

Ewolucyjne uczenie programów grających w gry

Maciej Uść
Marta Migiel
Sofia Bychkova

Marzec 2025

Streszczenie

W niniejszej pracy przedstawiono zastosowanie algorytmów ewolucyjnych do tworzenia programów grających w gry. Szczególną uwagę poświęcono algorytmom genetycznym i programowaniu genetycznemu jako technikom automatycznego generowania zachowań adaptacyjnych. Na potrzeby części praktycznej zaimplementowano system uczący się przechodzić labirynt o różnym poziomie trudności. System korzysta z funkcji przystosowania oceniającej skuteczność strategii poruszania się. Przeprowadzono eksperymenty, których wyniki przedstawiono graficznie i omówiono. Ostatecznie pokazano, że podejście oparte na programowaniu genetycznym może skutecznie uczyć programy rozwiązywać zadania oparte na eksploracji przestrzeni i podejmowaniu decyzji.

Spis treści

1	Wstęp	5
1.1	Czym są algorytmy ewolucyjne?	5
1.2	Algorytmy genetyczne i programowanie genetyczne	5
1.3	Jakość i ocena algorytmów	6
2	Wprowadzenie do projektu programu: Labirynt	7
3	Architektura systemu	7
3.1	Przegląd klas	7
4	Generator labiryntów (MazeGenerator)	7
4.1	Parametry konfiguracyjne	7
4.2	Algorytm generacji	8
4.3	Funkcja złożoności	8
5	Reprezentacja osobnika (Individual)	9
5.1	Struktura danych	9
5.2	Algorytm ruchu	9
5.3	Heurystyka odległościowa	9
5.4	Funkcja przystosowania	9
6	System pamięci ścieżek (PathMemory)	10
6.1	Struktury danych	10
6.2	Algorytm analizy ścieżki	10
7	Ewolucyjny solver (EvolutionaryMazeSolver)	10
7.1	Parametry algorytmu	10
7.2	Główna pętla algorytmu	11
7.3	Operatory ewolucyjne	11
7.3.1	Selekcja turniejowa	11
7.3.2	Krzyżowanie	11
7.3.3	Mutacja	12
8	System wizualizacji	12
8.1	Hierarchia warstw	12
8.2	Kalkulacja rozmiaru komórki	12
9	Analiza teoretyczna	13
9.1	Złożoność obliczeniowa	13
9.1.1	Generacja labiryntu	13
9.1.2	Pojedynczy krok osobnika	13
9.1.3	Jedno pokolenie	13
9.1.4	Całkowita złożoność	13
9.2	Analiza zbieżności	13
9.2.1	Warunki zakończenia	13
9.2.2	Mechanizm balance eksploracji i eksploatacji	13
10	Metryki wydajności	14
10.1	Zbierane statystyki	14
10.2	System oceny użytkownika	14

11 Zalety i ograniczenia	14
11.1 Zalety implementacji	14
11.2 Ograniczenia	14
12 Analiza wyników eksperymentalnych	15
12.1 Wykresy i ich interpretacja	16
12.2 Podsumowanie obserwacji eksperymentalnych	22
13 Zakończenie	22

1 Wstęp

1.1 Czym są algorytmy ewolucyjne?

Algorytm ewolucyjny to heurystyczny algorytm optymalizacji, który wykorzystuje techniki inspirowane procesami ewolucji biologicznej. Działa on na populacji potencjalnych rozwiązań (zwanymi osobnikami), które są modyfikowane z pokolenia na pokolenie za pomocą operatorów takich jak selekcja, krzyżowanie (rekombinacja) oraz mutacja. Każdy osobnik oceniany jest za pomocą funkcji przystosowania (ang. *fitness function*), która określa jakość danego rozwiązania. Proces ten iteruje, aż zostanie spełnione określone kryterium zatrzymania (np. liczba pokoleń lub osiągnięcie odpowiedniej jakości rozwiązania).

Algorytmy ewolucyjne są stosowane w różnych dziedzinach, takich jak optymalizacja kombinatoryczna, uczenie maszynowe, projektowanie automatyczne, czy rozwiązywanie problemów o dużej złożoności.

Działanie algorytmu ewolucyjnego można opisać w kilku podstawowych etapach. Proces rozpoczyna się od utworzenia początkowej populacji potencjalnych rozwiązań, zwanych osobnikami. Każdy osobnik jest reprezentowany przez chromosom zawierający „genetyczną” informację - czyli zakodowaną strukturę rozwiązania (np. wektor liczb, drzewo składniowe).

W każdej iteracji, zwanej generacją, chromosomy są dekodowane i oceniane za pomocą funkcji przystosowania, która mierzy jakość danego rozwiązania względem celu optymalizacji. Na podstawie tych ocen przeprowadzana jest selekcja - słabe osobniki są eliminowane, a te lepiej przystosowane mają większą szansę na przekazanie „materiału genetycznego” do kolejnego pokolenia.

Dalszy rozwój populacji odbywa się poprzez dwa podstawowe mechanizmy: krzyżowanie (rekombinację) oraz mutację. Krzyżowanie łączy fragmenty informacji genetycznej dwóch osobników, co pozwala eksplorować nowe obszary przestrzeni rozwiązań. Mutacja wprowadza drobne losowe zmiany, zapobiegając przedwczesnej zbieżności do lokalnego optimum i zwiększając różnorodność populacji.

Operator selekcji sam w sobie nie generuje nowych rozwiązań - tę rolę pełnią właśnie krzyżowanie i mutacja. Nowo utworzone osobniki tworzą kolejną generację, która zastępuje poprzednią. Proces powtarza się aż do spełnienia kryterium zatrzymania, którym może być np. osiągnięcie odpowiedniego poziomu przystosowania lub przekroczenie maksymalnej liczby generacji.

1.2 Algorytmy genetyczne i programowanie genetyczne

Algorytmy genetyczne (ang. *Genetic Algorithms*, GA) oraz programowanie genetyczne (ang. *Genetic Programming*, GP) są dwoma popularnymi podejściami należącymi do szerszej klasy algorytmów ewolucyjnych. Oba wykorzystują zasady inspirowane mechanizmami biologicznej ewolucji, takimi jak selekcja naturalna, krzyżowanie i mutacja, jednak różnią się sposobem reprezentowania rozwiązań oraz zakresem zastosowań.

Algorytmy genetyczne (GA)

W algorytmach genetycznych populacja kandydatów na rozwiązania (zwanymi osobnikami) jest ewoluowana w kierunku lepszych rozwiązań. Każdy osobnik posiada zestaw właściwości (chromosom lub genotyp), które mogą być modyfikowane poprzez mutacje i krzyżowanie. Tradycyjnie, rozwiązania są reprezentowane binarnie jako ciągi 0 i 1, ale możliwe są również inne kodowania.

