

Głębokie uczenie i Inteligencja obliczeniowa
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii
Biomedycznej

**Zastosowanie algorytmu genetycznego oraz algorytmu
rojowego w problemie rozwiązywania sudoku**

Skład zespołu:

Julia Krysiak
Sylwia Michalska
Artur Mazurkiewicz
Konrad Prokop
Kaia Rupniak
Filip Chrapla
Jan Rusek
Jakub Iskrzycki

8 kwietnia 2024

Spis treści

1	Cel projektu	2
2	Badany problem	2
3	Propozycja rozwiązania	2
3.1	Algorytmy – adaptacja	2
3.2	Algorytmy – pseudokody	4
4	Aplikacja	6
5	Eksperymenty	8
6	Podsumowanie i wnioski	10
7	Udział poszczególnych osób w dany etap projektu	11
8	Spis literatury	12

1 Cel projektu

Celem projektu było zapoznanie się z algorytmami genetycznymi oraz algorytmami inspirowanymi biologicznie. Pozyskaną wiedzę sprawdzono implementując wybrane algorytmy dla problemu rozwiązywania sudoku.

2 Badany problem

Problem przedstawiony w projekcie polega na napisaniu i zaimplementowaniu dwóch algorytmów w języku Python, które pozwolą rozwiązać zagadkę sudoku o trzech różnych stopniach trudności. Rozwiązywanie sudoku polega na uzupełnieniu macierzy cyfr, przy danej liczbie początkowych cyfr, tak żeby dana cyfra nigdy nie powtarzała się w danym wierszu, kolumnie lub jednej z mniejszych macierzy, na które podzielona jest zagadka sudoku (dla sudoku o wymiarach 9x9 istnieje 9 małych macierzy 3x3). Do tego projektu wykorzystano sudoku o wymiarze 9x9, przy czym stopień trudności zagadki został określony na podstawie liczby początkowych cyfr i zaawansowania technik potrzebnych do rozwiązywania sudoku, gdyby rozwiązującym był człowiek.

W literaturze można znaleźć wiele metod rozwiązywania sudoku przy użyciu programów komputerowych. Jedną z najbardziej popularnych jest **backtracking**, czyli metoda typu *brute force* polegająca na sprawdzeniu wszystkich możliwych kombinacji. W tym wypadku losowane są kolejne liczby dla każdej kolejnej komórki, a jeśli powtórzona zostanie jakaś liczba, to następuje cofnięcie się do poprzedniej komórki i testowanie innej kombinacji aż do rozwiązania sudoku.

Inną grupą metod pozwalających na rozwiązywanie sudoku są stochastyczne algorytmy optymalizacji, których działanie polega na losowym przypisaniu liczb do pustych komórek w zdefiniowanej siatce, następnie policzenie ilości błędów, a na końcu zamiana liczb w komórkach, dopóki liczba błędów nie spadnie do zera. Przykładowymi sposobami realizacji tego procesu są: **algorytm genetyczny**, **przeszukiwanie tabu** i algorytm **Simulated Annealing**.

Geometryczna optymalizacja rojem cząstek (GPSO) to technika rozwiązywania sudoku, która oparta jest na tradycyjnej metodzie PSO. W przypadku GPSO dysponuje się rojem cząstek, gdzie cząstka oznacza możliwe rozwiązanie sudoku. Aby znaleźć najlepsze możliwe rozwiązanie, wykorzystuje się prawdopodobieństwa mutacji, krzyżowanie i określanie wag danych pozycji.

Rozwiązywanie sudoku za pomocą algorytmów implementowanych w programach komputerowych jest wciąż rozwijającą się dziedziną i powyższy opis badanego problemu nie wyczerpuje wszystkich podejść do uzupełniania macierzy sudoku. Ustalono, że w projekcie przedstawione zostaną rozwiązania oparte na algorytmie genetycznym i geometrycznej optymalizacji rojem cząstek.

3 Propozycja rozwiązania

3.1 Algorytmy – adaptacja

Algorytm genetyczny

Główną ideą algorytmu genetycznego do rozwiązywania sudoku jest wybieranie rodziców i ich potomków z obecnej populacji (reprezentująca przypadkowe plansze sudoku), na podstawie której generowana jest nowa populacja bliższa prawidłowemu rozwiązaniu zagadki. Dalej opisano najważniejsze operacje algorytmu genetycznego.

Populacja początkowa

W algorytmie generowana jest populacja początkowa o zadanej wielkości. Użytkownik podaje planszę sudoku, w której część pól jest wypełniona liczbami. Algorytm na jej podstawie uzupełnia pozostałe elementy planszy przypadkowymi liczbami, ale w taki sposób, by w każdej siatce 3x3 nie znajdowały się duplikaty. Następnie dla każdego wygenerowanego rozwiązania obliczane jest dopasowanie. Dopasowanie zdefiniowane jest jako suma liczby duplikatów w każdym wierszu i kolumnie. W przypadku gdy dopasowanie jest zerowe (czyli nie ma duplikatów) uzyskano rozwiązanie prawidłowe.

