UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Michał Dettlaff
nr albumu: 164 622

Zastosowanie algorytmów genetycznych w projektowaniu optymalnego układu klawiatury

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
prof. zw. dr hab. inż. Sławomir Wierzchoń

Gdańsk 2011

Streszczenie

Projektowanie optymalnego, ergonomicznego układu klawiatury można potraktować jako problem optymalizacji kombinatorycznej, do rozwiązania którego można zastosować znane metaheurystyki. W poniższej pracy wykorzystany jest do tego celu algorytm genetyczny, w którym poszczególne osobniki utożsamiane są z konkretnymi układami klawiszy, a funkcja przystosowania odzwierciedla ilość pracy potrzebną do przepisania tekstu za pomocą danego układu. W definicji funkcji przystosowania został uwzględniony szereg czynników, które mają wpływ na szybkość oraz ergonomię pisania na danym układzie klawiatury. Zakodowanie osobników algorytmu genetycznego w postaci permutacji pozwoliło na zastosowanie operatorów krzyżowania oraz mutacji opracowanych dla problemów stosujących podobną reprezentację chromosomu, takich jak problem komiwojażera. Wpływ zastosowania poszczególnych operatorów na jakość końcowego rozwiązania został zbadany eksperymentalnie, po czym w ostatecznym algorytmie zostały zastosowane najlepsze z nich. Jako dane testowe algorytmu zostały wykorzystane zbiory tekstów w języku polskim i angielskim. Ze względu na różnice statystyczne pomiędzy tymi zbiorami, algorytm został uruchomiony osobno dla każdego z nich. W wyniku działania algorytmu zostają utworzone osobne układy klawiatury zoptymalizowane pod kątem języka polskiego oraz angielskiego. Następnie są one porównane ze standardowymi układami klawiatury (QWERTY, Dvorak), zaprojektowanymi bez udziału komputera, oraz rozwiązaniami znalezionymi przez inne algorytmy metaheurystyczne. Układy klawiszy znalezione przez zastosowany algorytm genetyczny uzyskują wyraźnie lepsze oceny od dotychczas istniejących, zwłaszcza w przypadku wyników dla języka polskiego.

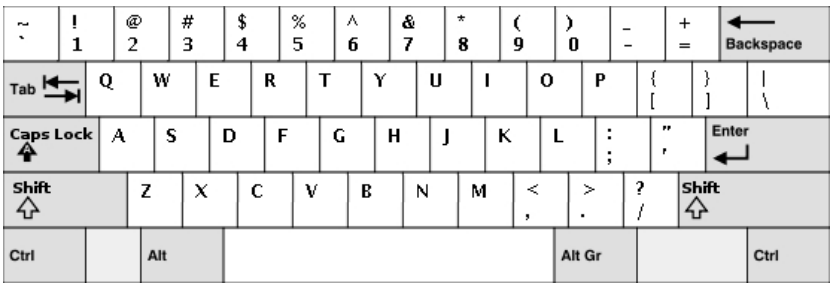
Słowa kluczowe

algorytmy genetyczne, optymalizacja kombinatoryczna, optymalizacja wielokryterialna, klawiatura, Dvorak, QWERTY

Spis treści

Wprowadzenie

Standardowy układ klawiatury (rys. 1), mimo swojej wszechobecności oraz pozostawania w użyciu od ponad 100 lat, jest pod wieloma względami nieoptymalny. Układ ten nazywany jest układem QWERTY (od pierwszych liter w górnym rzędzie w jego amerykańskiej wersji) lub układem Scholesa i został zaprojektowany w latach 70 XIX wieku przez Charles'a Lethama Sholesa. QWERTY zastąpił używany wcześniej układ alfabetyczny i miał na celu zapobieganie zacinaniu się mechanizmu maszyny do pisania, jeżeli użytkownik pisał zbyt szybko [1]. Z tego względu, litery które często występowały obok siebie w języku angielskim, takie jak *i* oraz *e*, zostały umieszczone po przeciwnych stronach klawiatury. Dzięki temu, belki na których znajdowały się klawisze nie zderzały się ze sobą, co zmniejszało prawdopodobieństwo zacięcia się. Mimo że problem zacinania się maszyny został szybko wyeliminowany i to jeszcze na długo przed nadejściem klawiatury komputerowej, układ klawiatury QWERTY pozostaje w powszechnym użyciu aż do dzisiaj i stanowi de facto standard.



Rysunek **1**. Standardowy układ klawiatury QWERTY

Źródło: Opracowanie własne

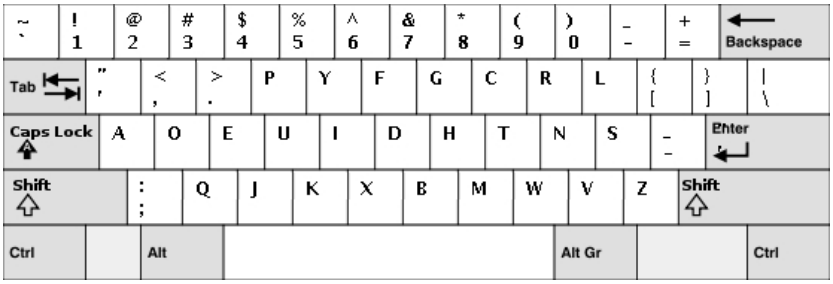
Spośród układów alternatywnych wobec QWERTY największą popularnością cieszy się klawiatura Dvoraka (Dvorak Simplified Keyboard). Została ona zaprojektowana w 1936 r. przez dr Augusta Dvoraka. W przeciwieństwie do QWERTY, podczas jej projektowania nie były już brane pod uwagę ograniczenia mechaniczne

maszyn do pisania, ze względu na postępy w ich produkcji. Głównym celem było natomiast umożliwienie efektywnego pisania oraz łatwość nauki. Dvorak sformułował szereg zasad, którymi kierował się podczas projektowania swojego układu klawiszy i które miały pozwolić na osiągnięcie tych celów [2]. Zgodnie z tymi zasadami, optymalna klawiatura powinna:

- Uwzględniać statystyczną analizę najczęściej występujących par liter.
- Zmniejszać prawdopodobieństwo popełniania błędów poprzez uczynienie pisania bardziej ”naturalnym” i wygodnym, poprzez zminimalizowanie ruchów palców pomiędzy rzędami klawiszy.
- Zwiększyć użycie środkowego rzędu (tzw. *home row*) oraz uderzeń pozwalających na jak najszybsze pisanie (takich które wykorzystują pisanie obiema rękami na zmianę, zamiast tą samą ręką przez dłuższy czas).
- W większym stopniu wykorzystywać prawą rękę, która jest silniejsza w przypadku większości użytkowników.
- Być łatwiejsza w nauce, przyczyniać się do popełniania mniejszej ilości błędów, oraz minimalizować wysiłek związany z pisaniem.

Warto zwrócić uwagę, że cechy statystyczne o których mowa w pierwszym z powyższych punktów są zależne od analizowanego języka. Dvorak przeprowadził analizę dla języka angielskiego, w związku z czym projektowanie klawiatury dla innych języków wiązałoby się z ponownym przeprowadzeniem analizy, a wynikowy układ klawiszy mógłby się znacznie różnić.

Jedną z głównych różnic pomiędzy układem QWERTY a klawiaturą Dvoraka jest przeniesienie najczęściej używanych klawiszy do środkowego rzędu. Dla przykładu, za pomocą samego środkowego rzędu klawiatury Dvoraka można napisać około 400 słów w języku angielskim, natomiast układ QWERTY pozwala w analogiczny sposób napisać tylko około 100 słów [3]. Charakterystyczną cechą klawiatury Dvoraka jest umiejscowienie wszystkich samogłosek po tej samej stronie. Wpływa to korzystnie na alternację rąk, ponieważ w języku angielskim samogłoski najczęściej nie występują obok siebie. Klawiatura Dvoraka jest łatwiejsza w nauce i pozwala na około 10% szybsze pisanie [1].



Rysunek **2**. Układ klawiatury Dvorak Simplified Keyboard

Źródło: Opracowanie własne

Wspomniane układy klawiatury zostały zaprojektowane ręcznie, bez udziału komputera. Podjęto też wiele prób rozwiązania problemu szukania optymalnego klawiatury (Keyboard Arrangement Problem - KAP) z użyciem algorytmów ewolucyjnych oraz innych metaheurystyk. Wagner [4] wygenerował układy klawiatury dla języka angielskiego, niemieckiego oraz francuskiego, stosując algorytm mrówkowy (Ant Colony Optimization) oraz szereg kryteriów ergonomicznych. Yin oraz Su [5] sformułowali problem ogólnie dla różnych typów klawiatur (General Keyboard Arrangement Problem - GKAP) oraz zastosowali do jego rozwiązania algorytm optymalizacji rojowej (Cyber Swarm Optimization). Deshwal oraz Deb [6] zaproponowali nowy projekt klawiatury dla języka Hindi, stosując algorytm genetyczny z reprezentacją układu klawiszy jako czterowymiarowej tablicy.

Podobne algorytmy stosowano również dla niestandardowych klawiatur, na przykład w telefonach komórkowych, na których pisze się z użyciem jednego lub dwóch palców [7]. Niniejsza praca dotyczy KAP rozumianego jako poszukiwanie optymalnego układu klawiszy dla standardowej klawiatury, zoptymalizowanego pod kątem techniki pisania bezwzrokowego (*touchtyping*). Pisanie bezwzrokowe zobrazowane jest na ilustracji poniżej.