Proces ewolucji rozpoczyna się od populacji losowo wygenerowanych osobników i przebiega iteracyjnie, przy czym populacja w każdej iteracji nazywana jest generacją. W każdej generacji oceniana jest przystosowalność każdego osobnika; przystosowalność zazwyczaj odpowiada wartości funkcji celu w rozwiązywanym problemie optymalizacji. Bardziej przystosowane osobniki są stochastycznie wybierane z bieżącej populacji, a ich genomy są modyfikowane (rekombinowane i ewentualnie losowo mutowane), aby utworzyć nową generację. Nowa generacja kandydatów na rozwiązania jest następnie wykorzystywana w kolejnej iteracji algorytmu.

Programowanie genetyczne (GP)

Programowanie genetyczne to technika sztucznej inteligencji naśladująca naturalną ewolucję, która operuje na populacji programów komputerowych. Stosuje operatory genetyczne, takie jak selekcja zgodnie z określoną miarą przystosowalności, mutacja i krzyżowanie.

Operacja krzyżowania polega na wymianie określonych części wybranych par (rodziców) w celu wytworzenia nowych i różnych potomków, którzy stają się częścią nowej generacji programów. Niektóre programy, które nie zostały wybrane do reprodukcji, są kopiowane z bieżącej generacji do nowej generacji. Mutacja polega na zastąpieniu losowej części programu inną losową częścią programu. Następnie selekcja i inne operacje są rekurencyjnie stosowane do nowej generacji programów.

Zazwyczaj członkowie każdej nowej generacji są średnio bardziej przystosowani niż członkowie poprzedniej generacji, a najlepszy program generacji często przewyższa najlepsze programy poprzednich generacji. Ewolucja zazwyczaj kończy się, gdy jakiś indywidualny program osiągnie zdefiniowany poziom biegłości lub przystosowalności.

W pracy tej skupiamy się głównie na programowaniu genetycznym, ponieważ naszym celem jest stworzenie programu, który uczy się przechodzić labirynt, a więc samodzielnie generuje i modyfikuje strategię działania.

1.3 Jakość i ocena algorytmów

Ocena jakości algorytmów ewolucyjnych jest kluczowa dla zrozumienia ich efektywności w rozwiązywaniu problemów optymalizacyjnych. Poniżej przedstawiono główne aspekty tej oceny:

Funkcja przystosowania (fitness function)

Funkcja przystosowania jest podstawowym narzędziem oceny jakości rozwiązań

generowanych przez algorytmy ewolucyjne. Określa ona, jak dobrze dany osobnik (czyli potencjalne rozwiązanie) spełnia założone kryteria. W praktyce funkcja ta może mierzyć różne aspekty, takie jak dokładność, czas wykonania czy minimalizację błędu. Ważne jest, aby funkcja przystosowania była dobrze dopasowana do konkretnego problemu, ponieważ niewłaściwie dobrana może prowadzić do nieoptymalnych wyników lub zbieżności do lokalnych ekstremów.

Zbieżność i różnorodność populacji

Zbieżność algorytmu odnosi się do jego zdolności do znajdowania optymalnych rozwiązań w przestrzeni poszukiwań. Jednak zbyt szybka zbieżność może prowadzić do tzw. przedwczesnej zbieżności, gdzie algorytm utknie w lokalnym optimum. Aby temu zapobiec, istotne jest utrzymanie odpowiedniej różnorodności populacji, co pozwala na eksplorację różnych obszarów przestrzeni rozwiązań i zwiększa szansę na znalezienie globalnego optimum.

Stabilność i powtarzalność wyników

Stabilność algorytmu oznacza jego zdolność do generowania spójnych wyników w kolejnych uruchomieniach. Algorytmy ewolucyjne, ze względu na swoją losową naturę, mogą prowadzić do różnych wyników przy każdym uruchomieniu. Dlatego ważne jest przeprowadzanie wielokrotnych eksperymentów i analizowanie statystyczne wyników, aby ocenić ich powtarzalność i wiarygodność.

2 Wprowadzenie do projektu programu: Labirynt

W dalszej części pracy przedstawiono szczegółową analizę implementacji ewolucyjnego algorytmu rozwiązywania labiryntów. Program wykorzystuje metody sztucznej inteligencji, w szczególności algorytmy ewolucyjne, do znalezienia optymalnej ścieżki w dynamicznie generowanych labiryntach.

3 Architektura systemu

3.1 Przegląd klas

System składa się z czterech głównych klas współpracujących ze sobą:

- `MazeGenerator` - generator labiryntów
- `Individual` - reprezentacja osobnika w populacji
- `PathMemory` - system pamięci ścieżek
- `EvolutionaryMazeSolver` - główny solver ewolucyjny
- `MazeGUI` - interfejs graficzny

4 Generator labiryntów (MazeGenerator)

4.1 Parametry konfiguracyjne

Generator tworzy labirynty o trzech poziomach złożoności:

$$\text{rozmiar} = \begin{cases} 21 \times 21 & \text{dla poziomu łatwy} \\ 31 \times 31 & \text{dla poziomu średni} \\ 41 \times 41 & \text{dla poziomu trudny} \end{cases} \quad (1)$$

4.2 Algorytm generacji

Wykorzystuje algorytm **Randomized Depth-First Search**:

Algorithm 1 Generacja labiryntu metodą DFS

```

1: Inicjalizuj labirynt wypełniony ścianami
2: Wybierz losowy punkt startowy o nieparzystych współrzędnych
3: Umieść punkt na stosie
4: while stos niepusty do
5:   Pobierz bieżący punkt ze stosu
6:   Wymieszaj kierunki:  $[(-2, 0), (2, 0), (0, -2), (0, 2)]$ 
7:   for każdy kierunek  $(\Delta x, \Delta y)$  do
8:      $x_{new} = x_{current} + \Delta x$ 
9:      $y_{new} = y_{current} + \Delta y$ 
10:    if punkt  $(x_{new}, y_{new})$  jest dostępny then
11:      Utwórz przejście w punkcie pośrednim
12:       $x_{mid} = x_{current} + \Delta x / 2$ 
13:       $y_{mid} = y_{current} + \Delta y / 2$ 
14:      Oznacz  $(x_{mid}, y_{mid})$  jako przejście
15:      Oznacz  $(x_{new}, y_{new})$  jako przejście
16:      Dodaj  $(x_{new}, y_{new})$  na stos
17:      BREAK
18:    end if
19:  end for
20:  if nie wykonano ruchu then
21:    Usuń punkt ze stosu
22:  end if
23: end while