Krzyżowanie

Pierwszym etapem operacji krzyżowania jest wybranie pary rodziców o największym dopasowaniu. Na ich podstawie tworzeni są potomkowie w następujący sposób: krzyżowanie zachodzi poprzez łączenie odpowiednich siatek 3x3 sudoku – w ten sposób powstaje nowe rozwiązanie będące połączeniem rozwiązań rodziców; punkt krzyżowania wybierany jest losowo (jest to liczba z zakresu 1-8). Następnie liczone jest ich

dopasowanie i rozwiązania dodawane są do populacji. Stosowanie operacji krzyżowania pozwala na stworzenie lepszych rozwiązań poprzez łączenie informacji z najlepszych dotychczasowych osobników populacji.

Mutacja

Mutacja polega na zamianie dwóch liczb w siatce 3x3 z zadaniem prawdopodobieństwem. Zamianie mogą być poddane tylko te liczby, które nie były podane w sudoku wejściowym. Operacja ta jest potrzebna, by rozszerzyć poszukiwania innych rozwiązań poprzez wprowadzenie elementu przypadkowości. W ten sposób odkrywa się nowe ewentualne rozwiązania, do których dojście mogłoby nie być możliwe wyłącznie za pomocą operacji krzyżowania.

Selekcja

Wybieranie populacji do kolejnej iteracji algorytmu nazywane jest selekcją. Nowa populacja tworzona jest poprzez połączenie części populacji rodziców (populacja początkowa), potomków (populacja po operacjach krzyżowania i mutacji) oraz dodanie do nich części przypadkowych rozwiązań. W przypadku tej implementacji algorytmu było to odpowiednio 20%, 50% i 30% całej populacji.

Warunek stopu

Warunkiem stopu zaimplementowanego algorytmu jest osiągnięcie zadanej maksymalnej liczby iteracji lub uzyskanie prawidłowego rozwiązania.

Algorytm rojowy - GPSO

Idea algorytmu GPSO opiera się na wykorzystaniu koncepcji roju cząstek, które zawarte są w przestrzeni rozwiązań i reprezentują potencjalne rozwiązania. W przeciwieństwie do klasycznego algorytmu PSO, geometryczny PSO nie bierze pod uwagę prędkości cząstek, a zamiast tego wykorzystuje operacje krzyżowania i mutacji do przemieszczania się cząstek w przestrzeni rozwiązań.

Rój początkowy

W algorytmie rój początkowy składa się z określonej przez parametr liczby cząstek. Każda cząstka, która jest dodawana w pętli do roju posiada wygenerowaną losowo pozycję startową, której podstawę stanowi początkowa plansza sudoku. Algorytm na jej podstawie uzupełnia pozostałe elementy planszy losowymi liczbami z zakresu 1-9, ale w taki sposób, aby w każdym wierszu nie występowały powtórzenia elementów.

Krzyżowanie

W GPSO krzyżowanie zachodzi pomiędzy trzema cząstkami. Na podstawie obecnej pozycji, najlepszej lokalnej pozycji oraz najlepszej globalnej pozycji oraz przy uwzględnieniu maski tworzone jest potomstwo. Przy implementacji krzyżowania zastosowane zostały dwa różne podejścia, w których na początku generowana jest maska:

- crossover1: generowana jest jedna maska, która określa, jaki wiersz planszy sudoku zostanie przekazany do potomka od którego z rodziców,
- crossover2: generowanych jest 9 masek, po jednej dla każdego z wierszy planszy sudoku. Każda maska określa, który element z danego wiersza zostanie przekazany do potomka od którego rodzica

Następnie, na podstawie maski łączone są odpowiednie części planszy sudoku od rodziców tworząc nową planszę - potomka:

- crossover1: nową planszę tworzy połączenie odpowiednich wierszy rodziców,
- crossover2: nowa plansza powstaje poprzez wybranie elementów od odpowiednich rodziców zgodnie z odpowiednią maską.

Zastosowanie operacji krzyżowania pozwala na wykorzystanie informacji genetyczną od różnych rodziców do generowania potomstwa, które może potencjalnie zawierać lepsze cechy od rodziców, co może wpłynąć na przyspieszenie procesu ewolucji w kierunku lepszych rozwiązań.

Mutacja

Mutacja polega na losowej zmianie dwóch elementów w wierszu planszy z określonym przez parametr `MUTATION_PROB` prawdopodobieństwem. Zamianie mogą być poddane tylko te liczby, które nie były podane

w sudoku wejściowym. Ma ona na celu wprowadzenie różnorodności w populacji cząstek oraz pomoc w uniknięciu osiągnięcia lokalnego minimum.

Warunek stopu

Waruniem stopu algorytmu jest osiągnięcie optymalnego dopasowania przez cząstkę realizowanego przy pomocy funkcji `converge` sprawdzającego, czy cząstka osiągnęła wartość fitness równą 243 lub wykonanie określonej przez parameter `N_ITERATIONS` liczby iteracji. Wartość fitness jest obliczana poprzez zsumowanie: liczby unikalnych elementów w każdym wierszu, liczby unikalnych elementów w każdej kolumnie i liczby unikalnych elementów w każdym polu. W przypadku GPSO dożymy do osiągnięcia maksymalnej wartości fitness 243.