W technice bezwzrokowej, w domyślnej pozycji palce lewej ręki znajdować się powinny nad klawiszami $asdf$, a prawej nad $jkl;$. Po wciśnięciu któregoś z pozostałych klawiszy, palec powinien powrócić na swoją standardową pozycję w środkowym rzędzie (*home row*). Kciuki zajmują się altem i spacją. Kolorami oznaczono,

Rysunek **3**. Użycie palców w pisaniu bezwzrokowym

Źródło: Opracowanie własne

którymi palcami powinno się uderzać poszczególne klawisze, na przykład: lewy serdeczny wciska x, s, w i 3, a prawy wskazujący n, m, h, j, y itd.

W problemie KAP, naszym zadaniem jest optymalizacja funkcji oceny danego układu klawiatury. Istnieje wiele klasycznych metod optymalizacji, które zazwyczaj dzieli się na analityczne, przeglądowe oraz losowe. Wśród metod analitycznych wyróżniamy metody bezpośrednie oraz pośrednie. W metodach pośrednich poszukujemy lokalnych ekstremów funkcji rozwiązując układ równań. Lokalny charakter tych metod oraz konieczność obliczania pochodnych sprawiają, że nie nadają się one do rozwiązywania problemów takich jak KAP, gdzie optymalizowana funkcja jest nieciągła i nieregularna.

Do bezpośrednich metod optymalizacji należą algorytmy wzrostu. W algorytmach wzrostu rozważa się sąsiedztwo aktualnie wybranego rozwiązania, czyli zbiór rozwiązań jakie mogą powstać na skutek niewielkiej modyfikacji danego rozwiązania. Następnie, spośród rozwiązań sąsiednich wybiera się to posiadające najkorzystniejszą wartość funkcji oceny i zastępuje się nim rozwiązanie aktualne, pod warunkiem że nowe rozwiązanie ma korzystniejszą wartość od aktualnego. Tego rodzaju algorytmy wzrostu są w stanie znaleźć jedynie lokalne optimum funkcji, ponieważ po osiągnięciu optimum lokalnego nie są już w stanie go opuścić. Algorytm zaczyna działanie od losowo wybranego punktu przestrzeni, w związku z czym prawdopodobieństwo że znalezione optimum lokalne jest jednocześnie optimum globalnym jest bardzo małe, jeżeli funkcja posiada wiele optimów. Nieco lep-

sze wyniki daje stosowanie wariantu iterowanego, w którym uruchamiamy algorytm wiele razy z różnych punktów startowych.

Ocena jakości układu klawiatury opiera się na statystycznych cechach tekstu i ma charakter eksperymentalny. W związku z tym funkcja oceny posiada nieregularny kształt, zawierający wiele optimów lokalnych. Dlatego też metody przeszukiwania lokalnego, takie jak algorytm wzrostu, mają znikome szanse na znalezienie optimum globalnego. Potrzeba zatem użyć algorytmu optymalizacji globalnej. Jednym z najprostszych przykładów takiego algorytmu jest metoda enumeracyjna, w której przeglądane są po kolei wszystkie punkty przestrzeni poszukiwań. Ponieważ obszar poszukiwań jest skończony, metoda ta gwarantuje znalezienie rozwiązania. Jednakże bardzo duża ilość wszystkich możliwych rozwiązań (wynosząca $30!$ dla 30 klawiszy) sprawia, że jest ona nieskuteczna w praktyce, z wyjątkiem bardzo niewielkich problemów. Podobna sytuacja ma miejsce w przypadku metod stosujących techniki błądzenia losowego.

Dobry algorytm optymalizacji globalnej powinien cechować się dobrymi zdolnościami eksploracyjnymi oraz eksploatacyjnymi. Eksploracyjność oznacza zdolność do znajdowania regionów przestrzeni poszukiwań zawierających dobre rozwiązania, a eksploatacyjność to zdolność znajdowania najlepszych rozwiązań w niewielkich, "dobrych" obszarach przestrzeni poszukiwań. Algorytmy metaheurystyczne to rodzaj algorytmów optymalizacyjnych ogólnego przeznaczenia, których celem jest zapewnienie dobrej wydajności pod względem obu tych cech. Cechują się one znacznie większą odpornością od klasycznych metod optymalizacji. Jednym ze sposobów na osiągnięcie tego celu jest stosowanie metod populacyjnych, co znajduje zastosowanie między innymi w algorytmach genetycznych. Dzięki temu rozwiązania mogą wpływać na siebie wzajemnie i wymieniać między sobą informacje, co nie następuje w metodach optymalizacji operujących na pojedynczej instancji rozwiązania (bardziej szczegółowe omówienie algorytmów genetycznych można znaleźć w kolejnym rozdziale niniejszej pracy). Zaletą wielu metaheurystyk, w tym również algorytmów genetycznych jest to, że najczęściej nie wymagają one do działania żadnych informacji pomocniczych o rozwiązywanym problemie, a tylko funkcję oceny rozwiązania. Mimo to, algorytmy metaheurystyczne są w stanie znaleźć satysfakcjonujące rozwiązania w rozsądnym czasie nawet dla dużych instancji problemów. Większość metaheurystyk ma charakter stochastyczny, jednak

nie oznacza to że stosowane jest przeszukiwanie losowe, ale tylko pewne aspekty algorytmu są zrandomizowane. Znalezienie optimum globalnego nie jest gwarantowane, ale stosowane są rozmaite techniki zmniejszające prawdopodobieństwo zbiegania do optimów lokalnych.

Dla wielu problemów, takich jak poszukiwanie minimalnego drzewa spinającego lub minimalnej ścieżki w grafie, istnieją dokładne algorytmy obliczające rozwiązanie w czasie wielomianowym. W takich przypadkach stosowanie metaheurystyk nie jest uzasadnione. Algorytmów takich nie opracowano jednak dla problemu KAP, a wieloaspektowy charakter problemu określenia ergonomiczności układu klawiatury sprawia, że ich opracowanie byłoby bardzo trudne, jeżeli nie niemożliwe.

Ponieważ przestrzeń poszukiwań ma rozmiar skończony, KAP jest przykładem problemu optymalizacji kombinatorycznej. Do klasycznych przykładów zagadnienia optymalizacji kombinatorycznej należy m.in. problem komiwojażera (Travelling Salesman Problem - TSP), który często stosowany jest do porównywania wydajności algorytmów metaheurystycznych. Algorytmy genetyczne były wielokrotnie stosowane z powodzeniem do tego rodzaju problemów, w tym również do TSP [8] [9] [10], co pozwala na przypuszczenie że uzyskane wyniki będą zadowalające również dla KAP. Ponadto, istnieje wiele operatorów genetycznych które zaprojektowano z myślą o ścieżkowej reprezentacji TSP [11]. Stosując podobną reprezentację problemu w przypadku KAP, możliwe jest ich ponowne wykorzystanie, bez konieczności projektowania genetycznych operatorów od nowa. Należące do nich operatory krzyżowania stworzono głównie z myślą o przekazywaniu informacji o kolejności elementów lub ich pozycjach bezwzględnych między osobnikami. Szczególnie ten drugi rodzaj operatorów jest korzystny dla KAP, ponieważ ocena układu klawiatury zależy głównie od bezwzględnych pozycji poszczególnych klawiszy.

W dotychczasowej literaturze algorytmy genetyczne zostały już zastosowane do KAP w pracy Goettla [3]. Użyty algorytm oraz funkcja przystosowania były jednak stosunkowo proste. Wypróbowany został tylko jeden rodzaj mutacji, polegający na zamianie elementów permutacji miejscami, oraz jeden rodzaj krzyżowania. Zbiór tekstów stanowiący dane testowe składał się tylko z trzech pozycji. W niniejszej pracy, standardowa forma algorytmu genetycznego została rozwinięta po-

przez zastosowanie modelu elitarnego z różnymi wielkościami elity, zastosowanie operatora selekcji dostosowanego do optymalizacji wielokryterialnej, oraz współbieżne obliczanie funkcji przystosowania. W funkcji przystosowania uwzględniono siedem różnych kryteriów w porównaniu do czterech wziętych pod uwagę w pracy Goettla. W danych testowych uwzględniono dziesięć różnorodnych tekstów, co powinno wpłynąć na zwiększenie wiarygodności wyników. Wypróbowanych zostało sześć operatorów mutacji oraz cztery operatory krzyżowania, ich wydajność została porównana i w końcowym algorytmie zostały użyte najkorzystniejsze z nich. W literaturze można znaleźć wiele przykładów na poszukiwanie optymalnego układu klawiszy dla języka angielskiego, francuskiego lub niemieckiego. W niniejszej pracy po raz pierwszy podjęta została próba zaprojektowania optymalnej klawiatury dla języka polskiego.

Kolejny rozdział poświęcony jest ogólnemu wprowadzeniu do tematyki algorytmów genetycznych, z opisem schematu algorytmu oraz podstawowych operatorów genetycznych. Następnie, zostanie opisany algorytm genetyczny przystosowany do problemu KAP, z odpowiednio dobraną funkcją przystosowania oraz operatorami mutacji, selekcji i krzyżowania. W kolejnym rozdziale zostaną omówione wyniki działania tego algorytmu dla tekstów w języku polskim oraz angielskim. Uzyskane wyniki porównane są z tymi uzyskanymi przez znane układy klawiatury zaprojektowane ręcznie, oraz układy wygenerowane za pomocą innych algorytmów. Układy wygenerowane przez algorytm uzyskują wyraźną przewagę, zwłaszcza w przypadku wyników dla języka polskiego. W zakończeniu pracy przedstawione zostaną uzyskane konkluzje oraz plany dalszych badań.