```

4.3 Funkcja złożoności

Dodatkowa złożoność wprowadzana jest przez współczynnik modyfikacji:

$$\text{complexity_factor} = \begin{cases} 0.02 & \text{łatwy (2\% modyfikacji)} \\ 0.04 & \text{średni (4\% modyfikacji)} \\ 0.06 & \text{trudny (6\% modyfikacji)} \end{cases} \quad (2)$$

Liczba modyfikacji obliczana jest jako:

$$N_{\text{modifications}} = \lfloor (\text{width} \times \text{height}) \times \text{complexity_factor} \rfloor \quad (3)$$

5 Reprezentacja osobnika (Individual)

5.1 Struktura danych

Każdy osobnik w populacji charakteryzuje się atrybutami:

- **path**: Lista odwiedzonych pozycji $[(y_1, x_1), (y_2, x_2), \dots, (y_n, x_n)]$
- **fitness**: Wartość przystosowania $f \in \mathbb{R}$
- **visited**: Zbiór odwiedzonych komórek (optymalizacja $O(1)$ lookup)
- **max_steps**: Limit kroków = width $\times$ height

5.2 Algorytm ruchu

Algorithm 2 Ruch osobnika w labirynie

```
1: if bieżąca pozycja = cel then
2:   finished = True
3:   RETURN True
4: end if
5: steps += 1
6: if steps > max_steps then
7:   stuck = True
8:   RETURN False
9: end if
10: Pobierz możliwe ruchy z bieżącej pozycji
11: Odfiltruj odwiedzone pozycje
12: if brak możliwych ruchów then
13:   stuck = True
14:   RETURN False
15: end if
16: Sortuj ruchy według heurystyki odległościowej
17: Wybierz losowo z 2 najlepszych opcji
18: Dodaj wybrany ruch do ścieżki
```

5.3 Heurystyka odległościowa

Program wykorzystuje odległość Manhattan jako funkcję heurystyczną:

$$h(n) = |x_n - x_{goal}| + |y_n - y_{goal}| \quad (4)$$

gdzie (x_n, y_n) to aktualna pozycja, a (x_{goal}, y_{goal}) to pozycja celu.

5.4 Funkcja przystosowania

Funkcja fitness różni się w zależności od stanu osobnika:

Dla udanego rozwiązania:

$$f_{success} = 1000 - |\text{path}| + (100 - N_{wrong_turns} \times 10) \quad (5)$$

Dla zablokowanego osobnika:

$$f_{stuck} = -h(\text{current_pos}) - N_{dead_ends} \times 5 \quad (6)$$

Dla aktywnego osobnika:

$$f_{active} = 100 - h(\text{current_pos}) - |\text{path}| \times 0.1 \quad (7)$$

6 System pamięci ścieżek (PathMemory)

6.1 Struktury danych

Klasa PathMemory przechowuje:

- `valid_paths`: Zbiór udanych ścieżek
- `invalid_paths`: Zbiór nieudanych ścieżek
- `dead_ends`: Zbiór komórek ślepych
- `branches`: Zbiór rozgałęzień
- `wrong_turns`: Zbiór błędnych skrętów
- `all_visited`: Zbiór wszystkich odwiedzonych komórek

6.2 Algorytm analizy ścieżki

Algorithm 3 Analiza ścieżki osobnika

```
1: for każda pozycja  $p$  w ścieżce do
2:    $M_{all} \leftarrow$  wszystkie możliwe ruchy z pozycji  $p$ 
3:    $M_{available} \leftarrow M_{all} \setminus \{\text{pozycje już odwiedzone}\}$ 
4:   if  $|M_{all}| > 2$  then
5:     Dodaj  $p$  do zbioru rozgałęzień
6:     if wybrany ruch  $\neq$  najlepszy heurystycznie then
7:       Dodaj  $p$  do zbioru błędnych skrętów
8:     end if
9:   end if
10:  if  $|M_{available}| = 0$  AND  $p \neq$  cel then
11:    Dodaj  $p$  do zbioru ślepych komórek
12:    Oznacz całą ścieżkę od ostatniego rozgałęzienia jako ślepą
13:  end if
14: end for
```

7 Ewolucyjny solver (EvolutionaryMazeSolver)

7.1 Parametry algorytmu

Kluczowe parametry algorytmu ewolucyjnego:

Parametr	Wartość	Uzasadnienie
Rozmiar populacji	50	Balans różnorodność/wydajność
Liczba elit	5	10% najlepszych osobników
Rozmiar turnieju	3	Umiarkowana presja selekcyjna
Prawdopodobieństwo mutacji	0.3	30% szans na mutację
Maksymalna liczba kroków	$w \times h$	Proporcjonalna do rozmiaru

7.2 Główna pętla algorytmu

Algorithm 4 Algorytm ewolucyjny

```

1: Inicjalizuj populację  $P_0$  o rozmiarze  $n$ 
2:  $t \leftarrow 0$ 
3: while nie znaleziono rozwiązania AND  $t < t_{max}$  do
4:   for każdy osobnik  $i \in P_t$  do
5:     Wykonaj krok symulacji dla  $i$ 
6:     Oblicz  $fitness(i)$ 
7:     Dodaj informacje o  $i$  do pamięci ścieżek
8:   end for
9:   if znaleziono rozwiązanie then
10:    BREAK
11:  end if
12:   $P_{t+1} \leftarrow \emptyset$ 
13:  Dodaj elitę do  $P_{t+1}$ 
14:  while  $|P_{t+1}| < n$  do
15:     $parent_1 \leftarrow selekcja\_turniejowa(P_t)$ 
16:     $parent_2 \leftarrow selekcja\_turniejowa(P_t)$ 
17:     $child \leftarrow krzyżowanie(parent_1, parent_2)$ 
18:     $mutacja(child)$ 
19:    Dodaj  $child$  do  $P_{t+1}$ 
20:  end while
21:   $P_t \leftarrow P_{t+1}$ 
22:   $t \leftarrow t + 1$ 
23: end while

```

7.3 Operatory ewolucyjne

7.3.1 Selekcja turniejowa

$$winner = \arg \max_{i \in T} f(i) \quad (8)$$

gdzie T to losowo wybrana grupa k osobników ($k = 3$).