3.2 Algorytmy – pseudokody

Algorytm genetyczny

Pseudokod do głównej funkcji algorytmu genetycznego przedstawiono poniżej.

Algorithm 1: Algorytm genetyczny dla sudoku

Function *GA***Data:** plansza sudoku**Result:** najlepsze znalezione rozwiązanieZainicjalizuj parametry algorytmu: `N_POPULATION`, `MAX_ITERATIONS`,
`MUTATION_PROB`, `CROSSOVER_PROB`;

Inicjalizacja populacji losowymi rozwiązaniami sudoku;

while *nie zostało znalezione rozwiązanie i nie osiągnięto MAX_ITERATIONS* **do**

Przeprowadź operację krzyżowania na populacji;

Przeprowadź operację mutacji na wynikowej populacji z krzyżowania;

Wybierz najlepsze rozwiązania do kolejnej iteracji;

end**return** *najlepsze znalezione rozwiązanie*;**end**

Dodatkowo zdefiniowano:

- klasę *Sudoku* zawierającą:
 - planszę sudoku (grid)
 - poziom trudności
 - metodę do wypisywania planszy
- klasę *Solution* zawierającą:
 - planszę sudoku (grid)
 - wartość dopasowania (fitness)
- funkcje pomocnicze:
 - *createPopulation* - tworzy początkową populację rozwiązań
 - *giveSubgrids* - dzieli planszę na bloki 3x3
 - *joinSubgrids* - łączy bloki 3x3 w pełną planszę
 - *mutation* - przeprowadza mutację na populacji z zadanyim prawdopodobieństwem
 - *crossover* - przeprowadza krzyżowanie na populacji
 - *converge* - sprawdza, czy algorytm znalazł rozwiązanie lub czy wszystkie rozwiązania są identyczne
 - *select* - wybiera rozwiązania do następnej iteracji na podstawie ich dopasowania
 - *best* - zwraca najlepsze rozwiązanie z populacji
- funkcja *main* zawierająca:
 - inicjalizacja planszy sudoku
 - wywołanie funkcji *GA* z planszą sudoku jako argumentem
 - wyświetlanie wyniku

Algorytm GPSO

Pseudokod do głównej funkcji algorytmu GPSO przedstawiono poniżej.

Algorithm 2: Algorytm GPSO dla sudoku

Function *GPSO***Data:** plansza sudoku**Result:** najlepsze znalezione rozwiązanieZainicjalizuj parametry algorytmu: *N_SWARM*, *MAX_ITERATIONS*, *W1*, *W2*, *W3*;

Inicjalizacja roju;

while *nie zostało znalezione rozwiązanie i nie osiągnięto MAX_ITERATIONS* **do****foreach** *cząstka w roju* **do**

Przeprowadź operację krzyżowania i mutacji;

Zaaktualizuj pozycję cząstki;

Zaaktualizuj najlepszą lokalną pozycję cząstki;

Zaaktualizuj najlepszą globalną pozycję roju;

end**end****return** *najlepsze znalezione rozwiązanie*;**end**

Dodatkowo zdefiniowano:

- klasę *Sudoku* zawierającą:
 - planszę sudoku (grid)
 - poziom trudności
 - metodę do wypisywania planszy
- klasę *Particle* zawierającą:
 - inicjalizacja cząstki z daną planszą sudoku
 - aktualną pozycję, najlepszą lokalną pozycję, wartość dopasowania oraz planszę sudoku
 - metody:
 - * ustalanie pierwszej pozycji
 - * aktualizacja obecnej pozycji
 - * ustalanie wartości dopasowania
 - * ustalanie najlepszej pozycji lokalnej
 - * generowanie maski
 - * przeprowadzanie krzyżowania (dwa podejścia)
 - * mutacja
- klasę *Swarm* zawierającą:
 - globalnie najlepszą pozycję, listę cząstek, rozmiar roju i wagi
 - metody:
 - * dodawania cząstek do roju
 - * ustalania globalnie najlepszej pozycji
- funkcję *converge* sprawdzającą, czy którakolwiek cząstka osiągnęła idealne dopasowanie
- funkcję *main*, która:
 - inicjalizuje problem sudoku
 - uruchamia algorytm GPSO
 - wyświetla wynik

4 Aplikacja

Celem implementacji algorytmów wykorzystano język Python w wersji WERSJA wraz z standardowymi bibliotekami:

- typing – anotacje typów zmiennych,
- randn – generowanie liczb pseudolosowych z zadanego rozkładu, pseudolosowe wybieranie elementów z listy oraz inne operacje pseudolosowe,
- copy – operacje kopiowania obiektów; w szczególności funkcja deepcopy, która służy do tworzenia kopii głębokich.