ROZDZIAŁ **1**

Algorytmy genetyczne

Algorytmy genetyczne to rodzaj algorytmów heurystycznych pozwalających na efektywne przeszukiwanie przestrzeni rozwiązań w poszukiwaniu optymalnego rozwiązania. Inspiracją dla algorytmów genetycznych jest ewolucja naturalna, w szczególności mechanizmy dziedziczenia oraz doboru naturalnego. Wykorzystują one biologicznie inspirowane operatory, do których należą rozrodczość, selekcja, mutacja oraz krzyżowanie. Autorem pierwszej publikacji o algorytmach genetycznych która uzyskała szerokie uznanie jest John Holland [12].

Algorytmy genetyczne należą do metod populacyjnych, w których operujemy na pewnym zmieniającym się zbiorze (populacji) rozwiązań. Daje to znaczącą przewagę nad prostszymi algorytmami wspinaczkowymi, w których zachowujemy tylko jedno rozwiązanie w danym momencie. Zachowywanie populacji rozwiązań zmniejsza prawdopodobieństwo przedwczesnej zbieżności w lokalnym optimum, zwiększając możliwości eksploracyjne algorytmu. Inne skrajne podejście, w którym przeszukujemy jednocześnie całą przestrzeń rozwiązań, jest również mało skuteczne ze względu na zbyt duży rozmiar tej przestrzeni dla większości interesujących problemów. Algorytm genetyczny unika tego problemu poprzez dobranie rozmiaru populacji znacznie mniejszego od rozmiaru całej przestrzeni poszukiwań.

Schemat algorytmu genetycznego można przedstawić za pomocą następującego pseudokodu:

```
i := 0
Zainicjuj populację początkową P(i)
Oceń osobniki z populacji początkowej
Powtarzaj dopóki nie jest spełniony warunek zakończenia:
   i := i + 1
   Wybierz osobniki z P(i-1) i utwórz nową populację P(i)
```

```
Zmień osobniki z P(i)
Oceń osobniki z P(i)
```

Warunkiem zakończenia może być upłynięcie określonej ilości iteracji, lub znalezienie rozwiązania które jest wystarczająco bliskie optimum. Rolę operacji `Zmień osobniki` w algorytmach genetycznych pełnią mutacja oraz krzyżowanie.

Algorytm genetyczny operuje na osobnikach w zakodowanej postaci, którą najczęściej nazywa się chromosomem. Każdy chromosom reprezentuje potencjalne rozwiązanie zadania. W klasycznych algorytmach genetycznych, chromosom ma postać ciągu bitów o stałej długości.

Pierwszym krokiem algorytmu jest wygenerowanie populacji początkowej. Powinna się ona cechować możliwie dużą różnorodnością, dlatego najczęściej stosowaną metodą jej tworzenia jest generowanie chromosomów w sposób losowy. Czasem w populacji początkowej umieszcza się też specjalnie przygotowane osobniki, o których spodziewamy się że leżą w pobliżu optimum.

Zaletą klasycznych algorytmów genetycznych jest fakt, że uniwersalna reprezentacja chromosomów jako ciągów bitów pozwala na ponowne wykorzystanie tych samych operatorów mutacji oraz krzyżowania. Jedyną rzeczą jaką trzeba zrobić aby zastosować algorytm genetyczny do nowego problemu, jest zaimplementowanie funkcji oceny oraz zakodowanie potencjalnych rozwiązań w postaci ciągu bitów. Takie podejście jest jednak problematyczne dla wielu zastosowań. Mutacja oraz krzyżowanie mogą prowadzić do powstawania osobników które stanowią rozwiązania nieprawidłowe, które muszą zostać odrzucone.

Algorytmy genetyczne znajdują szerokie zastosowania w zadaniach przeszukiwania oraz optymalizacji. Należą do nich między innymi: wytyczanie trasy połączeń kablowych, harmonogramowanie, sterowanie adaptacyjne, rozgrywanie gier, modelowanie poznawcze, zadania transportowe, zadanie komiwojażera, sterowanie optymalne oraz optymalizacja obsługi zapytań w bazach danych [8].

1.1. Selekcja

Wybór osobników do nowo tworzonych populacji jest procesem analogicznym do doboru naturalnego. W ujęciu Karola Darwina, największe szanse przetrwania mają osobniki najlepiej przystosowane do środowiska. Wpływ środowiska jest w algorytmach genetycznych modelowany przez funkcję oceny lub funkcję przystosowania. Ponieważ problemy optymalizacji zwykle definiuje się w kategoriach poszukiwania minimum, przyjmiemy konwencję zgodnie z którą mniejsza wartość funkcji oceny oznacza lepsze przystosowanie. Dlatego też w operacji wyboru (selekcji) osobników do nowej populacji powinny być faworyzowane osobniki o niższej wartości funkcji przystosowania.

Jedną z najprostszych implementacji operacji selekcji jest selekcja proporcjonalna, w której osobniki wybierane są z prawdopodobieństwem proporcjonalnym do wartości funkcji przystosowania. Wadą selekcji proporcjonalnej jest tendencja do zbyt szybkiego zdominowania populacji przez najbardziej przystosowane osobniki. Dlatego zaproponowano wiele alternatywnych implementacji operatora selekcji, takich jak selekcja rangowa, turniejowa, lub podwójna turniejowa.

Selekcja turniejowa to bardzo prosty algorytm selekcji, będący jednocześnie jednym z najpopularniejszych [13]. W selekcji turniejowej, wybieramy najlepiej przystosowanego osobnika spośród losowo wybranego podzbioru populacji, bez powtórzeń. Algorytm można dostosować poprzez zmianę parametru t, czyli rozmiaru wybieranego podzbioru. Innymi zaletami algorytmu są jego prostota oraz niezależność od szczegółów implementacji funkcji przynależności.

Pseudokod:

```
WEJŚCIE: populacja, t - rozmiar turnieju
Best := wybierz bez zwracania osobnika z populacji
Powtarzaj t - 1 razy:
   Next := wybierz bez zwracania osobnika z populacji
   jeżeli Next ma korzystniejszą wartość przystosowania niż Best:
      Best := Next
zwróć Best
```

Najczęściej wybieraną wartością parametru t w przypadku algorytmów genetycznych jest 2 [13].

Sam operator selekcji nie jest wystarczającym warunkiem do tego, aby wartość funkcji oceny poprawiała się w kolejnych populacjach. Potrzebny jest operator wprowadzający zmienność oraz innowacje do populacji. W klasycznych algorytmach genetycznych, służą do tego operacje mutacji oraz krzyżowania.

1.2. Mutacja

Operacja mutacji powinna powodować jedynie niewielką zmianę w położeniu rozwiązania w pejzażu dostosowań. To znaczy, niewielka modyfikacja chromosomu powinna wpływać w niewielkim stopniu na wartość funkcji przystosowania. W klasycznym algorytmie genetycznym, mutacja polega na losowej zmianie elementu chromosomu z jednakowym (niewielkim) prawdopodobieństwem na wartość przeciwną (z jedynki na zero i na odwrót) [14].

1.3. Krzyżowanie

Jednym z prostszych operatorów krzyżowania dla binarnej reprezentacji chromosomu jest krzyżowanie punktowe. W krzyżowaniu punktowym, wybieramy na chromosomie losowo punkt przecięcia. Aby utworzyć pierwszego potomka, kopiujemy geny leżące na lewo od punktu przecięcia z jednego rodzica, a geny leżące na prawo z drugiego rodzica. Drugi potomek powstaje analogicznie poprzez skopiowanie najpierw prawej, a następnie lewej strony. Przykładowo, w wyniku zastosowania krzyżowania punktowego z punktem przecięcia między drugą i trzecią pozycją do pary rodziców:

$$(a_1a_2a_3a_4a_5), (b_1b_2b_3b_4b_5)$$

otrzymamy następujące osobniki potomne:

$$(a_1a_2b_3b_4b_5), (b_1b_2a_3a_4a_5)$$

Wadą krzyżowania punktowego jest fakt, że geny leżące na sąsiadujących pozycjach będą rozdzielane z większym prawdopodobieństwem od genów leżących na bardziej odległych pozycjach. Jednym ze sposobów na zniwelowanie tego problemu jest zwiększenie liczby punktów przecięcia, w wyniku czego otrzymamy krzyżowanie wielopunktowe.

ROZDZIAŁ **2**

Opis algorytmu

2.1. Reprezentacja chromosomu

Układ klawiatury można zakodować jako ciąg binarny, opisując każdy klawisz za pomocą określonej ilości bitów. Niestety, stosowanie klasycznych operatorów genetycznych spowoduje, że zdecydowana większość powstałych w ten sposób rozwiązań będzie nieprawidłowych. Przykładowo, jeżeli użyjemy trzech bitów do opisania każdego klawisza klawiatury o 40 klawiszach, tylko jedna z każdych 1016 dowolnie wybranych struktur 120-bitowych jest strukturą o właściwej konfiguracji klawiszy [15].

W związku z powyższym, zastosowana zostanie permutacyjna reprezentacja chromosomu oraz specjalnie dla niej dostosowane operatory mutacji oraz krzyżowania. Układ klawiatury reprezentowany jest przez permutację 30 klawiszy. 26 klawiszy oznaczanych jest literami alfabetu łacińskiego A-Z, a pozostałe odpowiadają znakom ”,”, ”.”, ”?” oraz ”;”. Wynika z tego, że liczba wszystkich możliwych układów klawiatury wynosi

$$30! \simeq 2.65 * 10^{32}$$

co stanowi rozmiar przestrzeni poszukiwań na której będzie działał algorytm genetyczny. Zbiór wszystkich znaków wchodzących w skład permutacji oznaczymy jako A.