7.3.2 Krzyżowanie

Krzyżowanie jednopunktowe z filtrowaniem duplikatów:

Algorithm 5 Krzyżowanie

```
1:  $L_{min} \leftarrow \min(|\text{path}_1|, |\text{path}_2|)$ 
2:  $c \leftarrow$  losowy punkt cięcia z zakresu  $[1, L_{min} - 1]$ 
3:  $\text{path}_{child} \leftarrow \text{path}_1[0 : c]$ 
4: for każdy punkt  $p$  w  $\text{path}_2[c :]$  do
5:   if  $p \notin \text{visited}_{child}$  then
6:     Dodaj  $p$  do  $\text{path}_{child}$ 
7:     Dodaj  $p$  do  $\text{visited}_{child}$ 
8:   end if
9: end for
```

7.3.3 Mutacja

Mutacja przez obcięcie ścieżki:

$$P(\text{mutacja}) = 0.3 \quad (9)$$

Algorithm 6 Mutacja

```
1: if  $\text{random}() < 0.3$  AND  $|\text{path}| > 1$  then
2:    $c \leftarrow$  losowy punkt z zakresu  $[1, |\text{path}| - 1]$ 
3:    $\text{removed} \leftarrow \{\text{path}[c :]\}$ 
4:    $\text{path} \leftarrow \text{path}[0 : c]$ 
5:    $\text{visited} \leftarrow \text{visited} \setminus \text{removed}$ 
6: end if
```

8 System wizualizacji

8.1 Hierarchia warstw

Program wykorzystuje 9-warstwowy system wizualizacji:

1. **LIGHT_BLUE**: Wszystkie odwiedzone komórki (tło)
2. **ORANGE**: Nieudane ścieżki
3. **RED**: Komórki ślepe
4. **DARK_ORANGE**: Błędne skręty (koła)
5. **CYAN**: Rozgałęzienia (koła z obwódką)
6. **GREEN**: Udaane ścieżki (poza najlepszą)
7. **BLUE**: Aktywne ścieżki (bieżące próby)
8. **DARK_GREEN**: Najlepsze rozwiązanie (z obwódką)
9. **FUKSIA/DARK_GREEN**: Start i meta

8.2 Kalkulacja rozmiaru komórki

$$\text{cell_size} = \min \left(\frac{\text{maze_display_width}}{\text{maze_width}}, \frac{600}{\text{maze_height}} \right) \quad (10)$$

9 Analiza teoretyczna

9.1 Złożoność obliczeniowa

9.1.1 Generacja labiryntu

$$T_{generation} = O(w \times h) \quad (11)$$

$$S_{generation} = O(w \times h) \quad (12)$$

9.1.2 Pojedynczy krok osobnika

$$T_{step} = O(1) \quad (\text{sprawdzenie 4 kierunków}) \quad (13)$$

$$S_{step} = O(k) \quad \text{gdzie } k = |\text{path}| \quad (14)$$

9.1.3 Jedno pokolenie

$$T_{generation} = O(p \times s) \quad (15)$$

$$S_{generation} = O(p \times k) \quad (16)$$

gdzie p = rozmiar populacji, s = średnia liczba kroków na osobnika.

9.1.4 Całkowita złożoność

$$T_{total} = O(g \times p \times s) \quad (17)$$

$$S_{total} = O(p \times k + m) \quad (18)$$

gdzie g = liczba pokoleń, m = rozmiar pamięci ścieżek.

9.2 Analiza zbieżności

9.2.1 Warunki zakończenia

Algorytm kończy działanie gdy:

1. Znaleziono ścieżkę do celu: $\exists i : \text{path}[i][-1] = \text{goal}$
2. Osiągnięto limit kroków: $\text{steps} > w \times h$
3. Brak aktywnych osobników: $|\{i : \neg \text{finished}[i] \wedge \neg \text{stuck}[i]\}| < 0.1p$

9.2.2 Mechanizm balance eksploracji i eksploatacji

- **Eksploracja:** Mutacja (30%), losowość w heurystyce
- **Eksploatacja:** Elityzm (10%), selekcja turniejowa
- **Pamięć:** Akumulacja wiedzy o strukturze labiryntu

10 Metryki wydajności

10.1 Zbierane statystyki

Program automatycznie zbiera następujące metryki:

Metryka	Opis
total_attempts	Łączna liczba prób osobników
success_count	Liczba udanych rozwiązań
error_count	Liczba nieudanych prób
dead_ends_count	Liczba znalezionych ślepych komórek
generation	Numer bieżącego pokolenia
elapsed_time	Czas rozwiązywania [s]
solution_length	Długość najlepszego rozwiązania

10.2 System oceny użytkownika

Po znalezieniu rozwiązania użytkownik może ocenić jego jakość:

- Przycisk **"Prawidłowe"** - rozwiązanie poprawne
- Przycisk **"Nieprawidłowe"** - rozwiązanie niepoprawne

Wszystkie dane zapisywane są do pliku CSV z timestampem dla dalszej analizy.

11 Zalety i ograniczenia

11.1 Zalety implementacji

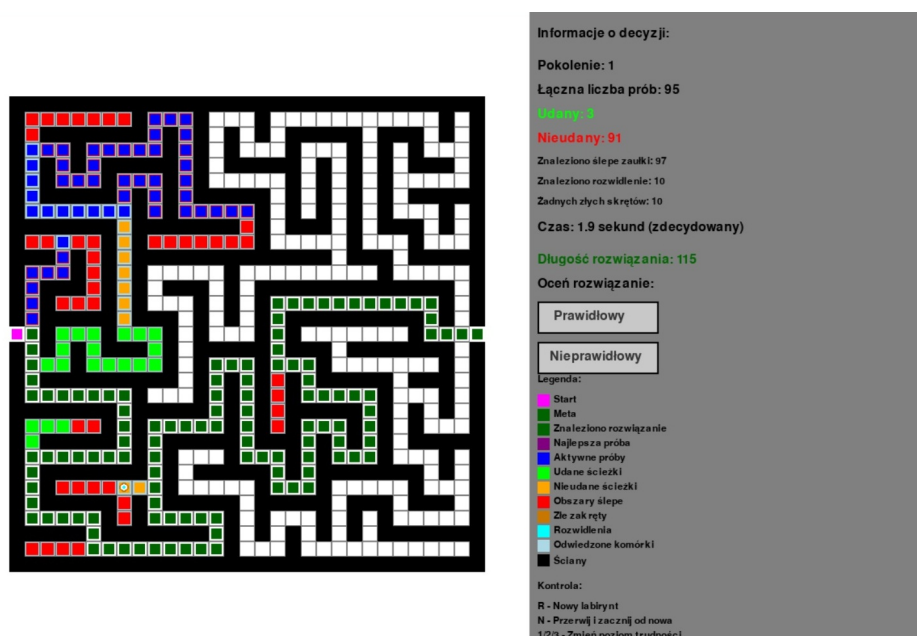
1. **Równoległość:** Wiele osobników eksploruje jednocześnie.
2. **Adaptacyjność:** Algorytm uczy się struktury labiryntu.
3. **Wizualizacja:** Bogata, wielowarstwowa wizualizacja procesu.
4. **Zbieranie danych:** Szczegółowe metryki i możliwość oceny jakości.
5. **Interaktywność:** Możliwość zmiany parametrów w czasie rzeczywistym.