Ponadto wykorzystano zewnętrzną bibliotekę numpy również do generowania pseudolosowych wartości. Użytkownik ma możliwość uruchomienia algorytmów dla sudoku zdefiniowanego w pliku `input_sudoku.py`. Struktura pliku wraz z przykładowym wypełnieniem znajduje się poniżej:

```
1 DIFFICULTY = 1
2 INPUT_GRID = [[6, 5, 0, 0, 0, 7, 9, 0, 3],
3 [0, 0, 2, 1, 0, 0, 6, 0, 0],
4 [9, 0, 0, 0, 6, 3, 0, 0, 4],
5 [1, 2, 9, 0, 0, 0, 0, 0, 0],
6 [3, 0, 4, 9, 0, 8, 1, 0, 0],
7 [0, 0, 0, 3, 0, 0, 4, 7, 9],
8 [0, 0, 6, 0, 8, 0, 3, 0, 5],
9 [7, 4, 0, 5, 0, 0, 0, 0, 1],
10 [5, 8, 1, 4, 0, 0, 0, 2, 6]]
```

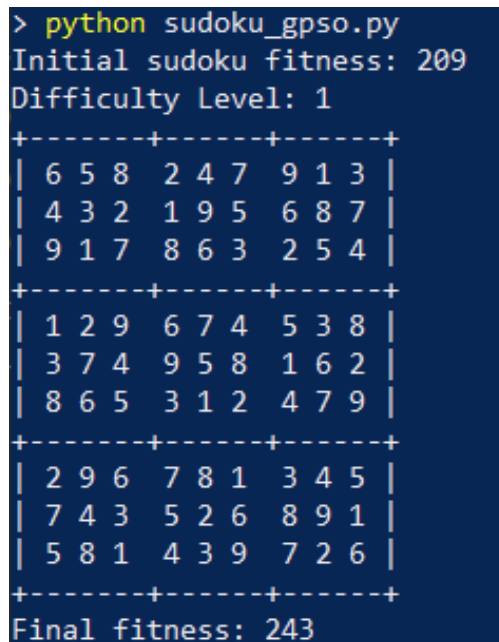
Użytkownik ma możliwość uruchomienia:

- dla algorytmu genetycznego: `sudoku_gen`
- dla algorytmu GPSO: `sudoku_gpso`

Uruchomienie odpowiedniego algorytmu może nastąpić poprzez command line, kolejno dla algorytmu genetycznego i GPSO:

```
1 python sudoku_gen.py
2 python sudoku_gpso.py
```

Przy uruchomieniu następuje wyświetlenie następujących przykładowych komunikatów w konsoli (rys.1):



```
> python sudoku_gpso.py
Initial sudoku fitness: 209
Difficulty Level: 1
+-----+-----+-----+
| 6 5 8 | 2 4 7 | 9 1 3 |
| 4 3 2 | 1 9 5 | 6 8 7 |
| 9 1 7 | 8 6 3 | 2 5 4 |
+-----+-----+-----+
| 1 2 9 | 6 7 4 | 5 3 8 |
| 3 7 4 | 9 5 8 | 1 6 2 |
| 8 6 5 | 3 1 2 | 4 7 9 |
+-----+-----+-----+
| 2 9 6 | 7 8 1 | 3 4 5 |
| 7 4 3 | 5 2 6 | 8 9 1 |
| 5 8 1 | 4 3 9 | 7 2 6 |
+-----+-----+-----+
Final fitness: 243
```

Rysunek 1: Komunikaty wyświetlane przy uruchomieniu programu

Program używając danych wejściowych wprowadzonych przez użytkownika zakłada, że plik `input_sudoku.py` jest uzupełniony poprawnie. W przypadku wpisania wartości nieprawidłowego formatu (np.: nieprawidłowa ilość wierszy) podniesie wyjątek i spowoduje to wyświetleniem błędu w konsoli. Jeżeli natomiast użytkownik uzupełni planszę w sposób uniemożliwiający poprawne rozwiązanie (np.: poprzez wprowadzenie duplikatów w wierszu/kolumnie/oknie 3x3) to program nie zgłosi błędów z tego wynikających - zwróci najlepsze dopasowane rozwiązanie, lecz nie zmieni lokalizacji cyfr wprowadzonych przez użytkownika.

Testy przeprowadzono w oparciu o 3 plansze sudoku o poziomie trudności ustalone jako:

- Łatwe: 1:

6	5				7	9		3
		2	1			6		
9				6	3			4
1	2	9						
3		4	9		8	1		
			3			4	7	9
		6		8		3		5
7	4		5					1
5	8	1	4				2	6

Rysunek 2: Plansza o poziomie trudności 1.

- Średnie: 2:

5								
8	1	7	2	6	5			
6						2	5	
					4			5
9	8		5	3				2
		5	7		8		6	
				8	1		7	9
	9	6						8
1		8	4		9			

Rysunek 3: Plansza o poziomie trudności 2.