Klawisze podzielone są na trzy grupy, gdzie pierwsze 10 oznacza klawisze górnego rzędu klawiatury, następne 10 klawisze rzędu środkowego, a ostatnie 10 - dolnego rzędu.

Dla przykładu, układ klawiatury QWERTY jest reprezentowany przez następującą permutację:

$$(qwertyuiopasdf ghjkl; zxcvbnm, .?)$$

2.2. Funkcja przystosowania

Naszym celem jest znalezienie układu klawiatury, który przyczyni się do zwiększenia prędkości pisania oraz zminimalizowania zmęczenia palców użytkownika. Inne pożądane cechy to minimalizowanie ilości błędów oraz łatwość nauki danego układu klawiatury. Dlatego też te cele będą brane pod uwagę podczas projektowania funkcji przystosowania algorytmu genetycznego.

W kolejnych podrozdziałach zostaną opisane kryteria, które składają się na funkcję przystosowania. Zostały opracowane głównie w oparciu o prace Wagnera [4] oraz Dvoraka [2]. Ostateczna postać funkcji przystosowania jest sumą ważoną wyników z poszczególnych kryteriów.

Do obliczania oceny układu klawiatury wykorzystywane są zbiory tekstów, stanowiących możliwie reprezentatywną próbkę z danego języka. Wynikowy układ powinien być jak najlepiej przystosowany do przepisywania podanego tekstu, dlatego użyte zostaną osobne zbiory tekstów dla języka polskiego oraz angielskiego. Naiwnym sposobem na obliczenie przystosowania byłaby symulacja przepisywania tekstu, wymagająca analizy całości tekstu za każdym razem od nowa podczas obliczania funkcji przystosowania. Takie podejście jest jednak zbyt kosztowne obliczeniowo, ponieważ złożoność funkcji przystosowania wyniosłaby $O(n)$, gdzie n stanowiące długość tekstu byłoby wartością rzędu 10^5 lub 10^6.

Z tego powodu funkcja przystosowania nie będzie obliczana na podstawie oryginalnego tekstu, ale jego cech statystycznych, które będą obliczane tylko raz przed uruchomieniem algorytmu. W ten sposób czas obliczania funkcji przystosowania nie będzie zależny od długości tekstu (jego złożoność wyniesie $O(1)$). Cechy statystyczne tekstu zostaną opisane za pomocą dwóch funkcji:

$$f_m : M \rightarrow \mathrm{R}$$

$$f_d : D \rightarrow \mathrm{R}$$

Do opisania tych funkcji przydatne będą dwie definicje. Jako *monograf* będziemy rozumieć rodzaj zdarzenia polegającego na naciśnięciu klawisza któremu odpowiada element chromosomu. Wpływ pozostałych klawiszy zostanie pominięty - przykładowo, zakładamy że podczas wpisywania wielkich liter klawisz SHIFT

zostaje uderzony przeciwną ręką do tej, która uderza klawisz z literą. *Diagraf* natomiast definiujemy jako parę monografów następujących zaraz po sobie. Przykładowo, w tekście o treści *barbaz* możemy wyróżnić 4 monografy: *b*, *a*, *r* oraz *z*, oraz 4 diagrafy: *ba*, *ar*, *rb*, oraz *az*. Dziedzinę funkcji f_m będziemy określać jako zbiór wszystkich monografów M, a dziedzinę funkcji f_d jako zbiór wszystkich możliwych diagrafów D.

$$M = \{a : a \in A\}$$

$$D = \{ab : a \in A, b \in A\}$$

f_m opisuje częstotliwości występowania monografów w zbiorze tekstów, a f_d częstotliwości występowania diagrafów. Przez częstotliwość występowania rozumiemy procentowy udział ilości wystąpień danego monografu (diagrafu) w sumie ilości wystąpień wszystkich monografów (diagrafów). Przykładowo, dla tekstu *barbaz* $f_m(z) = \frac{1}{6}$, a $f_d(ba) = \frac{2}{5}$.

2.2.1. Użycie palców

Pierwszym czynnikiem branym pod uwagę będzie rozkład pracy na poszczególne palce. Chcemy, aby najczęściej używane były najsilniejsze i najdłuższe palce, a najmniej używane palce najsłabsze.

W tym celu zdefiniujemy optymalny rozkład użycia, który podajemy w oparciu o [4]. f_1^{opt} do f_4^{opt} oznaczają udział palców lewej ręki w przepisywaniu tekstu, od małego palca do wskazującego, natomiast f_5^{opt} do f_8^{opt} analogicznie udział palców prawej ręki, od wskazującego do małego.

f_1^{opt}	f_2^{opt}	f_3^{opt}	f_4^{opt}	f_5^{opt}	f_6^{opt}	f_7^{opt}	f_8^{opt}
4.4%	12.3%	15.8%	17.5%	17.5%	15.8%	12.3%	4.4%

Czynnik funkcji przystosowania odpowiadający użyciu palców przyjmie następującą postać, odzwierciedlającą rozbieżność pomiędzy rozkładem otrzymanym a optymalnym:

$$p_{finger} = \sum_{i=1}^{8} (f_i^{opt} - f_i)^2$$

2.2.2. Użycie rzędów klawiszy

Podczas pisania techniką bezwzrokową, pozycją domyślną jest utrzymywanie palców nad klawiszami w środkowym rzędzie (*home row*). Z tego powodu optymalny układ klawiatury powinien zawierać najczęściej używane znaki w środkowym rzędzie. Znaki z górnego oraz dolnego rzędu wymagają ruchu palca z domyślnego rzędu środkowego, a następnie powrotu na pozycję domyślną, co jest mniej optymalne. Klawisze z górnego rzędu są przy tym nieco łatwiej dostępne od tych z dolnego.

Górny, środkowy oraz dolny rząd klawiszy oznaczymy odpowiednio jako: R_1, R_2, R_3. Jako optymalny rozkład częstotliwości użycia poszczególnych rzędów przyjmiemy rozkład układu klawiatury, w którym 10 najczęściej używanych klawiszy należy do R_1, 10 kolejnych do R_2, a 10 najmniej używanych klawiszy do R_3. Częstotliwości użycia obliczane są na podstawie stosowanego zbioru tekstów.

Przykładowe wartości: $f_{R_1} = 50\%$, $f_{R_2} = 35\%$, $f_{R_3} = 15\%$

Biorąc powyższe kryteria pod uwagę, czynnik funkcji przystosowania dotyczący położenia znaku w danym rzędzie przyjmie następującą postać:

$$p_{row} = \sum_{i=1}^{3} (f_{R_i}^{opt} - f_{R_i})^2$$

2.2.3. Alternacja rąk

Czynnikiem sprzyjającym szybkiemu i komfortowemu pisaniu jest unikanie pisania kolejnych znaków tą samą ręką. Jest to jedna z głównych zasad która została zastosowana przy projektowaniu klawiatury Dvoraka, gdzie w tym celu wszystkie samogłoski umieszczono z lewej strony klawiatury.

Liczbową reprezentacją tej zasady będzie następujące wyrażenie:

$$p_{hand\_alter} = \sum_{d \in D_{sh}} f_d(d)$$

Gdzie $D_{sh} \in D$ oznacza zbiór diagrafów pisanych tą samą ręką w danym układzie klawiszy.

2.2.4. Alternacja palców

Podobnie jak w poprzednim punkcie, nie jest korzystne pisanie kolejnych znaków tym samym palcem. Dodatkowe utrudnienie stanowi sytuacja, w której ten sam palec musi przebyć większą odległość. Dlatego też częstotliwość występowania diagrafów zostanie dodatkowo pomnożona przez współczynnik odległości $dist$.

$$p_{finger\_alter} = \sum_{d \in D_{sf}} f_d(d) dist(d)$$

gdzie zbiór $D_{sf} \in D$ oznacza zbiór diagrafów które pisane są tym samym palcem.

Jako odległość $dist$ zostanie przyjęta odległość Manhattan:

$$dist(d) = |c_2 - c_1| + |r_2 - r_1|$$

gdzie c_1 oraz r_1 stanowią współrzędne odpowiednio rzędu i kolumny pierwszego znaku z diagrafu d, a c_2 oraz r_2 współrzędne drugiego znaku diagrafu d.

2.2.5. Duże ruchy palcami tej samej ręki

W wypadku użycia tej samej ręki do naciśnięcia kolejno dwóch klawiszy, należy unikać sytuacji w której klawisze te znajdują się daleko od siebie. Dlatego też podczas obliczania tego współczynnika będą brane pod uwagę diagrafy wpisywane tą samą ręką, ale przy użyciu różnych palców, gdzie pionowa odległość jest większa niż jeden rząd (zbiór ten oznaczymy jako $D_{step} \in D$). W zależności od tego które palce zostaną użyte, zostaną przypisane odpowiednie wagi zgodnie z poniższą tabelą [4].

	wskazujący	środkowy	serdeczny	mały
wskazujący	0	5	8	6
środkowy	5	0	9	7
serdeczny	8	9	0	10
mały	6	7	10	0

Wzór przyjmie postać:

$$p_{step} = \sum_{d \in D_{step}} K(d) f_d(d)$$

gdzie $K(d)$ oznacza współczynnik wagowy przypisany zgodnie z tabelą.