11.2 Ograniczenia

1. **Brak gwarancji:** Program nie zawsze znajdzie rozwiązanie.
2. **Zużycie pamięci:** Przechowywanie wszystkich ścieżek może być kosztowne.
3. **Strojenie parametrów:** Wymaga doboru odpowiednich wartości.
4. **Stochastyczność:** Różne uruchomienia dają różne wyniki.

12 Analiza wyników eksperymentalnych

W niniejszym rozdziale przedstawiono analizę wyników działania algorytmu ewolucyjnego na zadaniu przechodzenia przez labirynt. Wykorzystano zbiór danych zawierający 300 prób rozwiązania labiryntów o różnym poziomie trudności. Dane obejmują m.in. liczbę pokoleń, czas działania, liczbę błędów, a także ocenę użytkownika. Analizę przeprowadzono w środowisku R z wykorzystaniem bibliotek `ggplot2`, `caret` oraz `randomForest`.

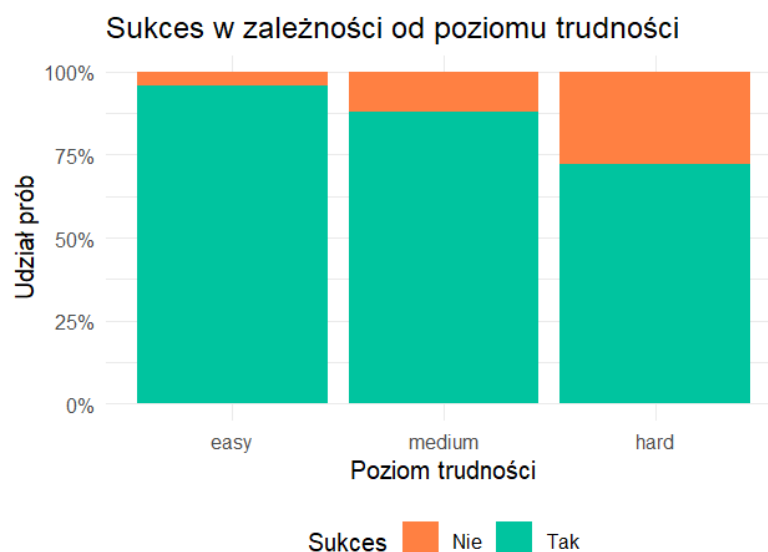


Rysunek 1: Widok graficzny środowiska testowego. Kolorami oznaczono m.in. ścieżki aktywnych i zakończonych prób, ślepe zaułki, rozwidlenia, błędne skrety oraz najlepszą odnalezioną trasę od punktu startowego (różowy) do mety (zielony).

Na Rysunku 1 przedstawiono widok graficzny labiryntu w aplikacji testowej. Interfejs umożliwia śledzenie aktywnych osobników, ocenę ich ścieżek oraz analizę jakości rozwiązania. Wizualizacja wspomaga ocenę poprawności działania algorytmu oraz ocenę zachowania osobników w czasie rzeczywistym.

Ocena prawidłowości wyniku działania programu zależy od użytkownika. W tym przypadku za poprawne uznaje się rozwiązania pokolorowane na zielono bez widocznych przerw, natomiast za błędne — te, które nie zostały znalezione lub zostały zaznaczone nieprawidłowo (z przerwami) na mapie labiryntu.

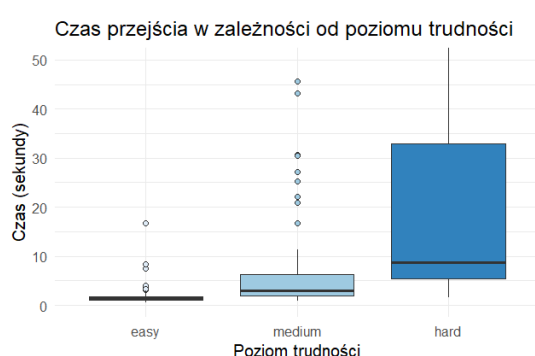
12.1 Wykresy i ich interpretacja



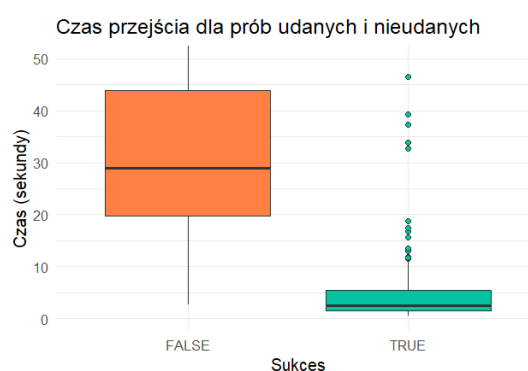
[htbp]

Rysunek 2: Sukces w zależności od poziomu trudności.

Interpretacja: Wykres przedstawia, jak zmienia się skuteczność algorytmu w zależności od wybranej trudności labiryntu. Dla poziomu „łatwy” odnotowano ponad 95% udanych przejść, przy „średni” skuteczność spadła do około 88%, a w przypadku „trudny” tylko około 72% prób zakończyło się sukcesem. Oznacza to, że wzrost złożoności struktury labiryntu (rozmiar, liczba ślepych zaułków, rozgałęzień) znacząco wpływa na efektywność działania algorytmu ewolucyjnego. Jest to zgodne z założeniem, że większa przestrzeń poszukiwań utrudnia zbieżność.



Rysunek 3: Czas a poziom trudności

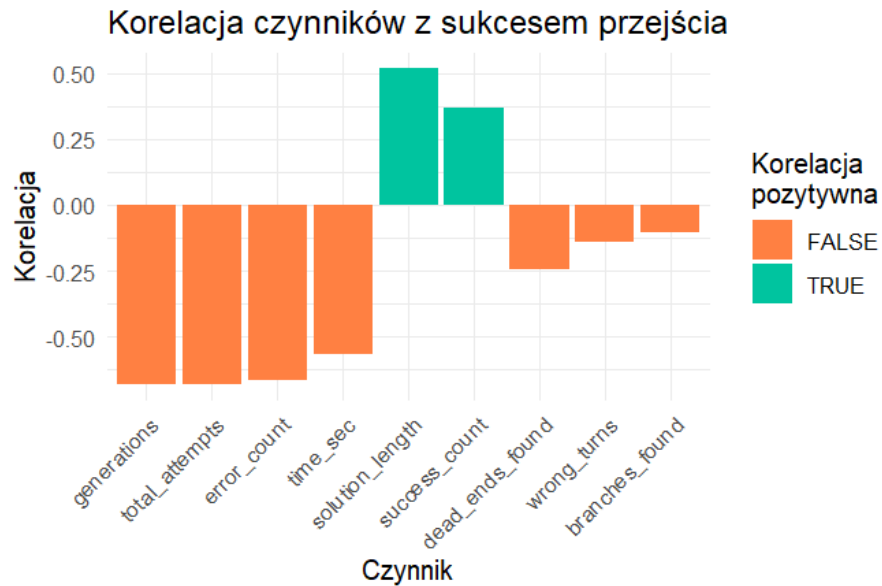


Rysunek 4: Czas a sukces przejścia

Interpretacja: Wraz ze wzrostem poziomu trudności rośnie zarówno mediana czasu przejścia, jak i jego wariancja. Dla poziomu „łatwy” większość rozwiązań mieści się w przedziale 1–2 sekundy, podczas gdy w przypadku „trudny” czasy przejścia bywają znacznie dłuższe i bardziej rozproszone. Dłuższe czasy wskazują na konieczność wykonania większej liczby pokoleń i analizowania większej liczby możliwych ścieżek przez osobniki.

