Projekt 1
Rozwiązywanie nonogramów

Sandra Leman

Kwiecień 2022

Spis treści

1 Wstęp

Nonogram (zwany też obrazkiem logicznym) to rodzaj zagadki, w której należy zamalować niektóre kratki na czarno tak, by powstał z nich obrazek. By odgadnąć, które kratki należy zamalować, należy odszyfrować informacje liczbowe przy każdym wierszu i kolumnie kratek obrazka. Przykładowo, jeśli przy wierszu stoi "2 3 1", to znaczy, że w danym wierszu jest ciąg dwóch zamalowanych kratek, przerwa (przynajmniej jedna biała kratka), trzy zamalowane kratki, przerwa, jedna zamalowana kratka. Umiejscowienie zamalowanych ciągów nie jest wskazane.

1.1 Przykład

Poniżej przedstawiony jest nierozwiązyny przykład wielkości 15x15.

								1	1						1	1			
								2	1	2	2		2	2	1	2			
						3		4	6	1	1	7	1	1	6	4		3	
					5	5	1	3	2	3	3	1	3	3	2	3	1	5	5
					5	3	1	1	1	3	3	3	3	3	1	1	1	3	5
			5	5															
		3	5	3															
		2	9	2															
1	2	1	2	1															
1	1	7	1	1															
		4	1	4															
		4	1	4															
				13															
			6	6															
		2	7	2															
	1	2	2	1															
		1	11	1															
		2	9	2															
		3	5	3															
			5	5															

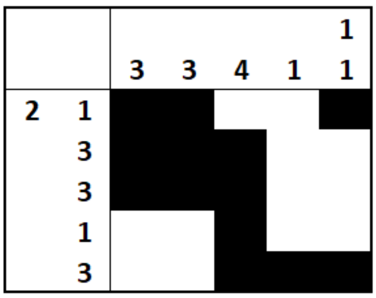
Istotne informacje zapisane są nad pustą planszą oraz po jej lewej stronie. Można te 2 elementy zapisać jako tablicę zawierającą kolejne tablice z liczbami reprezentujące daną kolumnę/wiersz. Powyższy przykład zapisany w ten sposób prezentuje się następująco:

```
[[
    [5, 5], [3, 5, 3], [1, 1], [1, 2, 4, 3, 1], [1, 1, 6, 2, 1],
    [2, 1, 3, 3], [2, 1, 3, 3], [7, 1, 3], [2, 1, 3, 3], [2, 1, 3, 3],
    [1, 1, 6, 2, 1], [1, 2, 4, 3, 1], [1, 1], [3, 5, 3], [5, 5]
], [
    [5, 5], [3, 5, 3], [2, 9, 2], [1, 2, 1, 2, 1], [1, 1, 7, 1, 1],
    [4, 1, 4], [4, 1, 4], [13], [6, 6], [2, 7, 2], [1, 2, 2, 1],
    [1, 11, 1], [2, 9, 2], [3, 5, 3], [5, 5]
]]
```

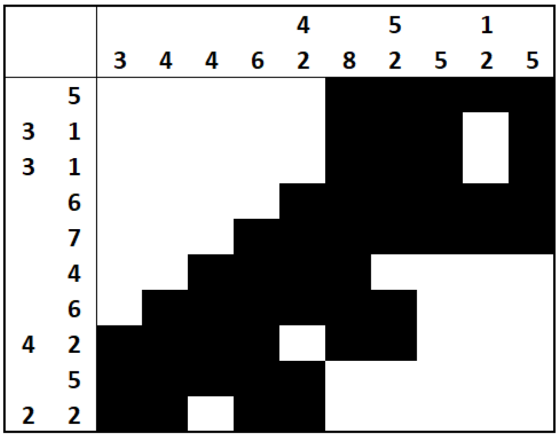
1.2 Instancje problemu

Rozpatrywnaymi problemami będą nonogramy w kształcie kwadratów o bokach od 5 do 15 kratek. Poniżej przedstowione są rozwiązania oraz sposób kodowania tych instancji.

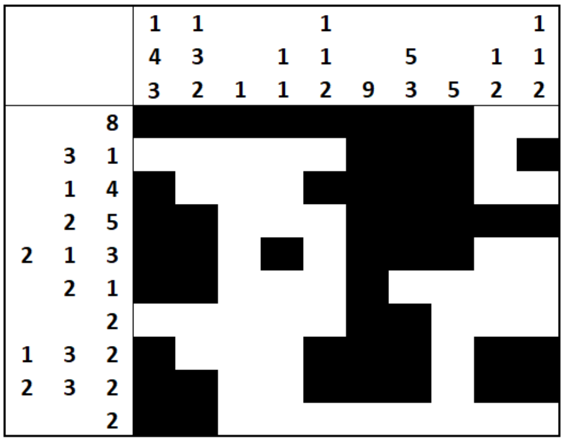
1. `[[[3], [3], [4], [1], [1, 1]], [[2, 1], [3], [3], [1], [3]]]`



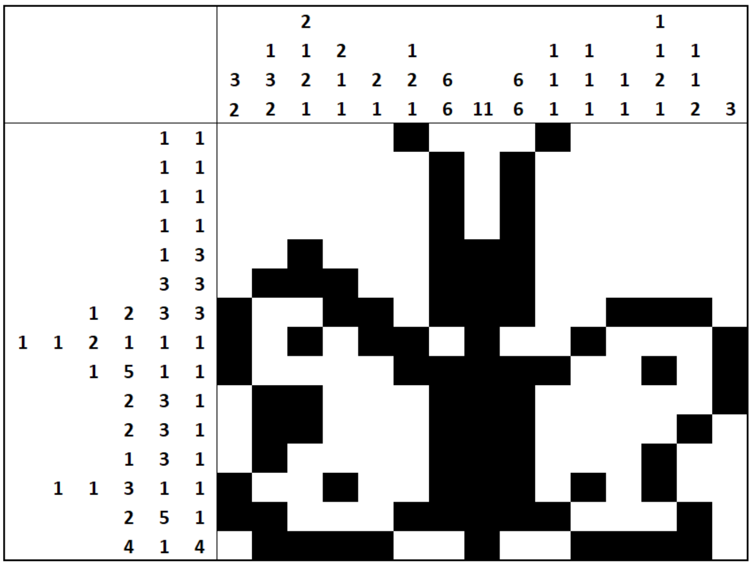
2.
```
[[
    [3], [4], [4], [6], [4, 2], [8], [5, 2], [5], [1, 2], [5]
], [
    [5], [3, 1], [3, 1], [6], [7], [4], [6], [4, 2], [5], [2, 2]
]]
```



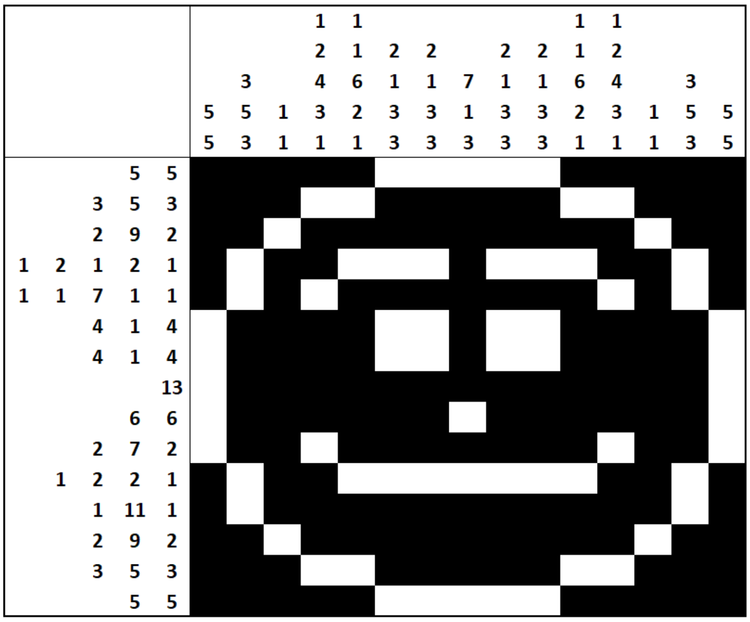
3.
```
[[
    [1, 4, 3], [1, 3, 2], [1], [1, 1], [1, 1, 2],
    [9], [5, 3], [5], [1, 2], [1, 1, 2]
], [
    [8], [3, 1], [1, 4], [2, 5], [2, 1, 3],
    [2, 1], [2], [1, 3, 2], [2, 3, 2], [2]
]]
```