2.2.6. Inboard stroke flow

Dla diagrafów pisanych z użyciem tej samej ręki, bardziej naturalny jest kierunek ich wpisywania prowadzący do środka, czyli w kierunku od małego palca do wskazującego (tzw. *inboard stroke flow*). Czynnikowi temu będzie odpowiadał następujący wzór, w którym sumujemy częstotliwości występowania diagrafów prowadzących w mniej korzystnym kierunku (zbiór takich diagrafów oznaczymy poniżej jako $D_{isf} \in D$):

$$p_{isf} = \sum_{d \in D_{isf}} f_d(d)$$

2.2.7. Użycie rąk

Wysiłek związany z pisaniem powinien być rozłożony proporcjonalnie na obie ręce. Dvorak [2] oraz część innych autorów [3] sugeruje, aby faworyzować w niewielkim stopniu prawą, silniejszą rękę. Przyjmiemy tutaj, że w idealnym przypadku prawa ręka powinna wykonywać 52,5% pracy związanej z pisaniem, karząc osobniki które odbiegają od tego rozkładu. Czynnik ten wyniesie zatem:

$$p_{hand\_usage} = (f_{left} - 47,5\%)^2 + (f_{right} - 52,5\%)^2$$

gdzie f_{left} oznacza procentowy udział znaków wpisanych lewą ręką, a f_{right} analogicznie prawą ręką.

Ostatecznie, funkcja przystosowania przyjmie postać:

$$p = p_{finger} w_{finger} + p_{row} w_{row} + p_{hand\_alter} w_{hand\_alter} + p_{finger\_alter} w_{finger\_alter} +$$

$$p_{step} w_{step} + p_{isf} w_{isf} + p_{hand\_usage} w_{hand\_usage}$$

gdzie *w* są wagami poszczególnych składowych. Wagi dobrane są eksperymentalnie w ten sposób, aby poszczególne kryteria miały zbliżony wpływ na ostateczną wartość funkcji przystosowania.

2.3. Operatory krzyżowania

2.3.1. Krzyżowanie cykliczne (CX)

Istotną cechą jaką powinien posiadać algorytm krzyżowania do naszych zastosowań, jest zachowywanie absolutnych pozycji elementów w chromosomie. Cechę tę posiada operator krzyżowania cyklicznego [16]. Średnio połowa absolutnych pozycji elementów z obu rodziców zostaje w nim zachowana.

W krzyżowaniu cyklicznym, wybieramy losowy element z losowo wybranego rodzica i umieszczamy go na tej samej pozycji w chromosomie potomnym. Procedura ta musi być odpowiednio zmodyfikowana, aby zapewnić że chromosom potomny stanowi prawidłową permutację (a zatem prawidłowy układ klawiatury).

Jeżeli okazuje się, że danego elementu z chromosomu rodzica nie można umieścić w chromosomie potomnym, zostaje wybrany element drugiego rodzica. Jeżeli ten element również nie może zostać wybrany, zostaje wybrany losowo jeden z pozostałych elementów, które są dozwolone. Kolejne elementy wybrane z jednego rodzica tworzą jeden cykl, stąd nazwa algorytmu.

Rozważmy przykład. Niech dane będą następujące osobniki rodzicielskie:

$$(abcdefgh), (bdfhgeca)$$

Na początku musimy wybrać pozycję, z której będzie pochodził pierwszy element. Załóżmy, że będzie to pierwsza. Na pierwszą pozycję chromosomu potomka możemy teraz wybrać a lub b, w zależności od tego który rodzic zostanie wybrany. Niech będzie to rodzic pierwszy. Chromosom osobnika potomnego wyglądać będzie następująco (gwiazdką zaznaczono pozycje jeszcze nieprzypisane):

$$(a * * * * * * *)$$

Rozpatrzmy teraz element na ostatniej pozycji. Możemy wybrać a lub h, ale jeżeli wybierzemy a, wynikowy chromosom nie będzie już prawidłową permutacją.

Dlatego wybieramy h.

$$(a * * * * * * h)$$

Geny na czwartej oraz drugiej pozycji muszą również zostać wybrane z pierwszego rodzica, aby uniknąć wybrania po raz kolejny tych samych elementów.

$$(ab * d * * * h)$$

Ponieważ wszystkie do tej pory wybrane elementy należą do tego samego rodzica, możemy powiedzieć że tworzą one cykl. Rozpatrzmy teraz elementy na trzeciej pozycji. Możemy wybrać dowolny z nich (c lub f). Jeżeli wybierzemy element z drugiego rodzica (f), wtedy elementy na piątej, szóstej i siódmej pozycji również mogą pochodzić z drugiego rodzica, tworząc drugi cykl. Ostatecznie, otrzymujemy następującego osobnika potomnego:

$$(abfdgech)$$

2.3.2. Krzyżowanie z częściowym odwzorowaniem (PMX)

Podobnie jak krzyżowanie cykliczne, krzyżowanie z częściowym odwzorowaniem [17] zachowuje częściowo absolutne pozycje elementów rodzicielskich. Fragment chromosomu jednego z rodziców jest przepisywany na potomka, a pozostała część chromosomu jest tworzona na podstawie odwzorowań.

Na początku, wybierane są losowo dwa punkty w chromosomie. Fragment znajdujący się pomiędzy nimi nazywać będziemy *sekcją odwzorowania*. Sekcja odwzorowania drugiego rodzica przepisywana jest na te same pozycje do pierwszego potomka, a sekcja odwzorowania pierwszego rodzica - do drugiego potomka. Następnie, pozostałe pozycje potomka przepisywane są z odpowiadającego mu rodzica, chyba że przepisywany element już istnieje w chromosomie potomka - w takim wypadku brany jest inny element, na podstawie odwzorowania określonego przez sekcje odwzorowania.

Dla przykładu, niech dane będą następujące osobniki rodzicielskie:

$$(abcdefgh), (cgeafhbd)$$

Załóżmy, że wylosowano punkty pomiędzy trzecim i czwartym, oraz szóstym i siódmym elementem chromosomu, co daje sekcję odwzorowania od czwartego do

szóstego elementu. Po przepisaniu sekcji odwzorowania, osobniki potomne mają następującą postać:

$$(***afh**), (***def**)$$

Sekcje odwzorowania (afh) oraz (def) definiują odwzorowania: $a \leftrightarrow d$, $f \leftrightarrow e$ oraz $h \leftrightarrow f$. Przypiszmy teraz brakujące elementy pierwszego potomka. Pierwszym elementem pierwszego rodzica jest a. Nie możemy jednak przepisać go na pierwszą pozycję potomka, ponieważ element ten występuje już w permutacji. Możemy jednak przypisać d, ponieważ zgodnie z odwzorowaniami a jest odwzorowany na d.

$$(d**afh**)$$

Elementy na drugiej, trzeciej oraz siódmej pozycji można przepisać bezpośrednio, ponieważ nie powodują one powstawania nieprawidłowej permutacji.

$$(dbcafhg*)$$

Aby uzupełnić ostatnią pozycję, trzeba zastosować odwzorowanie dwukrotnie. Nie możemy przepisać h, więc najpierw stosujemy odwzorowanie $h \rightarrow f$. f także występuje już w permutacji, dlatego odwzorowujemy $f \rightarrow e$.

$$(dbcafhge)$$

Warto zwrócić uwagę, że pozycje sześciu z ośmiu elementów zostały zachowane. Drugi potomek powstaje analogicznie, przez stosowanie tych samych reguł.

2.3.3. Krzyżowanie bazujące na porządku (POS)

Operator POS [18] również należy do kategorii operatorów, które mają na celu zachowanie tych samych pozycji dla jak największej ilości elementów chromosomów rodzicielskich. Na początku wybierany jest losowo zbiór pozycji na chromosomie. Elementy na tych pozycjach są przepisywane na te same pozycje z drugiego rodzica do pierwszego potomka i odwrotnie. Niech dane będą pozycje trzecia, czwarta i piąta, oraz osobniki rodzicielskie:

$$(abcdef), (edbfac)$$

Po przepisaniu elementów na danych pozycjach, otrzymujemy osobniki potomne:

$$(**bf*c), (**cd*f)$$

Pozostałe pozycje pierwszego potomka wypełniane są kolejnymi elementami pierwszego rodzica, przy czym elementy powtarzające się są pomijane. Element a zostaje przepisany na tą samą, pierwszą pozycję, b i c są pomijane, d zostaje przepisany na kolejną wolną pozycję (drugą), a e na pozycję piątą. Stosując analogiczną procedurę dla drugiego potomka, otrzymujemy w wyniku:

$$(adbfec), (ebcdaf)$$

2.3.4. Krzyżowanie naprzemienne (AP)

Ten typ krzyżowania polega na wybieraniu elementów na przemian z pierwszego i drugiego rodzica [11]. Jeżeli dany element znajduje się już w chromosomie osobnika potomnego, jest on zwyczajnie pomijany. Przykładowo, niech będą dane osobniki rodzicielskie $(abcdefgh)$ oraz $(dchbefga)$. Zaczynając od pierwszej pozycji, dobieramy najpierw element z pierwszej pozycji pierwszego rodzica, a następnie z pierwszej pozycji drugiego rodzica. Czynność tę powtarzamy dla drugiej pozycji, otrzymując:

$$(adbc****)$$

Następny w kolejności element c z pierwszego rodzica występuje już w osobniku potomnym, dlatego pomijamy go i wstawiamy następny element, jakim jest h z drugiego rodzica. Ostatecznie, otrzymamy następującego osobnika potomnego:

$$(adbchefg)$$

Drugi osobnik potomny powstaje przez przeprowadzenie analogicznej procedury z odwróconą kolejnością osobników rodzicielskich:

$$(dacbhefg)$$

2.4. Operatory mutacji

2.4.1. Operator zamiany (EM)

Operator zamiany [11] polega na wybraniu losowo dwóch elementów permutacji, a następnie zamienieniu ich miejscami. Na przykład, jeżeli w permutacji $(abcdefgh)$ wybierzemy element drugi i piąty, otrzymamy w efekcie:

$$(a\boldsymbol{e}cd\boldsymbol{b}fgh)$$

2.4.2. Zamiana z przemieszczeniem (DM)

Pierwszym krokiem w metodzie zamiany z przemieszczeniem [8] jest wybranie losowego odcinka permutacji. Załóżmy, że zaczęliśmy od permutacji $(abcdefgh)$ i wybraliśmy odcinek $(defg)$. Następnie, należy usunąć wybrany odcinek z permutacji i umieścić go w niej ponownie na losowo wybranej pozycji. W naszym przykładzie, po jego usunięciu permutacja przyjmie postać:

$$(abch)$$

Załóżmy, że w tak powstałej permutacji wylosowano pozycję pomiędzy elementami a oraz b. W takim wypadku, ostateczny wynik mutacji przyjmie następującą postać:

$$(a\boldsymbol{defg}bch)$$

2.4.3. Prosta inwersja (SIM)

Operator prostej inwersji [12] polega na odwróceniu kolejności elementów w wybranym fragmencie chromosomu. Przykładowo, niech dany będzie chromosom:

$$(abcdefgh)$$

Najpierw należy wybrać dwa punkty cięcia - niech będą to punkty pomiędzy elementami a i b, oraz e i f. Następnie fragment chromosomu pomiędzy punktami cięcia zostaje odwrócony, dając w efekcie następujący wynik:

$$(a\boldsymbol{edcb}fgh)$$

2.4.4. Mutacja z mieszaniem podlist (SM)

Podobnie jak w poprzednim przypadku, ten rodzaj mutacji również rozpoczyna od wyboru dwóch punktów cięcia definiujących fragment chromosomu [18]. Następnie, dokonywana jest losowa permutacja elementów na wybranym fragmencie. Załóżmy, że wybrany został fragment (abc) z poniższego chromosomu:

$$(abcd)$$

W takim wypadku istnieje sześć możliwych rezultatów, do których należą:

$$(\boldsymbol{abc}d), (\boldsymbol{acb}d), (\boldsymbol{bac}d), (\boldsymbol{bca}d), (\boldsymbol{cab}d), (\boldsymbol{cba}d)$$

2.4.5. Inwersja (IVM)

Operator IVM [19] działa podobnie jak DM, z tą różnicą, że dodatkowo odwracana jest kolejność elementów w wybranym fragmencie chromosomu. Niech dany będzie chromosom $(abcdefgh)$. Załóżmy, że podobnie jak w przypadku operatora DM wybraliśmy odcinek $(defg)$, a na jego nową pozycję wybrany został punkt pomiędzy elementami a oraz b. Przed ponownym umieszczeniem odcinka w chromosomie, najpierw zostanie odwrócona kolejność jego elementów:

$$(defg) \rightarrow (gfed)$$

Spowoduje to powstanie następującego wyniku końcowego:

$$(abc\boldsymbol{gfed}h)$$

2.4.6. Mutacja ze wstawianiem (ISM)

Ten rodzaj mutacji [8] polega na wybraniu losowego elementu chromosomu, usunięciu go, a następnie wstawieniu ponownie na innej pozycji. Na przykład, po wybraniu elementu szóstego z chromosomu $(abcdefgh)$, usunięciu go, a następnie wstawieniu ponownie na pozycji drugiej otrzymamy następujący wynik:

$$(a\boldsymbol{f}bcdegh)$$

Warto zwrócić uwagę, że jeśli wybrane punkty będą znajdowały się na odległych pozycjach, zmienią się absolutne pozycje wszystkich elementów znajdujących się pomiędzy nimi. Może mieć to bardzo negatywny wpływ na jakość rozwiązania, zwłaszcza ze względu na czynnik stopnia użycia poszczególnych palców.

2.5. Optymalizacja dla wielu kryteriów

W funkcji oceny uwzględnianych jest 7 różnych kryteriów, co sprawia że rozwiązywany problem należy do klasy problemów optymalizacji wielokryterialnej. W tego rodzaju zagadnieniach nie chcemy optymalizować pojedynczej funkcji, ale poszukujemy rozwiązania, które będzie optymalne pod względem wartości funkcji celu (oceny) dla każdego ze stosowanych kryteriów. Jednym ze sposobów podejścia do tego problemu jest połączenie wielu funkcji celu f_i w jedną funkcję celu F. Pozwala na to metoda ważonych celów:

$$F(x) = \sum_{i=1}^{k} w_i f_i(x)$$

gdzie w_i są wagami poszczególnych kryteriów, a k to ilość celów. Trudność polega tutaj na doborze wag, ponieważ często poszczególnych kryteriów nie daje się w łatwych sposób porównać.

Kolejnym sposobem, który nie wymaga podania wartości wag, jest użycie jednego z wielokryterialnych operatorów selekcji [13]. Możemy uporządkować kryteria pod względem ważności, a następnie zastosować *wielokryterialną leksykograficzną selekcję turniejową*. W tego rodzaju selekcji, porównujemy osobniki ze względu na kolejne kryteria zaczynając od tego mającego najwyższy priorytet, a kończąc na tym mającym najniższy priorytet. Jeżeli jeden z osobników przeważa nad innym pod względem danego kryterium, jest on preferowany i porównywanie zostaje zakończone. Innym sposobem jest porównywanie osobników ze względu na kryterium wybrane za każdym razem w sposób losowy (*wielokryterialna proporcjonalna selekcja turniejowa*).

Innym podejściem niewymagającym łączenia poszczególnych funkcji celów w jedną funkcję jest stosowanie pojęcia *dominacji Pareto*. Mówimy że rozwiązanie A dominuje w sensie Pareto nad rozwiązaniem B jeżeli $f(A)$ jest co najmniej tak

dobre jak $f(B)$ dla wszystkich celów, i lepsze dla co najmniej jednego celu (k to ilość celów):

$$(\forall i)\,(f_i(A) \leq f_i(B)) \wedge (\exists i)\,(f_i(A) < f_i(B))\,, \quad 1 \leq i \leq k$$

Przez rozwiązanie *zdominowane* rozumiemy takie, dla którego istnieje inne rozwiązanie które dominuje nad nim. Rozwiązanie *niezdominowane* to takie, dla którego nie istnieje inne rozwiązanie nad nim dominujące.

Istnieją algorytmy genetyczne wykorzystujące pojęcie dominacji Pareto. Jednym z nich jest NSGA [20] (Non-Dominated Sorting Genetic Algorithm), w którym osobniki porządkowane są pod względem dominacji. W tym algorytmie, najpierw oznacza się wszystkie osobniki niezdominowane w aktualnej populacji. Osobnikom tym nadajemy rangę równą 1. Następnie, oznaczamy wszystkie osobniki niezdominowane w zbiorze osobników jeszcze nieoznaczonych i nadajemy im rangę 2. Krok ten powtarzany jest dopóki wszystkim osobnikom w populacji nie zostaną przypisane rangi. Dzięki zastosowaniu tej procedury, wszystkim osobnikom lokalnie niezdominowanym można przypisać ten sam potencjał reprodukcyjny, zgodnie z posiadanymi przez nie rangami. Algorytm NSGA-II stanowi kontynuację algorytmu NSGA i różni się on zastosowaniem pewnego rodzaju elitarności, gdzie zapamiętujemy zbiór najlepszych osobników znalezionych do tej pory.

Algorytm HLGA (Hajela and Lin's Weighting-based Genetic Algorithm) stanowi rozwinięcie metody ważonych celów. Również stosuje on mnożenie funkcji celów f_i przez wagi w_i, ale różni się poprzez przypisywanie każdemu osobnikowi w populacji osobnego wektora wag. Pozwala to na znalezienie wielu paretooptymalnych rozwiązań podczas jednego uruchomienia algorytmu. Wadą HLGA, podobnie jak w przypadu innych metod opartych na wektorze wag, jest pomijanie rozwiązań leżących w niewypukłych obszarach paretooptymalnych.

Algorytm FFGA (Fonseca and Fleming's Multiobjective Genetic Algorithm) przypisuje osobnikom rangi, których wartość jest równa ilości osobników przez które dany osobnik jest zdominowany. Przeprowadzana jest selekcja rangowa, przy czym lepszą ocenę otrzymują osobniki z niższymi rangami. Podobnie jak w przypadku algorytmu NSGA, osobniki o jednakowej randze posiadają jednakowy potencjał reprodukcyjny.

Istnieje wiele innych algorytmów genetycznych wykorzystujących pojęcie dominacji Pareto. NPGA (Niched Pareto Genetic Algorithm) łączy selekcję turniejową z dominowaniem Pareto. SPEA (Strength Pareto Evolutionary Algorithm) wykorzystuje pojęcie *siły Pareto*, rozumianej jako liczba osobników w populacji zdominowanych przez danego osobnika.