Próby zakończone sukcesem są zazwyczaj znacznie szybsze i bardziej spójne czasowo (niska zmienność), natomiast próby nieudane cechują się znacznie dłuższym

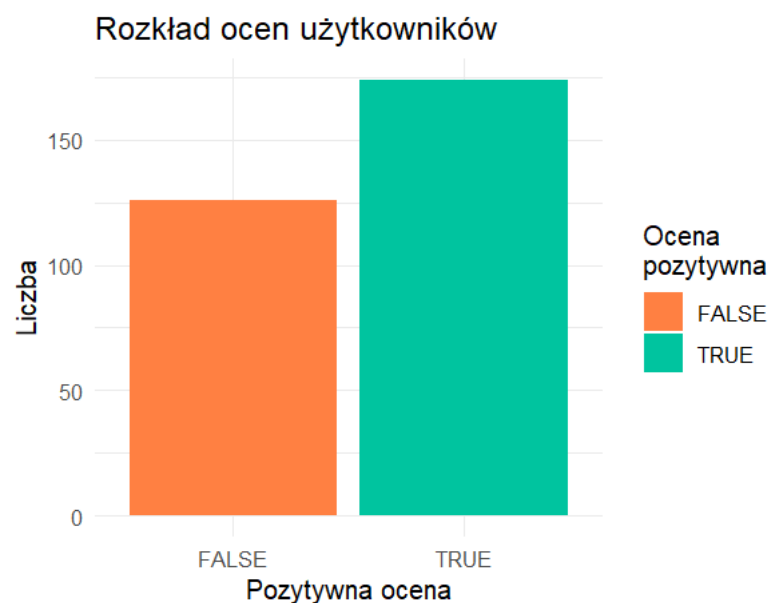
czasem oraz większą rozpiętością wyników. Oznacza to, że skuteczne strategie poruszania się po labiryncie są bardziej deterministyczne i szybciej prowadzą do celu, natomiast osobniki, które błędzą, poświęcają więcej czasu na eksplorację bez rezultatu.



[htbp]

Rysunek 5: Współczynniki korelacji pomiędzy sukcesem a poszczególnymi cechami rozwiązania.

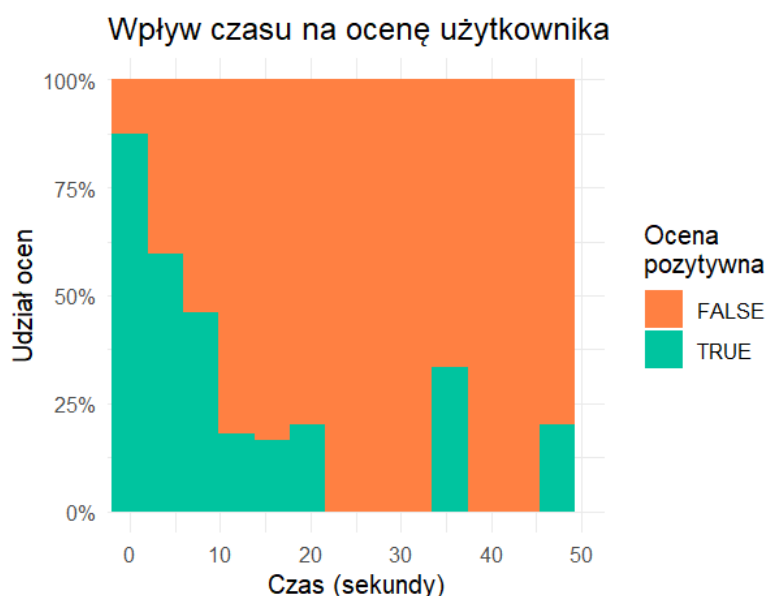
Interpretacja: Na wykresie pokazano siłę związku pomiędzy sukcesem a wybranymi zmiennymi ilościowymi. Najsilniejsza negatywna korelacja występuje dla liczby błędów, liczby pokoleń i czasu przejścia, co oznacza, że im więcej błędów i dłuższa ścieżka ewolucyjna, tym mniejsze prawdopodobieństwo sukcesu. Jedyną zmienną o pozytywnej korelacji z sukcesem jest długość rozwiązania - co może wskazywać, że bardziej rozbudowane, ale poprawne ścieżki są preferowane w stosunku do krótkich, ale nieefektywnych tras.



[htbp]

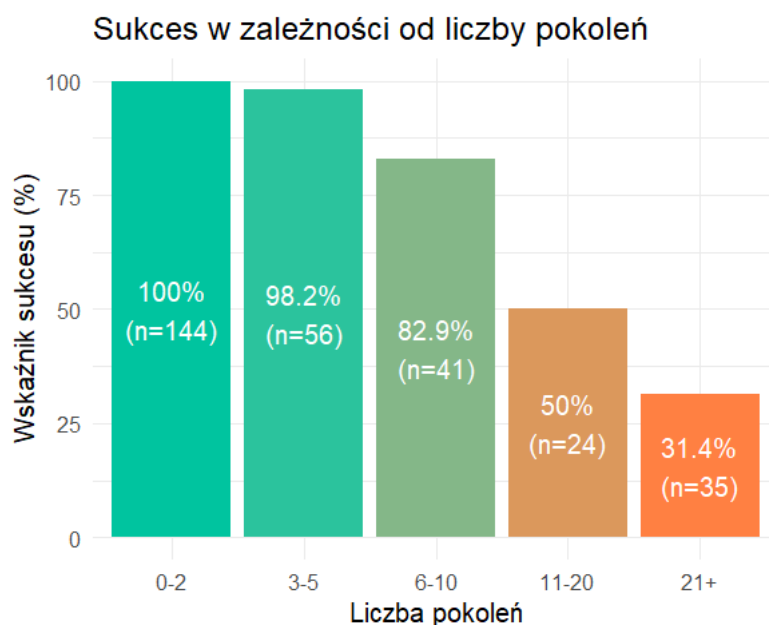
Rysunek 6: Rozkład pozytywnych i negatywnych ocen użytkowników.