- Trudne: 3:

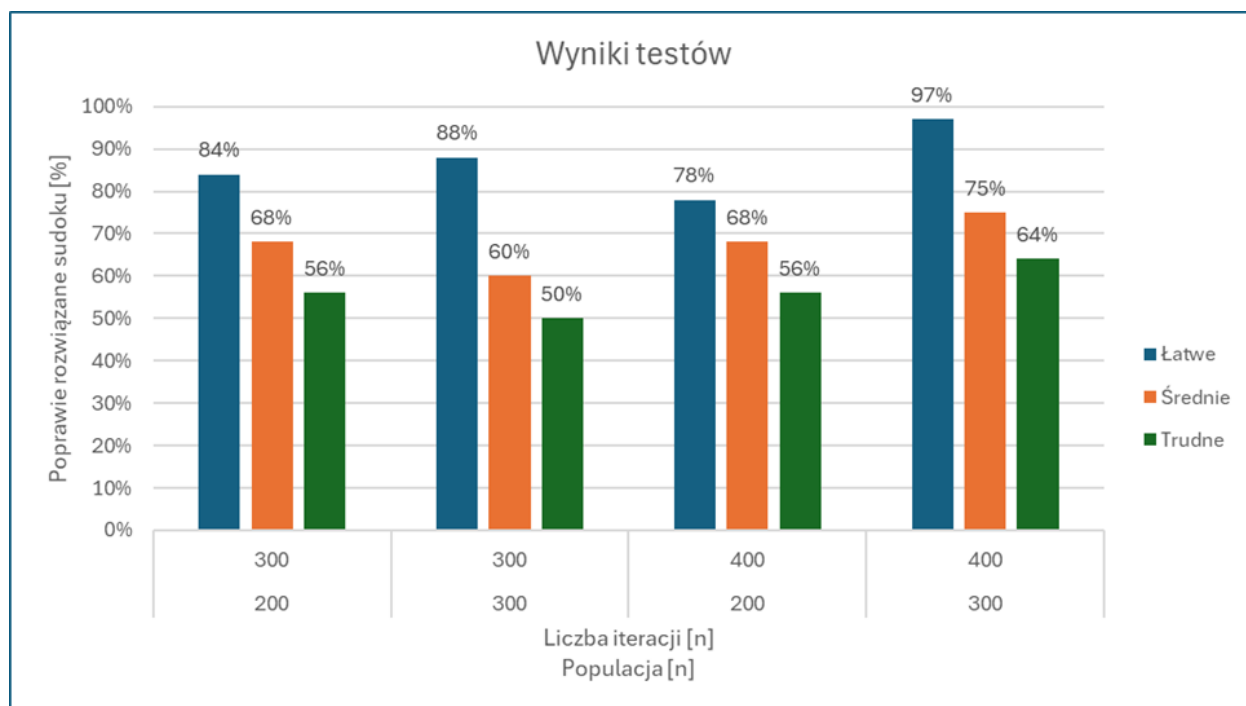
9		2							
							1	5	
7			6		2				
			7	9					
	6	1		8					
				3		1			
		7				9	4		
4							2	1	
	8				4	6			

Rysunek 4: Plansza o poziomie trudności 3.

5 Eksperymenty

Algorytm genetyczny

Dla powyższego algorytmu genetycznego zostały przeprowadzone testy, których wyniki prezentują się następująco. Najlepsze wyniki otrzymaliśmy przy prawdopodobieństwie mutacji równym 6% i dla takiej wartości parametru przeprowadziliśmy dalsze testy. Poniższy wykres prezentuje wyniki testów dla poszczególnych par wartości parametrów, czyli wielkość populacji oraz liczba iteracji.