Innym przykładem algorytmu genetycznego stworzonego z myślą o optymalizacji wielokryterialnej jest VEGA [21] (Vector Evaluated Genetic Algorithm). Główny pomysł polega w nim na podzieleniu populacji na podpopulacje o jednakowej wielkości, gdzie w każdej z nich odbywa się selekcja ze względu na osobne kryterium. Krzyżowanie natomiast może przekraczać granice podpopulacji, aby umożliwić wymianę informacji pomiędzy nimi. Zaletą algorytmu VEGA jest łatwość implementacji, ale wadą jest tendencja do odrzucania osobników pośrednich, które osiągają dobre wyniki ze względu na wszystkie kryteria, ale nie osiągają bardzo wysokiego rezultatu dla żadnego z nich.

W tej pracy zastosowane zostanie prostsze podejście, polegające na zastosowaniu funkcji przystosowania będącej sumą ważoną funkcji celu w połączeniu z jednym z wielokryterialnych operatorów selekcji. Zostanie on opisany szczegółowo w kolejnej sekcji *Selekcja*.

2.6. Selekcja

2.6.1. Wielokryterialna większościowa selekcja turniejowa

Wielokryterialna większościowa selekcja turniejowa (Multiobjective majority tournament selection - MMTS [13]) jest to rodzaj selekcji turniejowej, w której faworyzowane są osobniki posiadające jak największą ilość kryteriów, których wartość jest bardziej korzystna niż w pozostałych osobnikach.

Selekcję tą można przedstawić za pomocą następującego pseudokodu:

```
WEJŚCIE: populacja, t - rozmiar turnieju, n - ilość kryteriów
Best := wybierz bez zwracania osobnika z populacji
Powtarzaj t - 1 razy:
   Next := wybierz bez zwracania osobnika z populacji
```

```
   c := 0
   dla i := 1 do n powtarzaj:
      jeżeli p_i dla Next < p_i dla Best:
         c := c + 1
      jeżeli p_i dla Next > p_i dla Best:
         c := c - 1
   jeżeli c > 0:
      Best := Next
   jeżeli c = 0 ∧ p dla Next < p dla Best:
      Best := Next
zwróć Best
```

Wartość parametru n będzie w końcowym algorytmie równa 7, ponieważ tyle kryteriów uwzględniono w funkcji przystosowania. c oznacza ilość kryteriów, dla których dany osobnik jest lepszy od aktualnie najlepszego znalezionego osobnika. W algorytmie tym została wprowadzona jedna modyfikacja w porównaniu do standardowej wersji. Mianowicie, jeżeli żaden osobnik nie uzyska przewagi pod względem c, brana jest pod uwagę wartość przystosowania p obliczana z użyciem wag.

Rozważmy dla przykładu dwa osobniki A i B, dla których są określone cztery funkcje celu $f_i, 1 \leq i \leq 4$. Niech wartości funkcji celu dla A będą równe $f_1 = 1; f_2 = 1; f_3 = 1; f_4 = 1$, a dla osobnika B: $f_1 = 0,9; f_2 = 0,9; f_3 = 0,9; f_4 = 1,5$. Stosując standardową selekcję turniejową, preferowany będzie osobnik B, dla którego suma wszystkich funkcji celu jest większa, ale wpływ składnika f_4 jest nieproporcjonalnie duży. Natomiast zgodnie z selekcją MMTS, zostanie wybrany osobnik A, w którym trzy z czterech kryteriów mają korzystniejsze wartości, oraz wpływ poszczególnych kryteriów jest bardziej zrównoważony. Żaden z osobników A i B nie dominuje nad pozostałym w sensie Pareto, ale zauważmy że w osobniku A wystarczy poprawić wynik tylko jednej funkcji celu, aby był on dominujący.

2.7. Ostateczna postać zastosowanego algorytmu

Zastosowany algorytm można opisać za pomocą następującego pseudokodu:

```
Oblicz cechy statystyczne zbioru tekstów
Zainicjuj początkową populację losowo utworzonymi osobnikami
Oblicz przystosowanie osobników
Powtarzaj zadaną ilość pokoleń:
   Utwórz nową, pustą populację osobników
   Dodaj osobniki elitarne do nowej populacji
   Dopóki wielkość nowej populacji jest mniejsza od poprzedniej:
      Oblicz przystosowanie osobników istniejącej populacji
      Wybierz parę osobników za pomocą operatora selekcji
      Zastosuj z zadanym prawdopodobieństwem operację krzyżowania
      wobec wybranych osobników
      Zastosuj z zadanym prawdopodobieństwem operację mutacji
      wobec wybranych osobników
      Dodaj osobniki do nowej populacji
```

W celu przyspieszenia obliczeń, funkcja przystosowania obliczania jest współbieżnie. Jest to możliwe ponieważ obliczanie funkcji przystosowania dla jednego osobnika jest operacją niezależną od obliczania jej dla pozostałych osobników. Dlatego też populacja dzielona jest na n części, gdzie n jest równa ilości rdzeni procesora, i funkcja przystosowania dla każdej części obliczana jest przez osobny wątek.

Kolejną różnicą pomiędzy powyższym algorytmem a standardowym algorytmem genetycznym opisanym wcześniej jest stosowanie modelu elitarnego. Elitarność polega na inicjowaniu nowo tworzonej populacji określoną ilością najlepiej przystosowanych osobników z poprzedniej populacji. Dzięki temu wartość przystosowania najlepszego osobnika z populacji nie będzie pogarszać się w kolejnych iteracjach algorytmu. Elitarność może przyczyniać się do przedwczesnej zbieżno-

ści algorytmu [13], dlatego aby temu zapobiec, wielkość elity będzie ograniczona do kilku osobników.

ROZDZIAŁ **3**

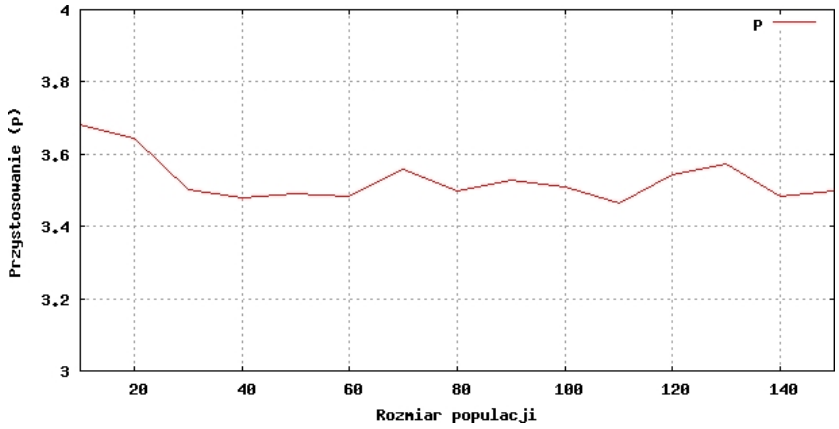
Wyniki i ich analiza

Algorytm został uruchomiony na serwerze z 8-rdzeniowym procesorem Intel Xeon 3.00 Ghz oraz 32 GB pamięci RAM. Ponieważ algorytm genetyczny zdecydowaną większość czasu spędza na ocenie rozwiązań, zastosowanie współbieżnego ich oceniania za pomocą 8 wątków spowodowało 5-krotne zmniejszenie czasu obliczeń.

Uruchomienie 100 iteracji algorytmu dla populacji o wielkości 100 osobników zajmuje 3,4 sekundy i pozwala na ocenę co najwyżej 10000 osobników. Ilość ta stanowi zaledwie $3.77 \cdot 10^{-27}$% całej przestrzeni poszukiwań. Ocena wszystkich możliwych rozwiązań metodą brute force zajęłaby około 10^{21} lat.

3.1. Dobór parametrów oraz operatorów

Algorytm został uruchomiony dla 400 iteracji. Rozmiar populacji został ustalony na 50, ponieważ dalsze jego zwiększanie nie ma wyraźnego wpływu na jakość rozwiązania, co przedstawiono na wykresie poniżej.

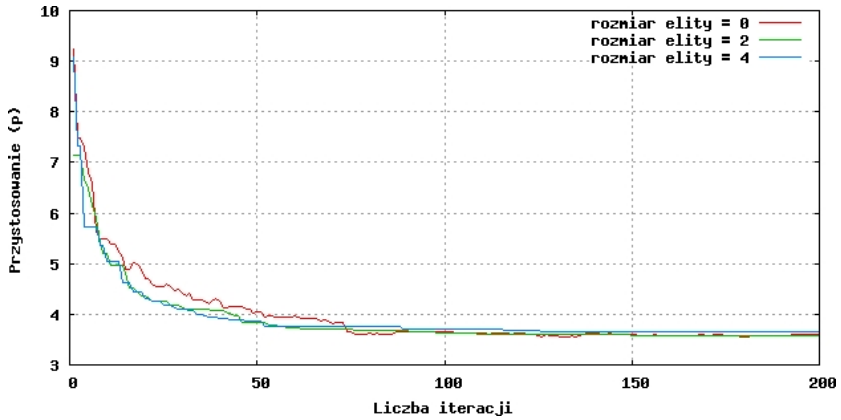


Rysunek **3.1**. Wpływ wielkości populacji na jakość rozwiązania

Wartości wag kryteriów zostały dobrane eksperymentalnie, tak aby wpływ poszczególnych kryteriów na końcowe rozwiązanie był jak najbardziej zrównoważony. Niektóre kryteria stanowią odchylenie od wartości optymalnej i ich wartości są stosunkowo niewielkie, w idealnym przypadku równe zeru. Inne natomiast to sumy częstotliwości, które przyjmują wartości o około 4 rzędy wielkości większe. Dlatego też wartości dla tych ostatnich zostały odpowiednio przeskalowane, poprzez dobranie dla nich wag o wartościach rzędu 10^{-4}. Przyjęte zostały następujące wartości wag:

$$w_{finger} = 37,2 \qquad w_{row} = 33,3$$
$$w_{hand\_alter} = 4,5 \cdot 10^{-4} \qquad w_{finger\_alter} = 1,6 \cdot 10^{-3}$$
$$w_{step} = 1,9 \cdot 10^{-4} \qquad w_{isf} = 2,5 \cdot 10^{-4}$$
$$w_{hand\_usage} = 71,4$$

Zastosowanie strategii elitarnej przyspiesza zbieżność algorytmu, co obrazuje poniższy wykres. Rozmiar elity nie ma dużego wpływu na jakość rozwiązania końcowego, dopóki jest on niewielki. Dla rozmiaru elity większego niż kilka osobników, jakość rozwiązania pogarsza się.