4. [[
    [3, 2], [1, 3, 2], [2, 1, 2, 1], [2, 1, 1], [2, 1],
    [1, 2, 1], [6, 6], [11], [6, 6], [1, 1, 1],
    [1, 1, 1], [1, 1], [1, 1, 2, 1], [1, 1, 2], [3]
], [
    [1, 1], [1, 1], [1, 1], [1, 1], [1, 3], [3, 3],
    [1, 2, 3, 3], [1, 1, 2, 1, 1, 1], [1, 5, 1, 1], [2, 3, 1],
    [2, 3, 1], [1, 3, 1], [1, 1, 3, 1, 1], [2, 5, 1], [4, 1, 4]
]]



5. [[
    [5, 5], [3, 5, 3], [1, 1], [1, 2, 4, 3, 1], [1, 1, 6, 2, 1],
    [2, 1, 3, 3], [2, 1, 3, 3], [7, 1, 3], [2, 1, 3, 3], [2, 1, 3, 3],
    [1, 1, 6, 2, 1], [1, 2, 4, 3, 1], [1, 1], [3, 5, 3], [5, 5]
], [
    [5, 5], [3, 5, 3], [2, 9, 2], [1, 2, 1, 2, 1], [1, 1, 7, 1, 1],
    [4, 1, 4], [4, 1, 4], [13], [6, 6], [2, 7, 2],
    [1, 2, 2, 1], [1, 11, 1], [2, 9, 2], [3, 5, 3], [5, 5]
]]



2 Algorytm genetyczny

2.1 Ewolucja rozwiązań

Dla większości przedstawionych niżej rozwiązań parametry algorytmu genetycznego prezentują się następująco:

```
gene_space = [0, 1]            # 1 = zamalowane, 0 = niezamalowane
num_genes = height * width     # Wielkość planszy
sol_per_pop = 200
num_parents_mating = 100
num_generations = 1000
keep_parents = 5
parent_selection_type = "sss"
crossover_type = "single_point"
mutation_type = "random"
mutation_percent_genes = numpy.ceil(100 / num_genes)
```

Funkcje fitness dla konkretnych programów opisanę są poniżej:

1. Program sprawdza ile reguł jest, a ile nie jest spełnionych zarówno w wierszach jak i kolumnach.To podejście nie daje dobrych wyników, ponieważ nie uwzględnia jak bardzo otrzymane bloki zamalowanych kratek różnią się od oczekiwanego rezultatu.

2. Program sprawdza jak otrzymane bloki zamalowanych kratek różnią się od zadanych reguł. Działa lepiej od poprzednika, jednak różnice

między dopasowaniami nadal są zbyt małe, aby zawsze otrzymywać dobre wyniki.

3. Program oprócz dopasowania samych bloków sprawdza również różnicę w ilości bloków oraz ilości zamalowanych kratek. Takie porównanie daje dosyć dobre dopasowanie, jednak nadal ma problemy z większymi instancjami problemu.

4. Program obiera inną strategię jeśli chodzi o wybór genów:

```
gene_space = range(width)
num_genes = len(row_blocks)
```

   Strategia ta opiera się na założeniu, że bloki zamalowanych kratek są z góry ustalone, a gen określa ich miejsce w danym rzędzie. Takie rozwiązanie jest następnie przekształcane na tablicę reprezentującą całą planszę, a wartość fitness wyliczana jest jak w programie poprzednim. Możemy dostrzec znaczną poprawę w prędkości rozwiązywania problemów.

5. Program jest inną werją programu poprzedniego różniącą się tym, że nie wstawia bloków wierszami, tylko kolumnami:

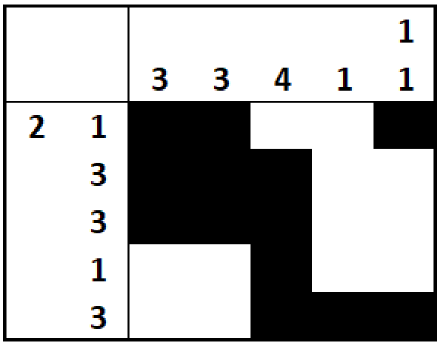
```
gene_space = range(height)
num_genes = len(column_blocks)
```

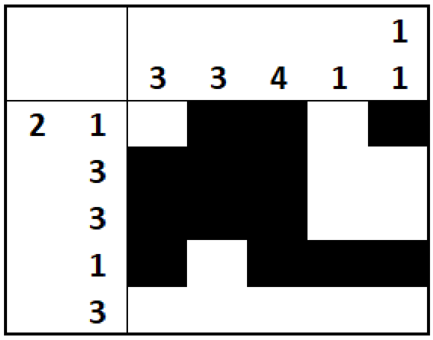
2.2 Pomiary czasu i poprawności

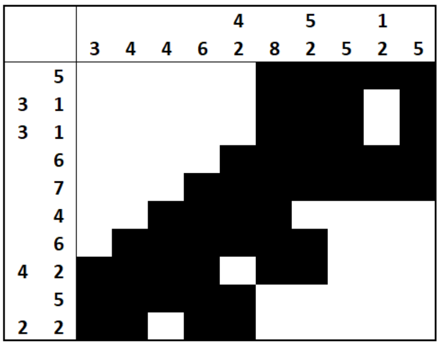
Program 1

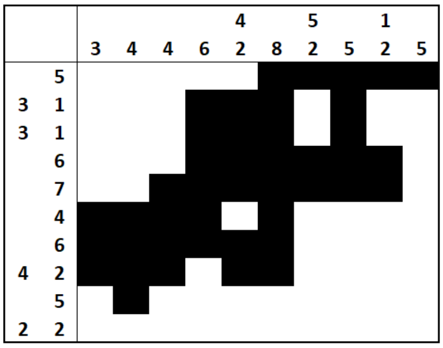
Instancja problemu	Średni czas s	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość fitness najlepszego dopasowania (na 10 prób)
2.1.1	14.37	0%	-2
2.1.2	29.05	0%	-7
2.1.3	29.66	0%	-8
2.1.4	52.12	0%	-16
2.1.5	52.62	0%	-26

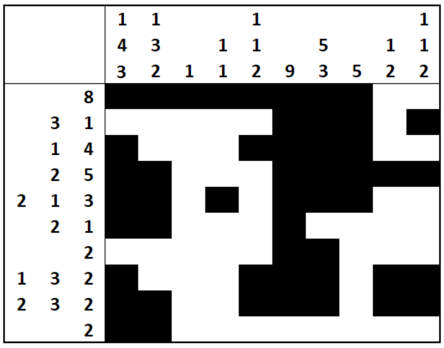
Jak widać program nie znalazł prawidłowego rozwiązania dla żadnej rozpatrywanej instancji problemu. Poniżej przedstawione jest porównanie najlepszych rozwiązań z tymi prawidłowymi:

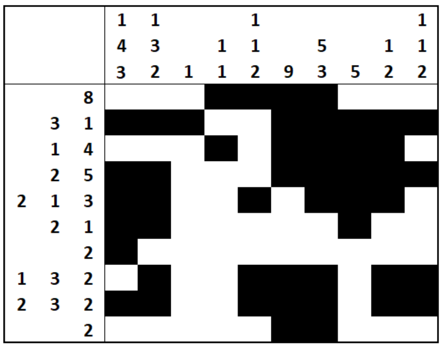

1
3 3 4 1 1
2 1
3
3
1
3

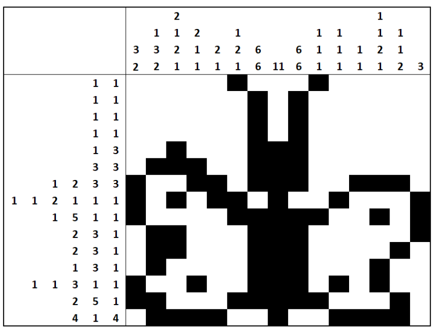

1
3 3 4 1 1
2 1
3
3
1
3

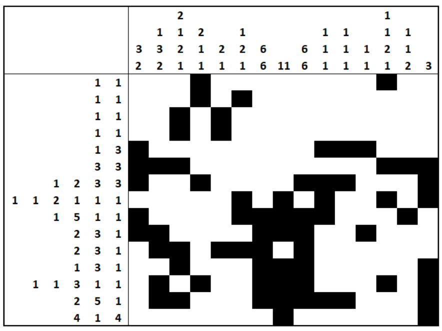

4 5 1
3 4 4 6 2 8 2 5 2 5
5
3 1
3 1
6
7
4
6
4 2
5
2 2

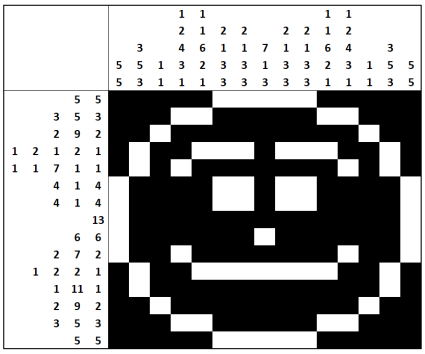

4 5 1
3 4 4 6 2 8 2 5 2 5
5
3 1
3 1
6
7
4
6
4 2
5
2 2

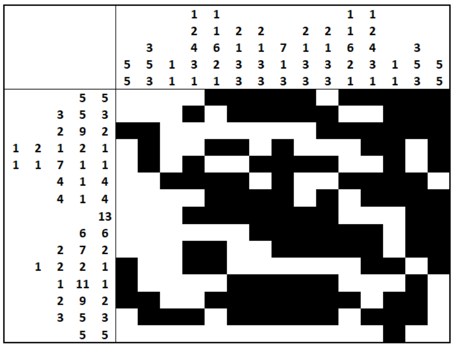

1 1 1 1
4 3 1 1 5 1 1
3 2 1 1 2 9 3 5 2 2
8
3 1
1 4
2 5
2 1 3
2 1
2
1 3 2
2 3 2
2


1 1 1 1
4 3 1 1 5 1 1
3 2 1 1 2 9 3 5 2 2
8
3 1
1 4
2 5
2 1 3
2 1
2
1 3 2
2 3 2
2





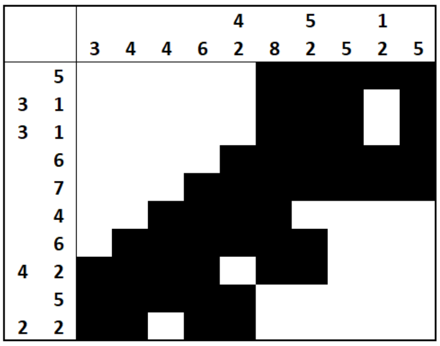


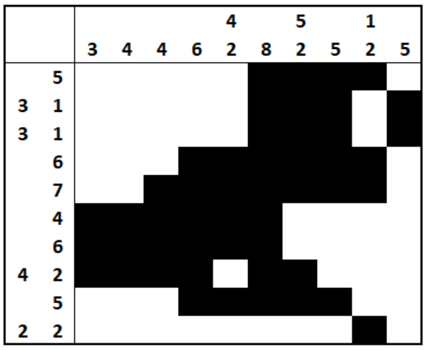


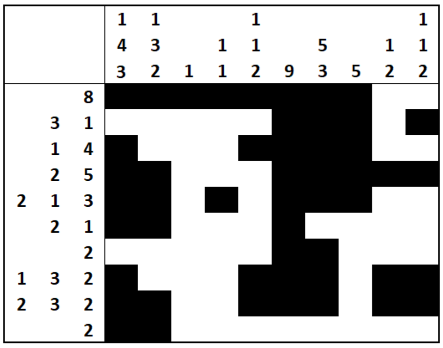
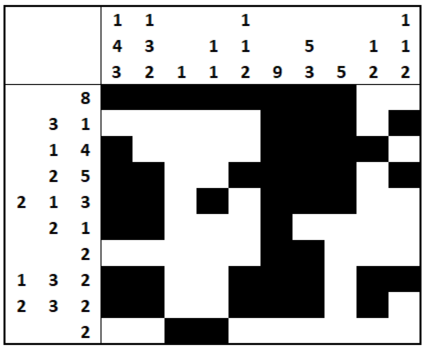
Program 2

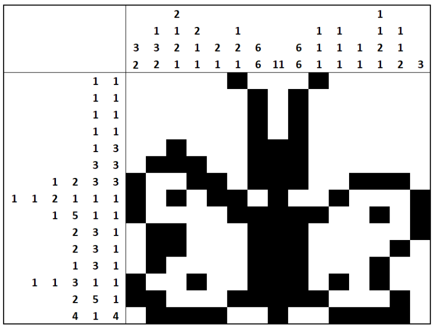
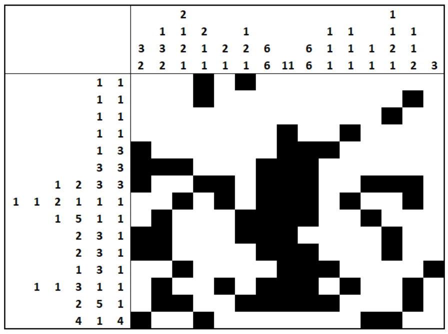
Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość fitness najlepszego dopasowania (na 10 prób)
2.1.1	2.53	100%	0
2.1.2	34.12	0%	-16
2.1.3	38.16	0%	-11
2.1.4	67.53	0%	-27
2.1.5	71.07	0%	-66

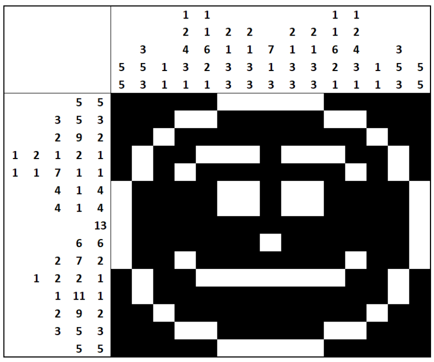
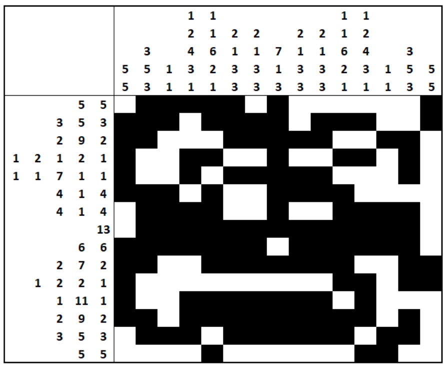
Program dobrze radzi sobie z małymi instancjami problemu, jednak już średnie instancje sprawiają mu problem. Poniżej przedstawione jest porównanie najlepszych rozwiązań z tymi prawidłowymi:





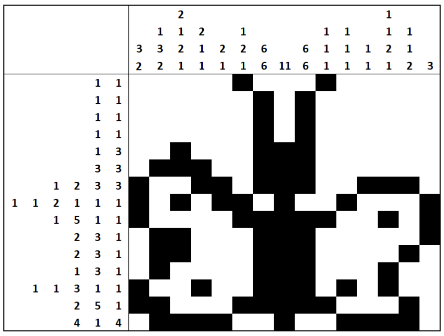
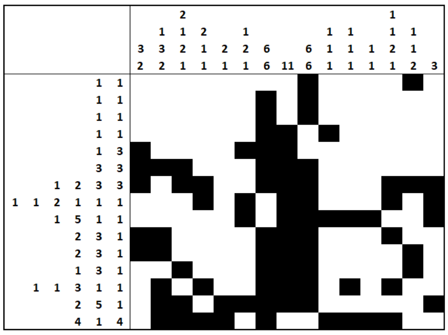



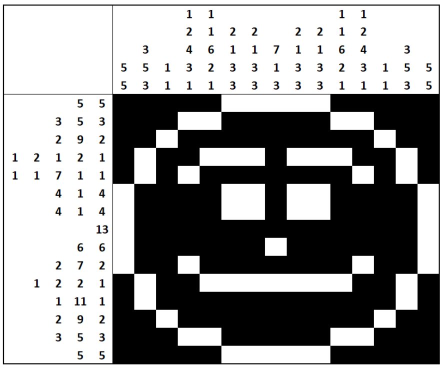
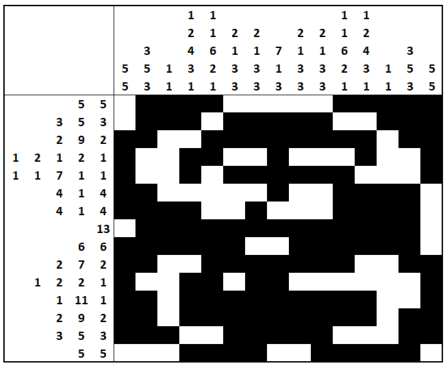



Program 3

Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość fitness najlepszego dopasowania (na 10 prób)
2.1.1	0.33	100%	0
2.1.2	34.45	40%	0
2.1.3	24.70	70%	0
2.1.4	104.94	0%	-23
2.1.5	104.85	0%	-66

Program potrafi znaleźć rozwiązanie dla instancji wielkości 10x10, jednak nie w każdym przypadku. Poniżej przedstawione jest porównanie najlepszych rozwiązań z tymi prawidłowymi dla instancji wielkości 15x15:

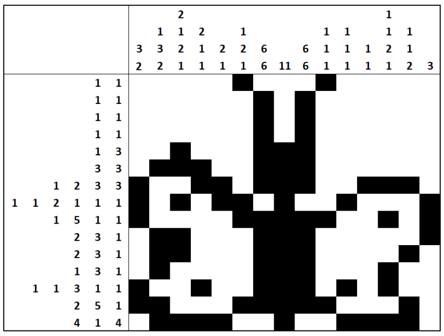
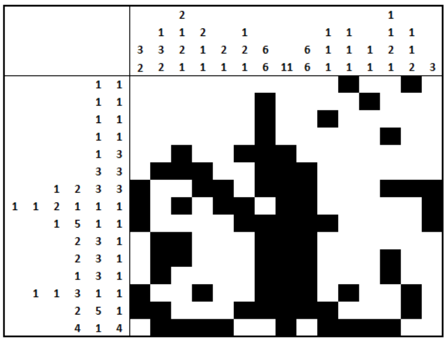



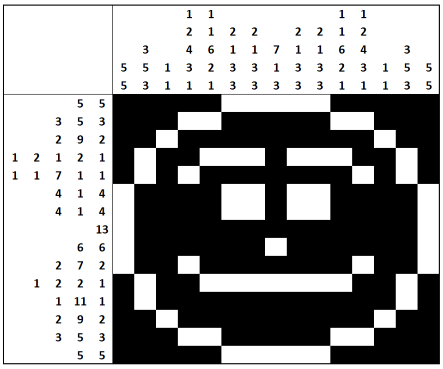
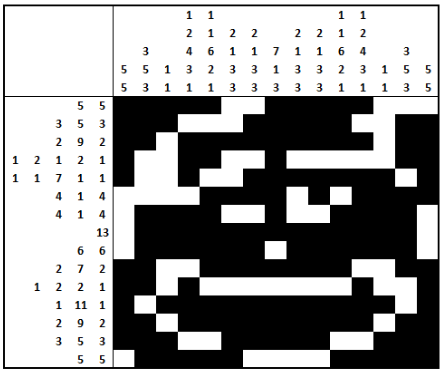



Program 4

Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość fitness najlepszego dopasowania (na 10 prób)
2.1.1	0.13	100%	0
2.1.2	20.32	70%	0
2.1.3	10.65	100%	0
2.1.4	120.69	0%	-15
2.1.5	119.26	0%	-61

Program znacznie lepiej radzi sobie z małymi i średnimi instancjami problemu, chociaż nadal nie jest w stanie znaleźć rozwiązania dla wielkości 15x15. Poniżej przedstawione jest porównanie najlepszych rozwiązań z tymi prawidłowymi dla tych instancji. Możemy z niego wnioskować, że dopasowanie jest znacznie lepsze od wcześniejszych wersji.

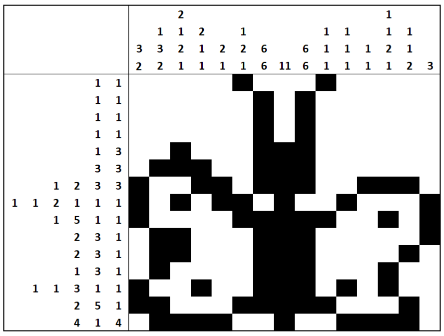
 

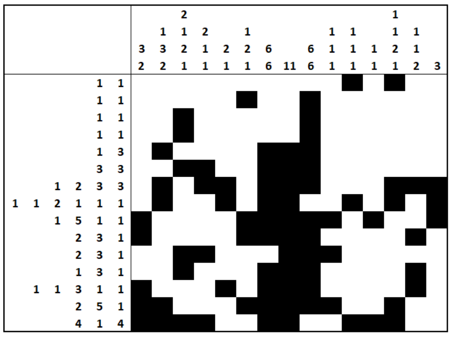
 

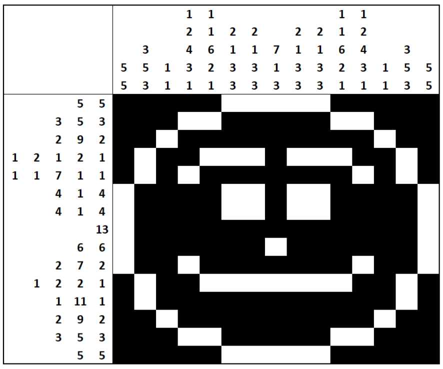
Program 5

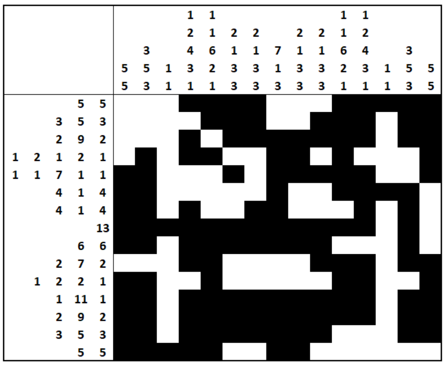
Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość fitness najlepszego dopasowania (na 10 prób)
2.1.1	0.12	100%	0
2.1.2	36.96	40%	0
2.1.3	31.41	100%	0
2.1.4	119.08	0%	-10
2.1.5	122.41	0%	-96