Rysunek 5: Wyniki testów dla algorytmu genetycznego

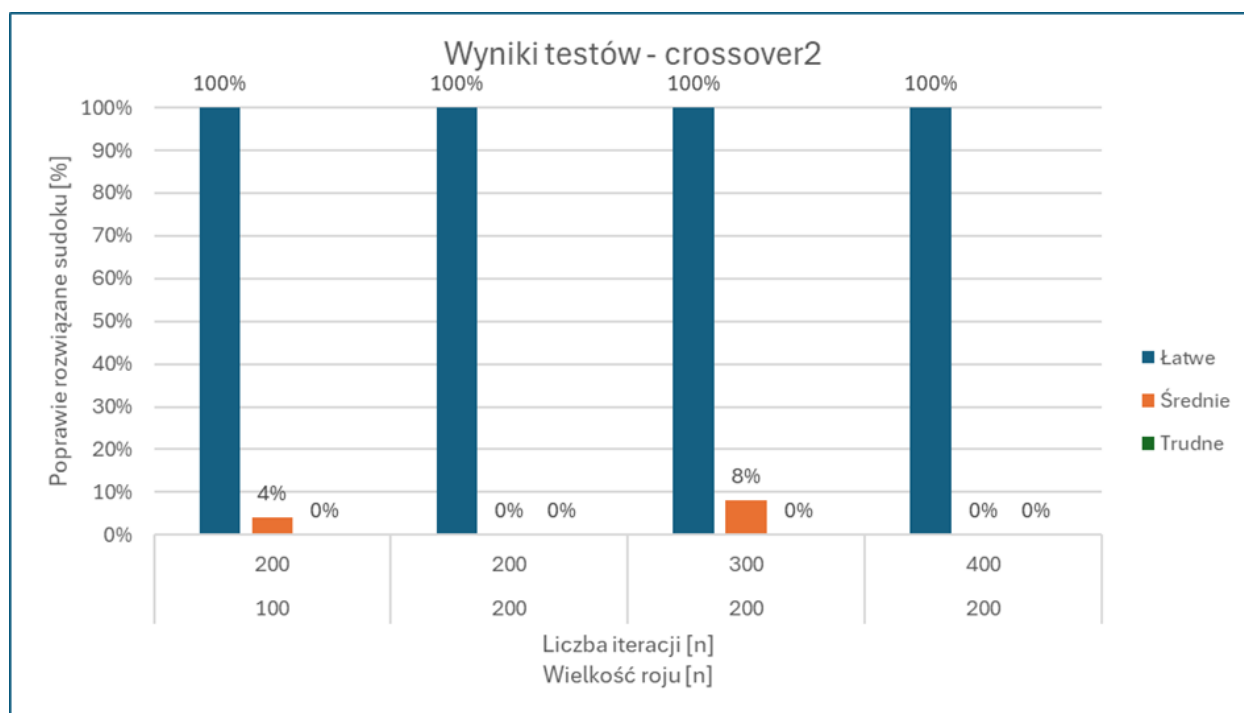
Dla pary początkowej populacja = 300, liczba iteracji = 200 otrzymaliśmy poprawność wyników na poziomie sudoku łatwe = 84. Kolejne testy wykazały że zwiększenie populacji bez zwiększenia liczby iteracji zwiększa jedynie poprawność rozwiązań łatwego sudoku ale z drugiej strony obniża wyniki dla średniego oraz trudnego (testy dla pary 300, 300). Natomiast samo zwiększenie liczby iteracji nie ma wpływu na średnie i trudne sudoku a nawet pogarsza wyniki dla łatwego (testy dla pary 400, 200). Znaczącą poprawę wyników otrzymujemy jedynie gdy zwiększamy zarówno wielkość populacji jak i liczbę iteracji (testy dla pary 400, 300).

Algorytm GPSO

Dla algorytmu GPSO parametry dla których zostały przeprowadzone testy prezentują się następująco:

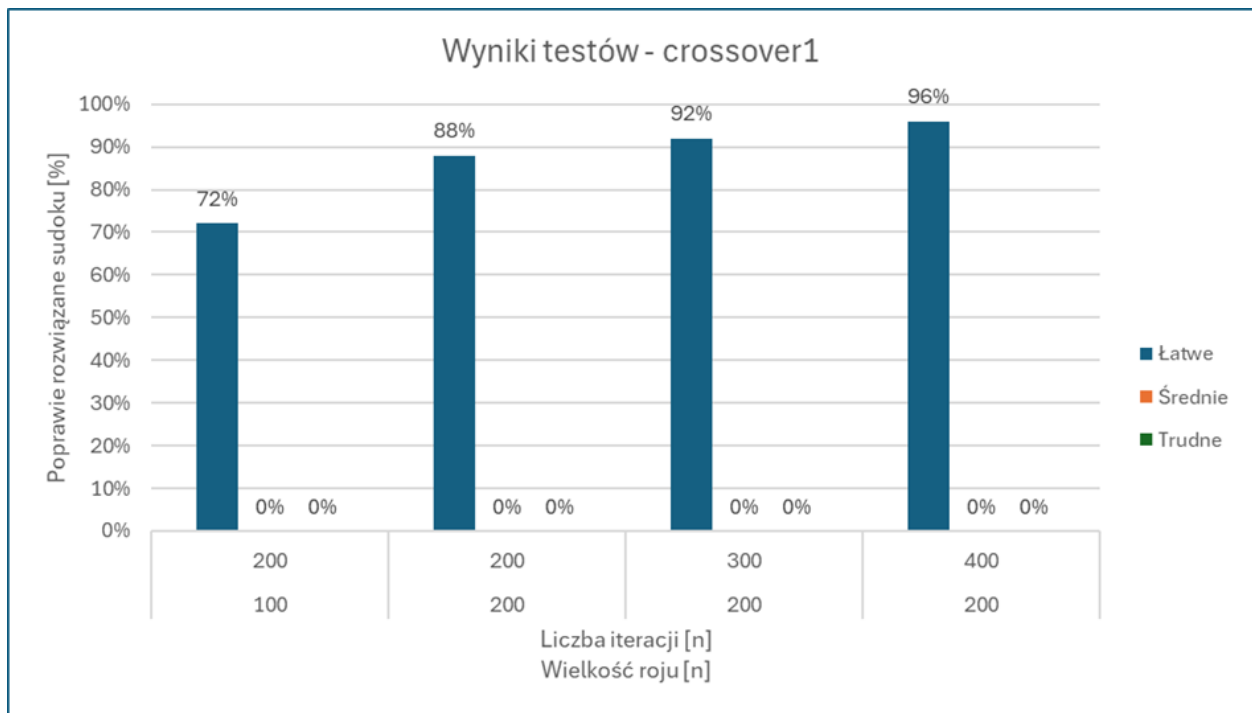
- Waga obecnej pozycji = 0.2
- Waga najlepszej pozycji lokalnej = 0.6
- Waga najlepszej pozycji globalnej = 0.2
- Prawdopodobieństwo mutacji = 50

Testy wstępne wykazały że są to najbardziej odpowiednie parametry. Poszczególne części wykresu prezentują wyniki zależnie od zmiany dwóch parametrów: wielkości roju oraz liczby iteracji. Poniższy wykres prezentuje wyniki otrzymane dla algorytmu wykorzystującego drugą wersję funkcji crossover – crossover2.