Interpretacja: Użytkownicy oceniający działanie programu zazwyczaj uznawali jego działanie za poprawne - około 60% ocen miało charakter pozytywny. Sugeruje to, że w większości przypadków znalezione rozwiązania spełniają intuicyjne kryteria poprawności (np. brak błędnych skrętów, prosta i krótka ścieżka, szybkie wykonanie).



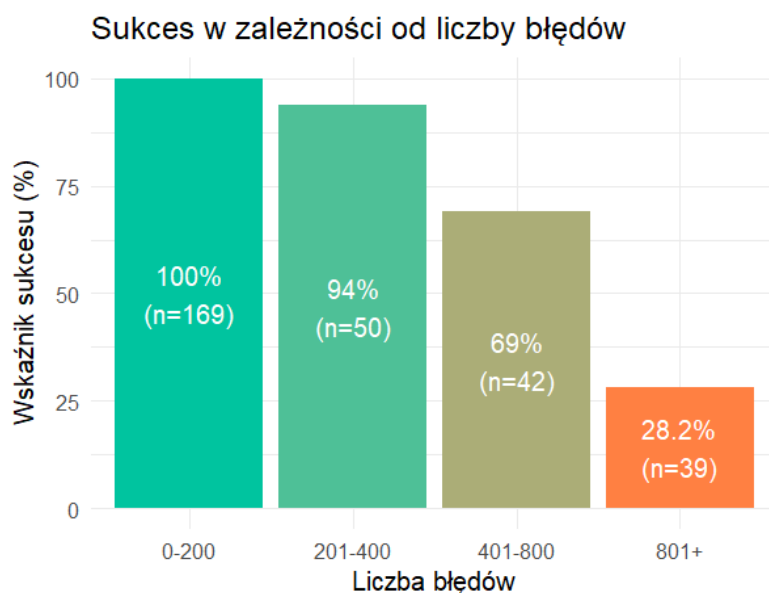
Rysunek 7: Wpływ czasu przejścia na ocenę użytkownika.

Interpretacja: Im szybciej zakończyło się rozwiązanie labiryntu, tym większe było prawdopodobieństwo, że użytkownik oceni je jako poprawne. Dla czasów poniżej 5 sekund pozytywne oceny dominowały, natomiast przy czasach przekraczających 40 sekund większość ocen była negatywna. Może to wynikać z faktu, że użytkownicy utożsamiają czas działania z efektywnością i jakością algorytmu.



Rysunek 8: Wskaźnik sukcesu w zależności od liczby pokoleń.

Interpretacja: Aż 100% prób zakończonych sukcesem miało miejsce w ciągu pierwszych 2-3 pokoleń. Wraz z dalszą ewolucją skuteczność stopniowo spada. Może to wskazywać, że optymalne rozwiązania są znajdowane bardzo szybko (wczesna zbieżność), a długie ewolucje wiążą się raczej z przypadkami trudnymi, gdzie algorytm błędzi lub wpada w lokalne minima.



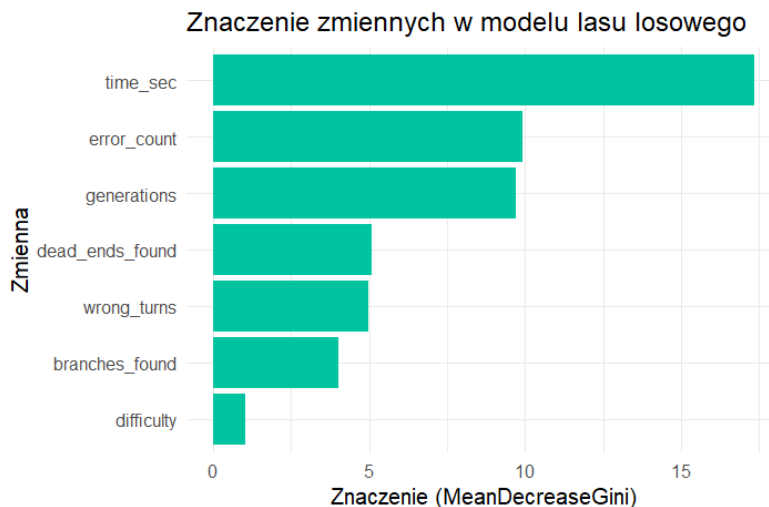
Rysunek 9: Sukces w zależności od liczby błędów.

Interpretacja: Wskaźnik sukcesu wyraźnie maleje wraz ze wzrostem liczby błędów popełnionych przez osobnika. Dla ścieżek z mniej niż 200 błędami sukces wynosił 100%, natomiast przy liczbie przekraczającej 800 - tylko 28%. Wskazuje to na kluczowe znaczenie jakości strategii poruszania się, szczególnie unikania ślepych zaułków i błędnych skrętów.



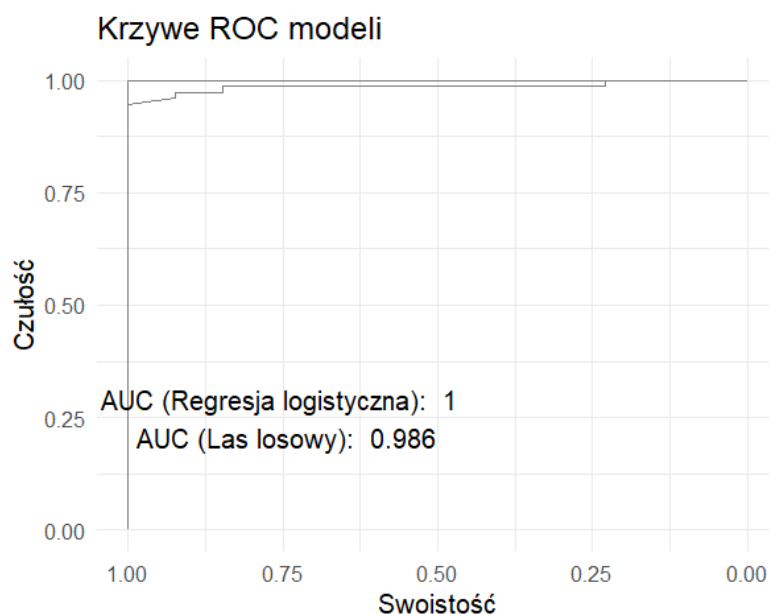
Rysunek 10: Wpływ przerw na sukces rozwiązywania.