Program działa gorzej od poprzednika, mimo że użyto podobnej strategii. Poniżej przedstawione jest porównanie najlepszych rozwiązań z tymi prawidłowymi:





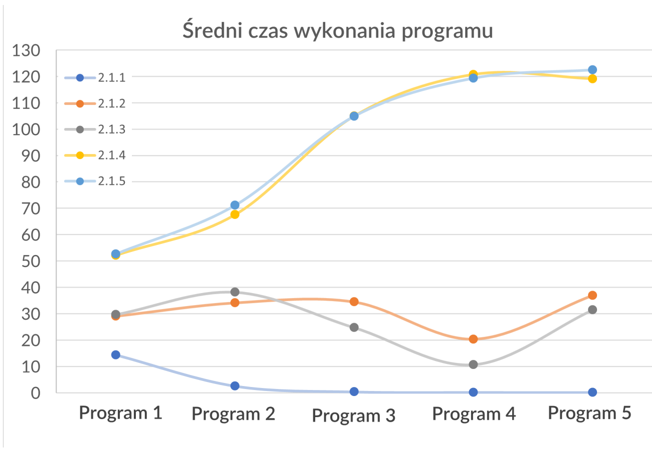


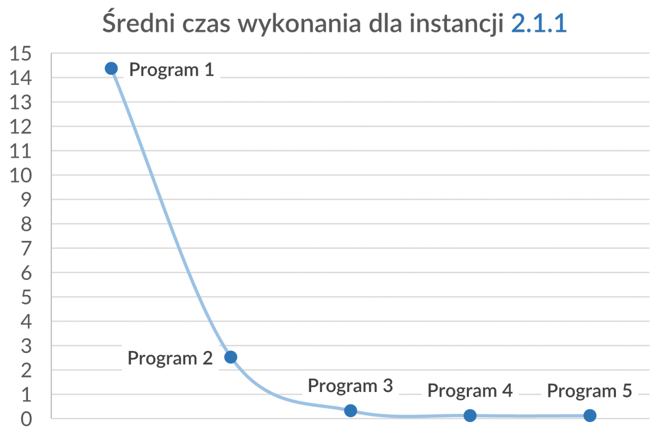


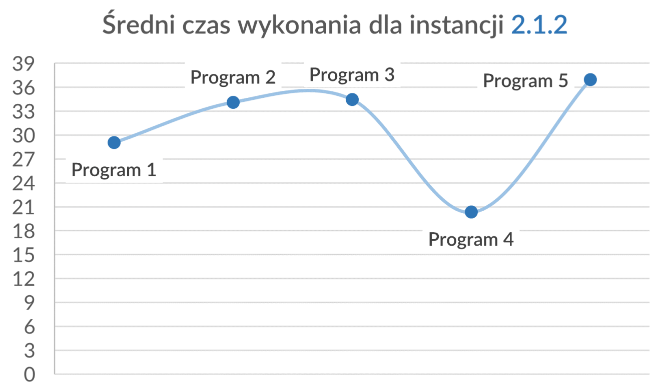
2.3 Porównanie

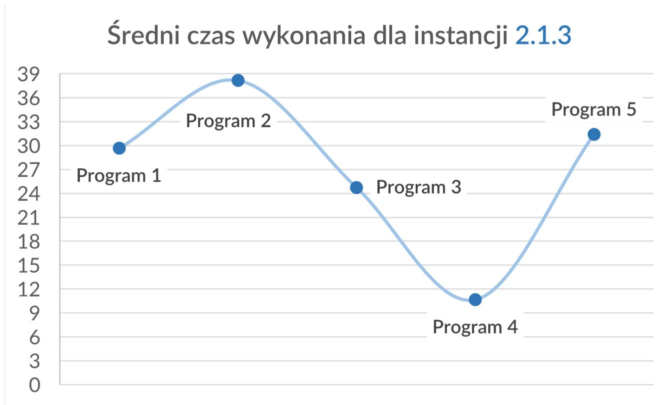
Poniżej przedstawione są przykładowe wyniki dla pojedynczego eksperymentu każdego z programów i instancji oraz wykresy pokazujące interesujące zależności między średnim czasem wykonania każdego programu dla wszystkich instancji problemu.

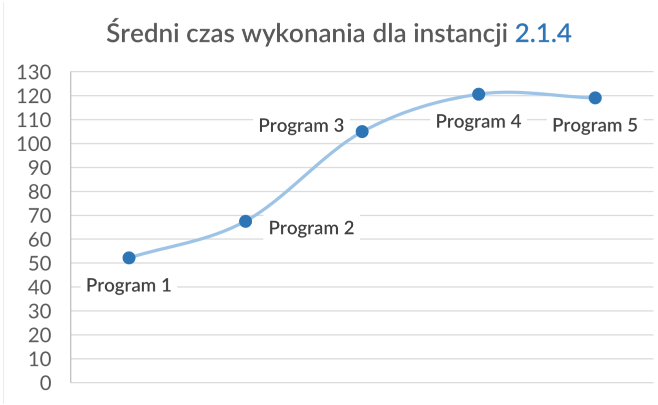
2.3.1 Porównanie czasu

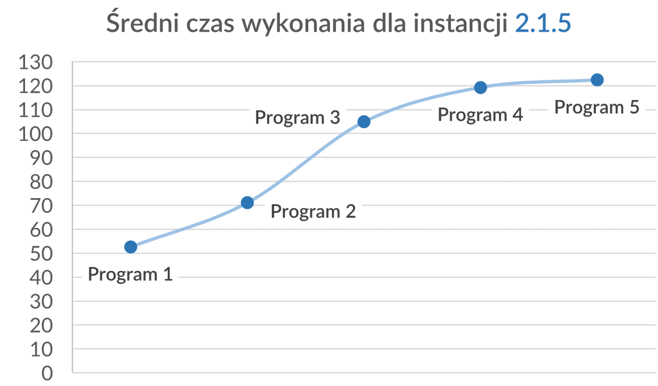



Średni czas wykonania dla instancji 2.1.1
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Program 1
Program 2
Program 3
Program 4
Program 5


Średni czas wykonania dla instancji 2.1.2
39
36
33
30
27
24
21
18
15
12
9
6
3
0
Program 1
Program 2
Program 3
Program 4
Program 5


Średni czas wykonania dla instancji 2.1.3
39
36
33
30
27
24
21
18
15
12
9
6
3
0
Program 1
Program 2
Program 3
Program 4
Program 5



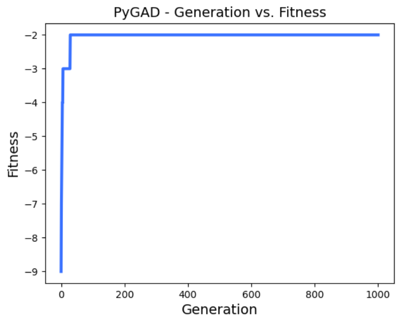


Od razu widać, że kształty przy instancjach tych samych rozmiarów są podobne. Możemy również zauważyć tendencje wzrostowe dla kolejnych programów gdy miały problem ze znalezieniem rozwiązania i tendencje malejące gdy rozwiązanie znajdowały. Wyjątek może stanowić program 5 który momentami działał gorzej od poprzednika. Należy jednak pamiętać, że duży wpływ na rozwiązanie ma wybór instancji problemu.