Rysunek 6: Wyniki testów dla algorytmu GPSO z zastosowaniem krzyżowania: crossover2.

Dla początkowej pary parametrów wielkość roju = 100 oraz liczba iteracji = 200, otrzymaliśmy następujące wyniki: łatwe sudoku było zawsze rozwiązywane poprawnie, jednak dla średniego było to jedynie 4% a trudne nie zostało rozwiązane poprawnie ani raz. Modyfikacja pary parametrów niewiele zmieniła. Niewielką poprawę otrzymaliśmy dla pary parametrów równej 300, 200 a mianowicie był to zysk 4 punktów procentowych dla średniego sudoku. Warto zauważyć że dla jeszcze większej liczby iteracji wyniki znacząco się pogarszają wyniki dla średniego sudoku spadają do zera (testy dla pary 400, 200). Poniższy wykres prezentuje wyniki otrzymane dla algorytmu wykorzystującego pierwszą wersję funkcji crossover – crossover1.



Rysunek 7: Wyniki testów dla algorytmu GPSO z zastosowaniem krzyżowania: crossover1.

Obserwując powyższy wykres można zauważyć że crossover1 wymaga znacznie większych wartości wielkości roju oraz liczby iteracji aby dla łatwego poziomu trudności zbliżyć się wynikami do funkcji crossover2. Natomiast zarówno sudoku ze średnim jak i trudnym poziomem trudności nie zostało rozwiązane poprawnie ani raz.

6 Podsumowanie i wnioski

Projekt przedstawia implementację dwóch algorytmów optymalizacyjnych - algorytmu genetycznego oraz algorytmu GPSO (Geometric Particle Swarm Optimization) - do rozwiązywania problemu sudoku. Każdy z algorytmów został szczegółowo opisany, a także dostarczono ich pseudokody oraz informacje dotyczące implementacji w języku Python.

Algorytm genetyczny opiera się na koncepcji ewolucji biologicznej, wykorzystując mechanizmy selekcji, krzyżowania, mutacji oraz warunek stopu do generowania coraz lepszych rozwiązań plansz sudoku. Operuje na populacji rozwiązań, gdzie każde rozwiązanie jest oceniane pod kątem dopasowania do wymagań sudoku. Krzyżowanie pozwala na połączenie cech najlepszych rozwiązań, mutacja wprowadza element losowości, a selekcja wybiera najlepsze rozwiązania do kolejnej iteracji. Warunkiem stopu może być uzyskanie optymalnego rozwiązania lub osiągnięcie maksymalnej liczby iteracji.

Algorytm GPSO bazuje na koncepcji roju cząstek, gdzie każda cząstka reprezentuje potencjalne rozwiązanie sudoku. Algorytm ten wykorzystuje wagi aktualizacji pozycji, prawdopodobieństwo mutacji oraz różne wersje funkcji krzyżowania w celu znalezienia optymalnego rozwiązania.

Testy przeprowadzone dla obu algorytmów wykazały, że odpowiednie dobranie parametrów oraz dostosowanie operacji genetycznych ma kluczowe znaczenie dla efektywności rozwiązywania sudoku. W przypadku algorytmu genetycznego, wielkość populacji i liczba iteracji wpływają na poprawę wyników. Zwiększenie populacji bez zwiększenia liczby iteracji przynosi wzrost poprawności rozwiązań dla sudoku łatwego, lecz obniża wyniki dla sudoku średniego i trudnego. Natomiast zwiększenie liczby iteracji nie miało wpływu na sudoku średnie i trudne, a nawet pogarszało wyniki dla sudoku łatwego. Poprawa wyników została zaobserwowana po zwiększeniu zarówno wielkości populacji, jak i liczby iteracji.

Dla algorytmu GPSO na wyniki wpłynęły waga obecnej pozycji, waga najlepszej pozycji lokalnej i globalnej oraz prawdopodobieństwo mutacji. Analiza wyników pokazała, że modyfikacja parametrów wielkości roju i liczby iteracji miała niewielki wpływ, choć zwiększenie liczby iteracji mogło prowadzić do znacznego pogorszenia skuteczności dla sudoku średniego. Również obserwowano, że wykorzystanie drugiej wersji funkcji krzyżowania (crossover2) przynosiło lepsze wyniki w porównaniu z pierwszą wersją (crossover1), która

wymagała większej liczby iteracji i roju, ale nie gwarantowała poprawnych rozwiązań dla sudoku o średnim i trudnym poziomie trudności.