Interpretacja: Wszystkie sukcesy miały miejsce w próbach nieprzerwanych. Oznacza to, że każde przerywanie ewolucji (np. przez limit czasu lub błędne zakończenie) skutecznie uniemożliwia osiągnięcie celu. To silny argument za mechanizmami monitorującymi stabilność algorytmu i zapobiegającymi jego przedwczesnemu zatrzymaniu.



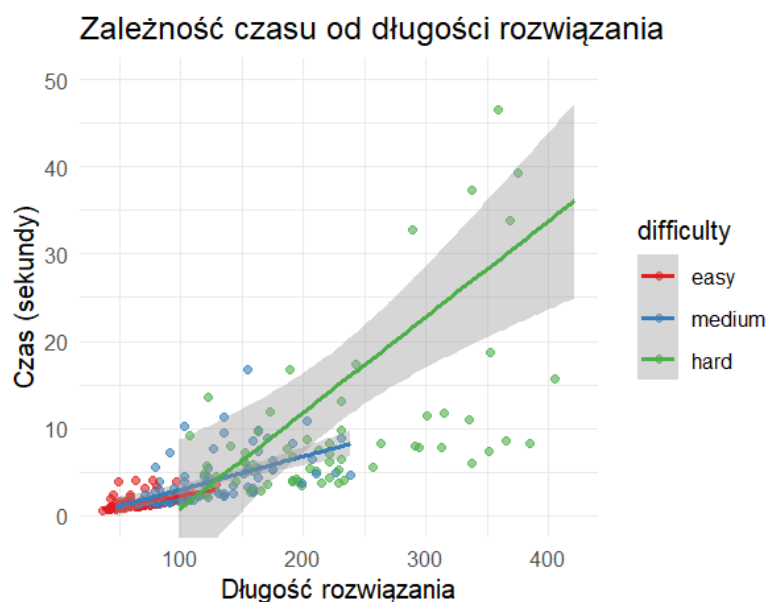
Rysunek 11: Znaczenie zmiennych w modelu lasu losowego.

Interpretacja: Najbardziej wpływowymi zmiennymi predykcyjnymi były: `time_sec`, `error_count` oraz `generations`. To oznacza, że czas, błędy i pokolenia są nie tylko kluczowe dla sukcesu, ale także najważniejsze w budowie modelu predykcyjnego. Zmienna `difficulty` miała relatywnie niewielki wpływ - co może wynikać z tego, że pozostałe zmienne lepiej odzwierciedlają złożoność ścieżki.



Rysunek 12: Krzywe ROC modeli predykcyjnych.

Interpretacja: Oba modele - regresja logistyczna oraz las losowy - uzyskały bardzo wysoką jakość predykcji sukcesu. Wartości AUC wyniosły odpowiednio 1.00 (regresja) i 0.986 (las). Mimo minimalnej przewagi regresji, model lasu może być preferowany ze względu na odporność na nadmiarowość danych i możliwość interpretacji ważności cech.



Rysunek 13: Zależność czasu przejścia od długości rozwiązania, z podziałem na poziomy trudności.

Interpretacja: Długość ścieżki koreluje dodatnio z czasem rozwiązania. Dla trudnych labiryntów wzrost długości jest silnie związany ze wzrostem czasu. Dla łatwych i średnich poziomów czas wzrasta wolniej. Może to świadczyć o tym, że w prostych labiryntach długość nie jest główną barierą, natomiast w złożonych — wydłużenie trasy znacząco obciąża algorytm.

12.2 Podsumowanie obserwacji eksperymentalnych

W tabeli sporządzono skrótowe podsumowanie wniosków wyciągniętych z wyżej zaprezentowanych wykresów.

Tabela 1: Podsumowanie kluczowych obserwacji eksperymentalnych

Obserwacja	Wniosek
Sukces zależny od trudności	Im wyższa trudność labiryntu, tym niższy wskaźnik sukcesu (spadek z 96% do 72%).
Czas a sukces	Udane próby są znacznie szybsze niż nieudane - skuteczne ścieżki są krótsze i mniej złożone.
Korelacje	Najsilniejsze negatywne korelacje z sukcesem mają liczba błędów, czas i liczba pokoleń.
Oceny użytkowników	Pozytywne oceny dominują przy czasie poniżej 10 s, negatywne - powyżej 30 s.
Generacje i sukces	Większość sukcesów pojawia się w ciągu 5 pierwszych generacji; dalsze ewolucje są mniej skuteczne.
Błędy	Liczba błędów powyżej 800 praktycznie eliminuje szansę na sukces.
Przerwania	Próby przerwane mają 0% skuteczności - przerwanie procesu ewolucji jest krytyczne.
Predyktory sukcesu	Najważniejsze cechy wg modelu RF: czas, liczba błędów, liczba pokoleń.
Skuteczność modeli	Oba modele (regresja i las losowy) mają bardzo wysokie AUC (≥ 0.98).
Długość rozwiązania a czas	Dłuższe rozwiązania skutkują dłuższym czasem przejścia, zwłaszcza przy wyższej trudności.

Zestawienie potwierdza, że największy wpływ na skuteczność mają jakość strategii i liczba błędów. Modele predykcyjne osiągają niemal perfekcyjną dokładność, co czyni je potencjalnie użytecznym komponentem przyszłych wersji systemu.

13 Zakończenie

W pracy zaprezentowano zastosowanie ewolucyjnego podejścia do automatycznego generowania programów rozwiązujących labirynty. Stworzony system, oparty na programowaniu genetycznym, został przetestowany w środowisku symulacyjnym z różnymi poziomami trudności oraz poddany wszechstronnej analizie ilościowej i jakościowej.

Otrzymane wyniki potwierdzają skuteczność podejścia ewolucyjnego: przy sprzyjających warunkach (niskim poziomie błędów i krótkim czasie działania) algorytmy osiągały wysoką skuteczność, sięgającą nawet 96% dla łatwych labiryntów. Dodatkowo, zbudowane modele predykcyjne wykazały się bardzo wysoką trafnością ($AUC \approx 0,99$).

Przeprowadzona analiza pokazuje, że programowanie genetyczne może być użytecznym narzędziem do rozwiązywania problemów eksploracyjnych i

planistycznych, a jego dalszy rozwój otwiera ciekawe możliwości zarówno badawcze, jak i aplikacyjne.

Literatura

- [1] E. Figielska, *Algorytmy ewolucyjne i ich zastosowania*, Zeszyty Naukowe 81-92.
https://zeszyty-naukowe.wysi.edu.pl/zeszyty/zeszyt1/Algorytmy_Ewolucyjne_I_Ich_Zastosowania.pdf
- [2] https://en.wikipedia.org/wiki/Genetic_algorithm
- [3] https://en.wikipedia.org/wiki/Genetic_programming
- [4] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- [5] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
- [6] Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing* (2nd ed.). Springer.
- [7] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.