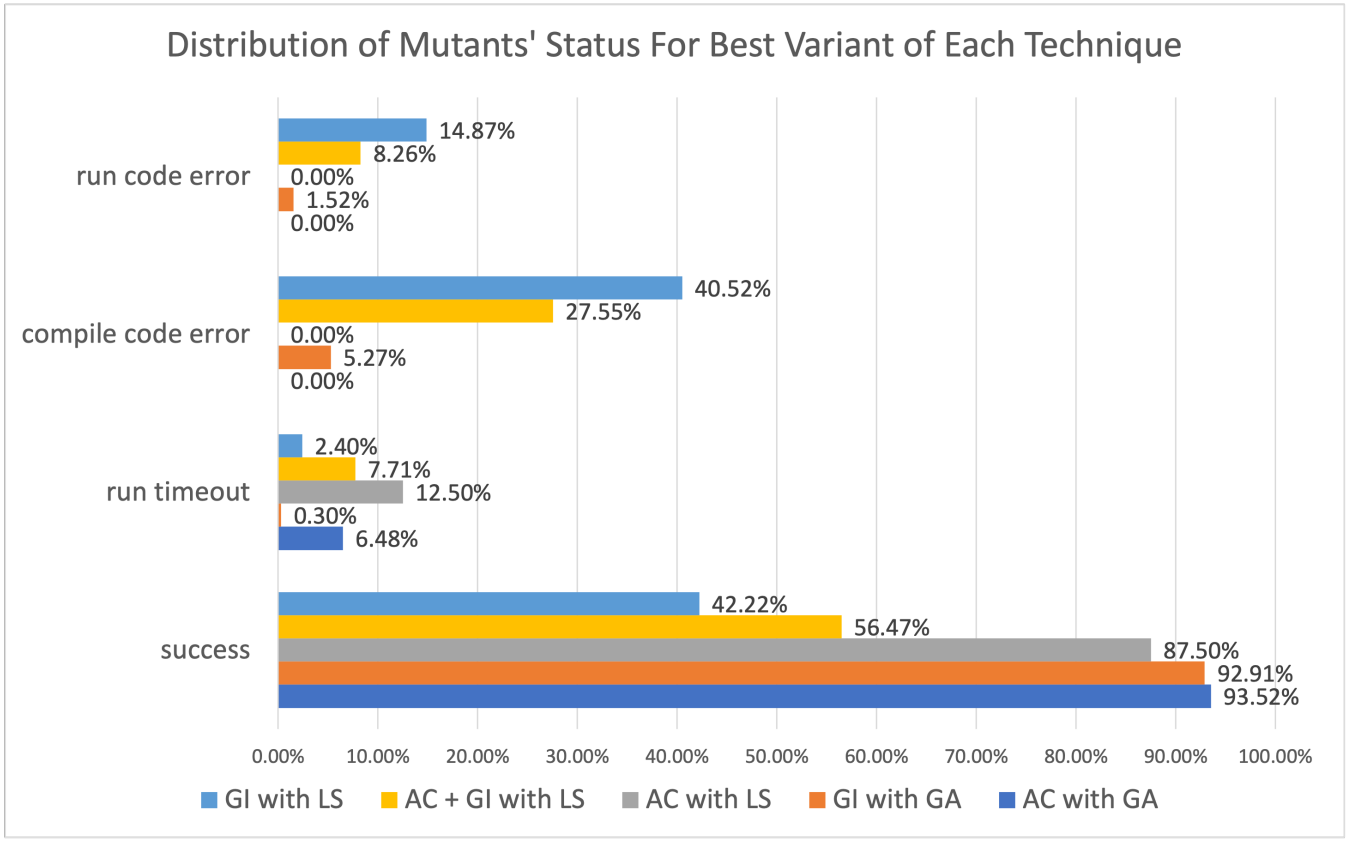

Figure 7: Distribution of every mutant's status generated during the training phase, for single best variant from each experiment. AC + GI with GA was not displayed here because no variants generalise to test instances.