Podsumowując, projekt dostarcza kompleksowego spojrzenia na wykorzystanie algorytmów optymalizacyjnych do rozwiązywania problemów sudoku, prezentując ich działanie, implementację, wyniki testów oraz wnioski z nich wynikające. Może to stanowić podstawę do dalszych badań nad tym tematem oraz rozwijania bardziej zaawansowanych technik rozwiązywania sudoku.

7 Udział poszczególnych osób w dany etap projektu

Osoba	Zrealizowane zadania
Filip Chrapla	<ol style="list-style-type: none"> 1. Algorytm genetyczny <ul style="list-style-type: none"> - implementacja funkcji fitness 2. Algorytm GPSO <ul style="list-style-type: none"> - implementacja funkcji fitness 3. Dokumentacja <ul style="list-style-type: none"> - podsumowanie i wnioski
Jakub Iskrzycki	<ol style="list-style-type: none"> 1. Algorytm genetyczny <ul style="list-style-type: none"> - implementacja funkcji mutacji 2. Algorytm GPSO <ul style="list-style-type: none"> - implementacja funkcji mutacji 3. Dokumentacja <ul style="list-style-type: none"> - pseudokody
Julia Krysiak	<ol style="list-style-type: none"> 1. Algorytm genetyczny <ul style="list-style-type: none"> - implementacja szkieletu algorytmu z opsem funkcji - implementacja funkcji crossover 2. Algorytm GPSO <ul style="list-style-type: none"> - implementacja szkieletu algorytmu z opisem funkcji - implementacja funkcji converge - implementacja funkcji set_global_best 3. Dokumentacja <ul style="list-style-type: none"> - opis algorytmu genetycznego 4. Inne <ul style="list-style-type: none"> - debugging - organizacja spotkań - zredagowanie sprawozdania w latexie
Artur Mazurkiewicz	<ol style="list-style-type: none"> 1. Algorytm genetyczny <ul style="list-style-type: none"> - implementacja funkcji joinSubgrids 2. Algorytm GPSO <ul style="list-style-type: none"> - implementacja funkcji crossover1 - implementacja funkcji crossover2 - implementacja dekoratora do funkcji krzyżowania - wprowadzenie generowania ilości wierszy z rozkładu trójkątnego do funkcji mutation 3. Dokumentacja <ul style="list-style-type: none"> - opis aplikacji 4. Inne <ul style="list-style-type: none"> - wprowadzenie danych przez użytkownika w input_sudoku.py - wyniki w postaci wypisywania do konsoli - debugging - refactoring
Sylwia Michalska	<ol style="list-style-type: none"> 1. Algorytm genetyczny <ul style="list-style-type: none"> - implementacja szkieletu algorytmu z opisem funkcji - implementacja funkcji crossover 2. Algorytm GPSO <ul style="list-style-type: none"> - implementacja szkieletu algorytmu z opisem funkcji 3. Dokumentacja <ul style="list-style-type: none"> - opis algorytmu rojowego

Osoba	Zrealizowane zadania
	4. Inne - debugging - koordynacja zadań, stworzenie repozytorium, organizacja pracy w repozytorium - zredagowanie sprawozdania w latexie
Konrad Prokop	1. Algorytm genetyczny - implementacja funkcji create_population 2. Algorytm GPSO - implementacja funkcji first_position 3. Dokumentacja - przeprowadzanie testów algorytmu genetycznego - sporządzenie wykresu do testów algorytmu genetycznego - opisanie wyników testów algorytmu genetycznego - przeprowadzanie testów algorytmu GPSO - sporządzenie wykresów do testów algorytmu GPSO - opisanie wyników testów algorytmu GPSO
Kaia Rupniak	1. Algorytm genetyczny - implementacja funkcji giveSubgrids 2. Algorytm GPSO - implementacja funkcji set_local_best_pos 3. Dokumentacja - podsumowanie i wnioski
Jan Rusek	1. Algorytm genetyczny - implementacja funkcji __str__ method 2. Algorytm GPSO - implementacja funkcji __str__ method 3. Dokumentacja - badany problem - spis literatury

8 Spis literatury

Literatura

- [1] A. Moraglio, J. Togelius. *Geometric particle swarm optimization for the sudoku puzzle*. Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007.
- [2] J. M. Weiss. *Genetic Algorithms and Sudoku*. MICS, 2009.
- [3] T. Mantere, J. Koljonen. *Solving, rating and generating Sudoku puzzles with GA*. IEEE Congress on Evolutionary Computation, 2007.
- [4] R. Lewis. *Metaheuristics Can Solve Sudoku Puzzles*. Journal of Heuristics, vol. 13 (4), pp 387-401, 2007.
- [5] M. Perez, T. Marwala. *Stochastic Optimization Approaches for Solving Sudoku*. arXiv:0805.0697, 2008.
- [6] *Sudoku solving algorithms*. https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
- [7] *Sudoku backtracking*. <https://www.geeksforgeeks.org/sudoku-backtracking-7/>