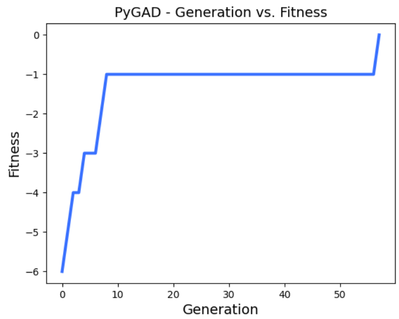
2.3.2 Przykładowe wyniki

Przykładowe wyniki eksperymentu dla instancji 2.1.1

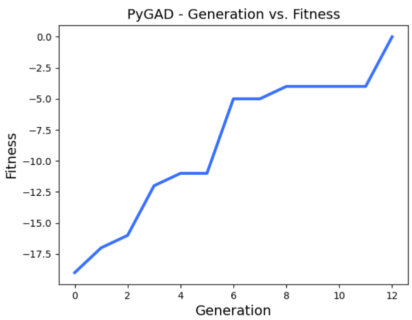
Program 1



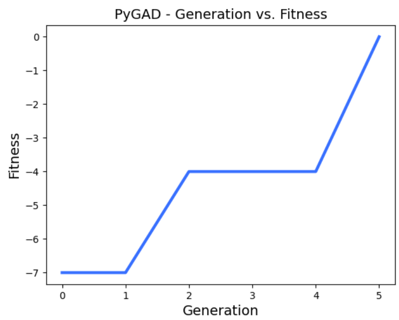
Program 2



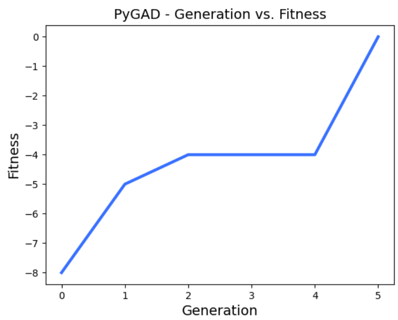
Program 3



Program 4

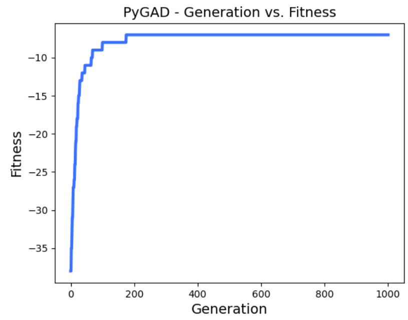


Program 5

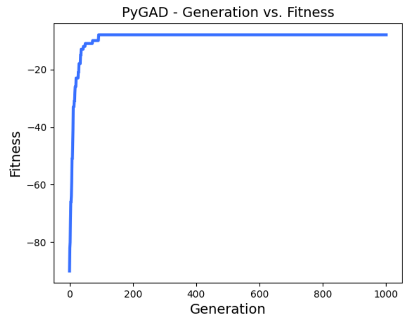


Przykładowe wyniki eksperymentu dla instancji 2.1.2

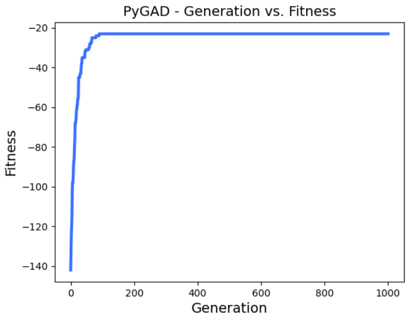
Program 1



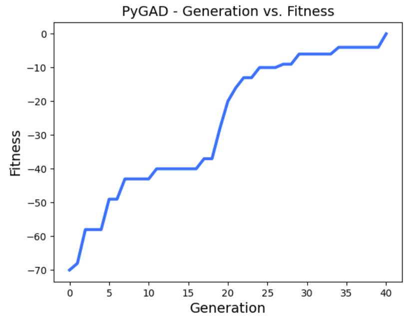
Program 2



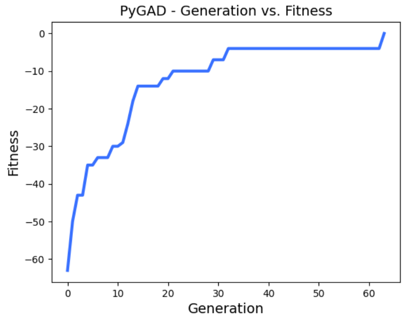
Program 3



Program 4

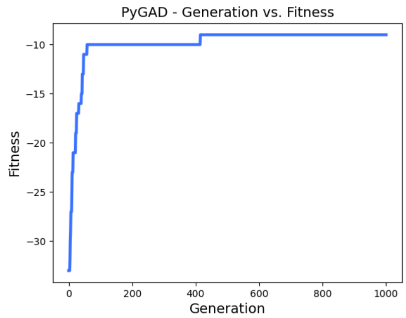


Program 5

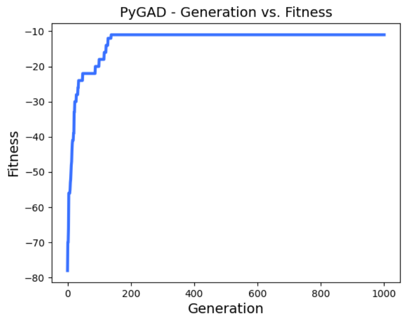


Przykładowe wyniki eksperymentu dla instancji 2.1.3

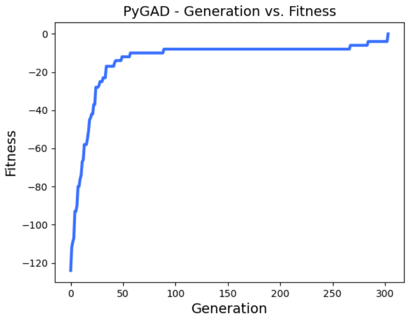
Program 1



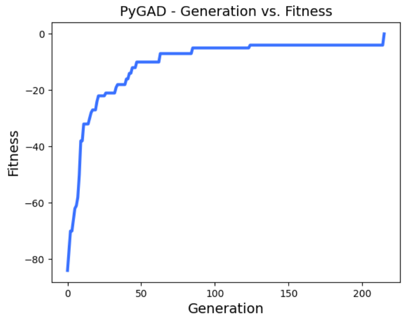
Program 2



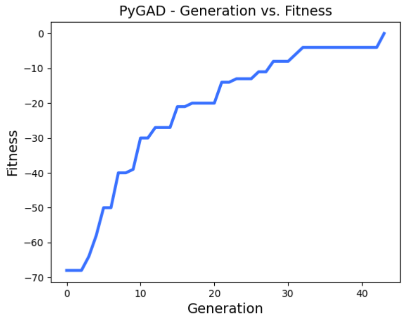
Program 3



Program 4

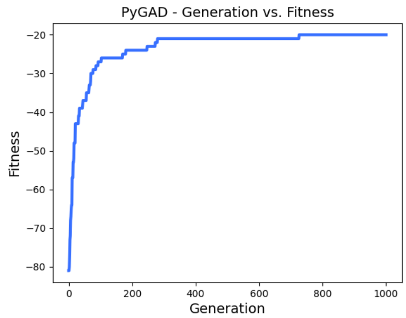


Program 5

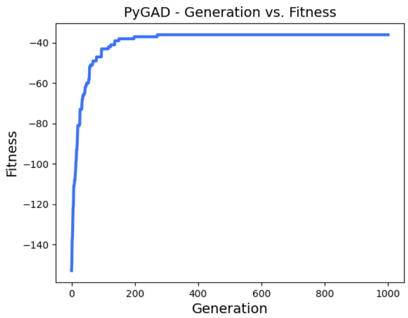


Przykładowe wyniki eksperymentu dla instancji 2.1.4

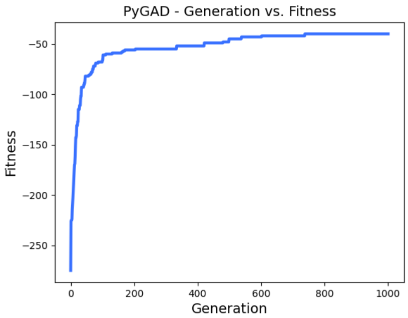
Program 1



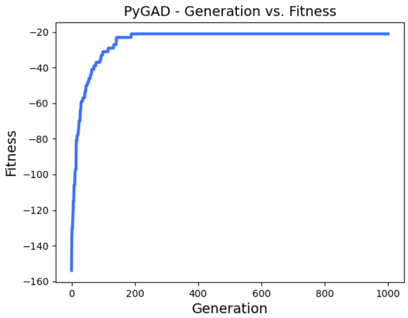
Program 2



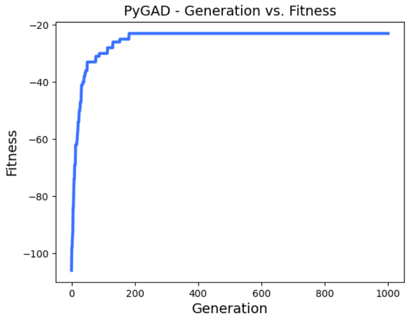
Program 3



Program 4

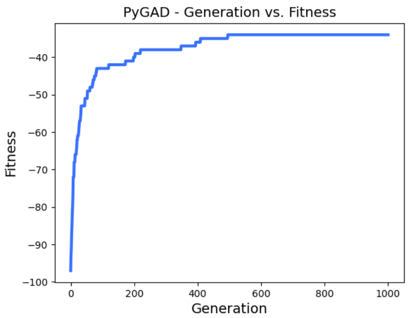


Program 5

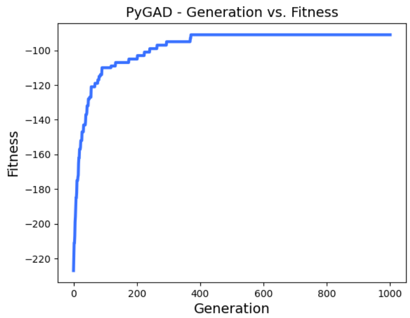


Przykładowe wyniki eksperymentu dla instancji 2.1.5

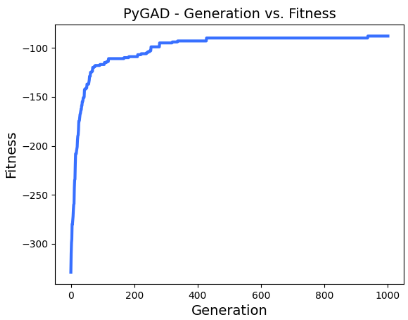
Program 1



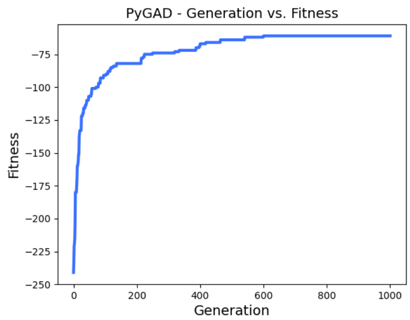
Program 2



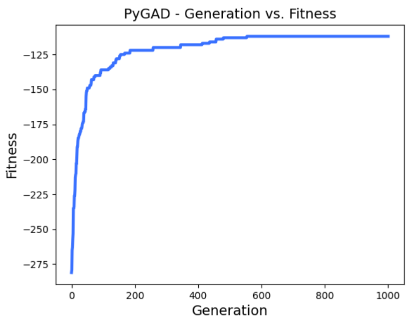
Program 3



Program 4



Program 5



3 Optymalizacja przez rój

Dla pierwszych 3 programów można spróbować skorzystać z optymalizacji przez rój (BinaryPSO). Wówczas wartości funkcji fitness podnoszone są do kwadratu, aby zmienić znaki oraz zwiększyć różnice miedzy wynikami. Taka wartość obliczana jest dla każdego punktu roju. Można zauważyć sporą poprawę w prędkości wykonania, jednak poprawność rozwiązań jest zdecydowanie gorsza. Parametry użyte w programach to:

```
options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9, 'k': 10, 'p': 1}
n_particles = 50
dimensions = width * height
iters = 1000
verbose = True
```

3.1 Pomiary czasu i poprawności

Program 1'

Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość funkcji najlepszego dopasowania (na 10 prób)	Pierwiastek z najlepszej wartości funkcji
2.1.1	3.63	0%	4	2
2.1.2	11.38	0%	900	30
2.1.3	11.31	0%	729	27
2.1.4	23.36	0%	4356	66
2.1.5	22.88	0%	6561	81

Program 2'

Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość funkcji najlepszego dopasowania (na 10 prób)	Pierwiastek z najlepszej wartości funkcji
2.1.1	2.30	20%	0	0
2.1.2	7.40	0%	2601	51
2.1.3	8.09	0%	1296	36
2.1.4	15.70	0%	9801	99
2.1.5	15.92	0%	24964	158

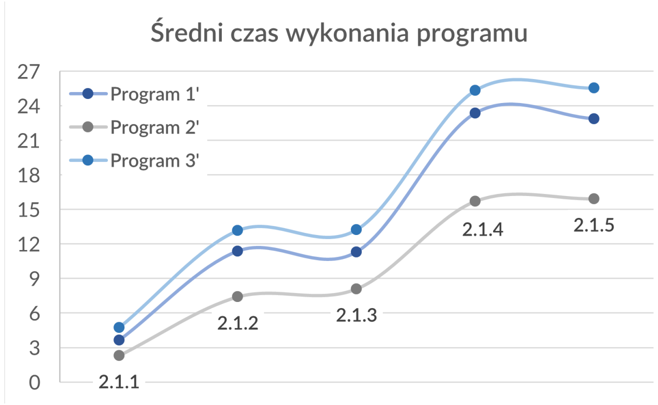
Program 3'

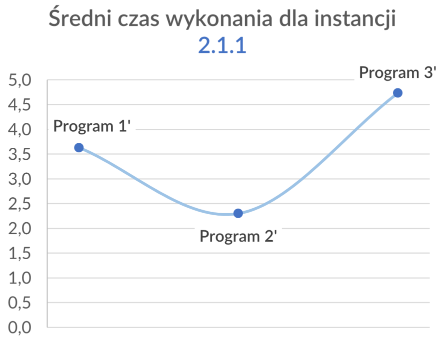
Instancja problemu	Średni czas [s]	Procent dobrze znalezionych rozwiązań (na 10 prób)	Wartość funkcji najlepszego dopasowania (na 10 prób)	Pierwiastek z najlepszej wartości funkcji
2.1.1	4.73	70%	0	0
2.1.2	13.16	0%	8100	90
2.1.3	13.23	0%	4096	64
2.1.4	25.32	0%	31329	117
2.1.5	25.53	0%	68644	262

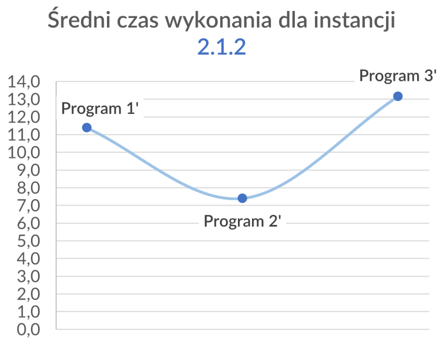
3.2 Porównanie

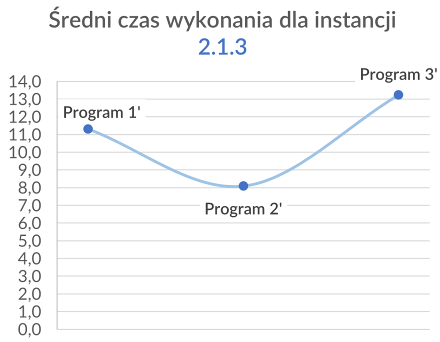
Poniżej przedstawione są przykładowe wyniki dla pojedynczego eksperymentu każdego z programów i instancji oraz wykresy pokazujące interesujące zależności między średnim czasem wykonania każdego programu dla wszystkich instancji problemu.

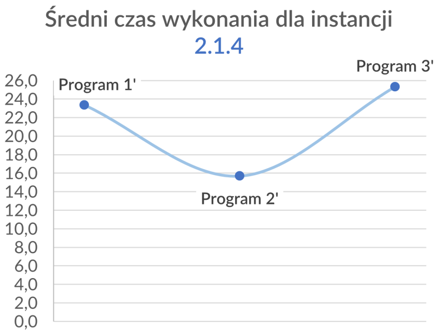
3.2.1 Porównanie czasu

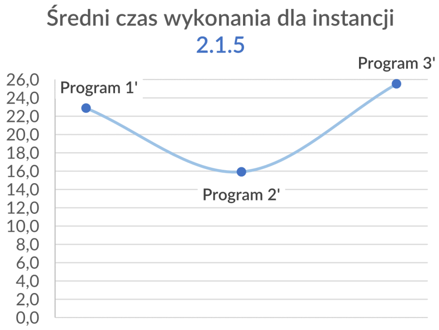










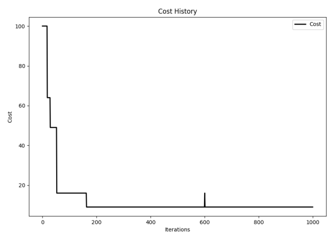


Widać, że w tym wypadku średni czas wykonania programów zmienia się analogicznie dla wszytskich wielkości problemu. Może to być spowodowane faktem, że optymalizacja przez rój wykonuje wszytskie iteracje niezależnie od tego, czy znalała rozwiązanie, czy nie oraz tym, że programy mają problem z rozwiązaniem większości instancji branych pod uwagę.

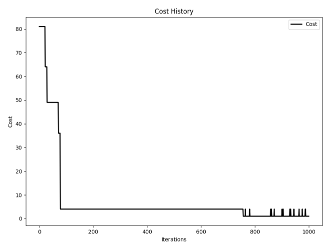
3.2.2 Przykładowe wyniki

Przykładowe wyniki eksperymentu dla instancji 2.1.1

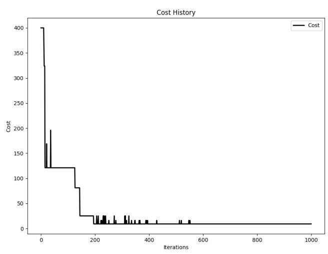
Program 1’



Program 2’

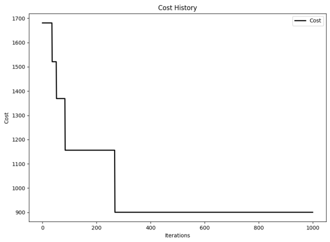


Program 3’

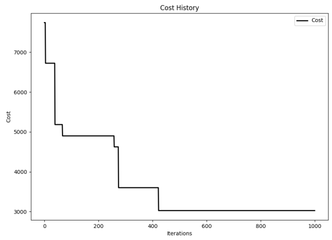


Przykładowe wyniki eksperymentu dla instancji 2.1.2

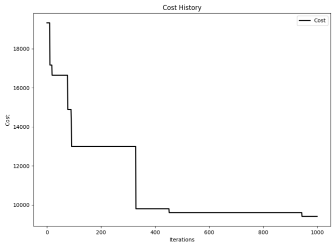
Program 1'



Program 2'

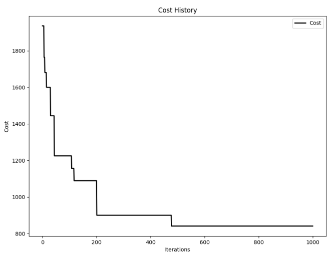


Program 3'

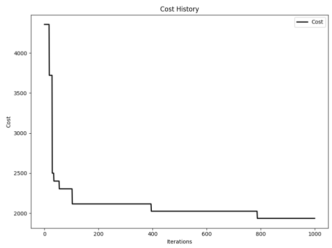


Przykładowe wyniki eksperymentu dla instancji 2.1.3

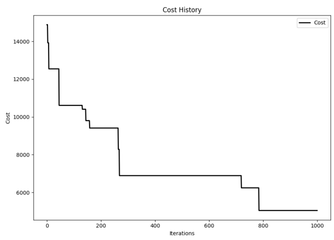
Program 1'



Program 2'

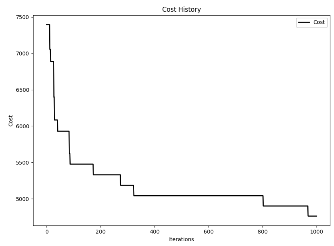


Program 3'

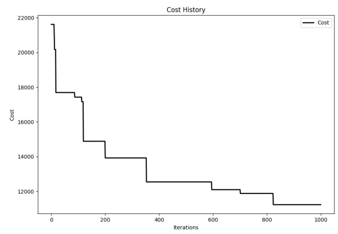


Przykładowe wyniki eksperymentu dla instancji 2.1.4

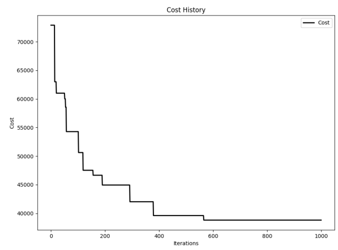
Program 1'



Program 2'

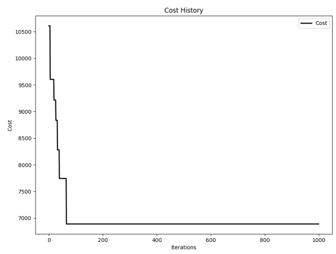


Program 3'

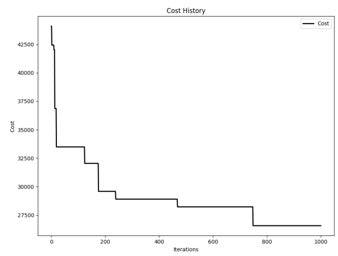


Przykładowe wyniki eksperymentu dla instancji 2.1.5

Program 1'



Program 2'



Program 3'