Rysunek **3.2**. Wpływ zastosowania strategii elitarnej

Celem zastosowania operatora selekcji MMTS jest zrównoważenie wpływu poszczególnych składowych funkcji oceny, aby móc osiągnąć lepszy wynik dla wszystkich składowych w stosunku do innych rozwiązań, takich jak klawiatura Dvoraka. Dla standardowej selekcji turniejowej, po 100 uruchomieniach algoryt-

mu przewaga w stosunku do klawiatury Dvoraka została osiągnięta średnio w 4,55 kryteriach na 7. Dla zmodyfikowanej selekcji MMTS, współczynnik ten wyniósł 4,76 kryteriów na 7. Dla standardowego algorytmu MMTS bez użycia go w połączeniu z metodą ważonych celów, wyniósł on 4,69.

Zbadany został wpływ operatorów mutacji oraz krzyżowania na jakość najlepszego rozwiązania (wartość funkcji przystosowania p) znalezionego w wyniku uruchomienia algorytmu. Wyniki dla operatorów mutacji: zamiany (EM), zamiany z przemieszczeniem (DM), mutacji ze wstawianiem (ISM), prostej inwersji (SIM), inwersji (IVM) oraz z mieszaniem podlist (SM) przedstawiono w poniższej tabeli. Wyniki zostały uśrednione dla 10 uruchomień algorytmu.

operator mutacji	najlepsza znaleziona wartość p
EM	3,553
DM	3,73
SIM	3,746
SM	3,876
IVM	3,882
ISM	4,301

Wyniki dla operatorów krzyżowania: cyklicznego (CX), z częściowym odwzorowaniem (PMX), bazującego na porządku (POS) oraz naprzemiennego (AP) przedstawiono poniżej, również uśrednione dla 10 uruchomień algorytmu.

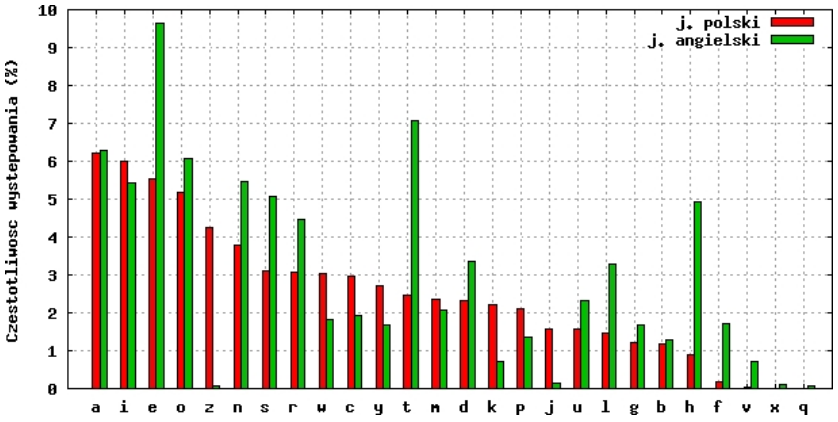
operator krzyżowania	najlepsza znaleziona wartość p
PMX	3,449
CX	3,509
POS	3,543
AP	3,582

Największy wpływ na wyniki ma stosowanie operatora mutacji, natomiast krzyżowanie ma stosunkowo niewielkie znaczenie. Stosowanie samego operatora mutacji bez krzyżowania prowadzi do otrzymywania rozwiązań około 1% gorszych w porównaniu do stosowania obu, natomiast samego krzyżowania bez mutacji, około 62% gorszych.

3.2. Dane testowe

Do obliczania oceny układów klawiatury dla języka polskiego został użyty zbiór tekstów powstały przez konkatenację 10 najpopularniejszych książek w języku polskim ze strony `www.gutenberg.org`. Analogicznie, do oceny układów klawiatury dla języka angielskiego wykorzystano 10 najpopularniejszych książek w języku angielskim. Zbiór tekstów w języku polskim ma wielkość 1,9MB (kodowanie UTF-8), a w języku angielskim 5,4MB.

Cechy statystyczne tekstów uwzględniane w funkcji przystosowania, czyli częstotliwości występowania monografów oraz diagrafów, różnią się znacząco dla różnych języków. Na poniższym wykresie porównano częstotliwości występowania monografów w języku polskim oraz angielskim.



Rysunek **3.3**. Porównanie częstotliwości występowania liter w języku polskim i angielskim

Pomimo pewnego podobieństwa, różnice są wyraźne, zwłaszcza w przypadku liter *e*, *z*, *t* oraz *h*. Jeszcze większe różnice widoczne są w przypadku diagrafów. Dla przykładu, najpopularniejsze pary znaków w języku angielskim, czyli *he* oraz *th*, w języku polskim nie występują prawie wcale. Analogicznie, często występujący w języku polskim dwuznak *rz* jest niespotykany w języku angielskim. Warto zwrócić uwgę, że para *th* jest bardzo łatwa do wpisania na klawiaturze Dvoraka. Porównanie częstotliwości występowania najczęstszych diagrafów w języku polskim i angielskim przedstawiono w tabeli poniżej.

diagraf	język polski	język angielski	diagraf	język angielski	język polski
ie	4,11%	0,30%	he	3,73%	0,11%
ni	2,78%	0,26%	th	3,68%	0,00%
na	1,64%	0,21%	in	2,39%	0,28%
rz	1,61%	0,00%	er	2,16%	0,68%
cz	1,58%	0,00%	an	2,10%	1,03%

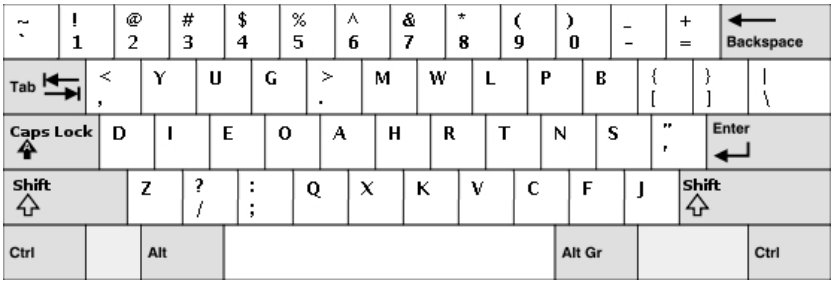
Ze względu na opisane wyżej różnice, wyniki algorytmu dla języka polskiego oraz angielskiego zostaną opisane osobno.

3.3. Wyniki dla języka angielskiego

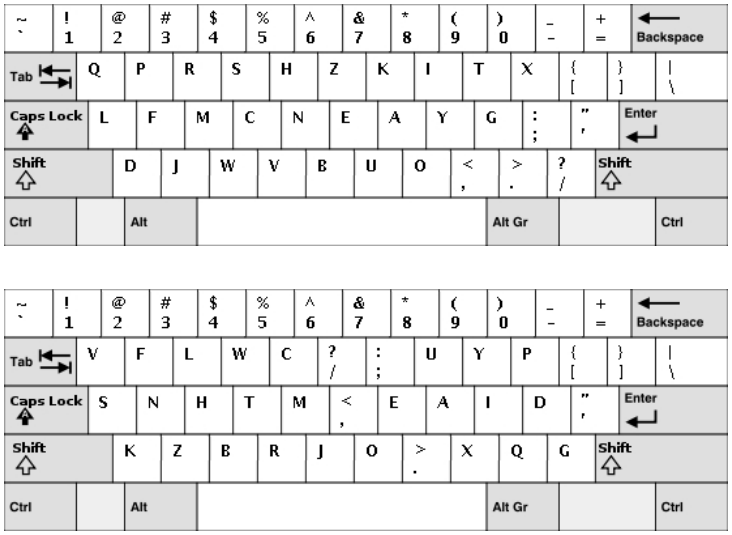
Porównanie wartości funkcji przystosowania oraz jej składników dla różnych układów klawiatury przedstawiono w tabeli poniżej. Układ znaleziony przez algorytm genetyczny opisany w tej pracy oznaczono jako AG. SW to najlepsze rozwiązanie znalezione przez algorytm symulowanego wyżarzania [22], a AG2 - przez inny algorytm genetyczny opisany w artykule [3].

kryterium	losowy	QWERTY	Dvorak	AG	SW	AG2
p	11,96	13,36	3,92	3,44	9,76	5,1
$p_{finger}w_{finger}$	1,12	0,38	0,31	0,12	1,4	0,49
$p_{row}w_{row}$	5,9	7,28	0,06	0,02	4,3	0,97
$p_{hand\_alter}w_{hand\_alter}$	1,1	1,11	0,81	0,79	0,78	0,81
$p_{finger\_alter}w_{finger\_alter}$	1,34	0,93	0,62	0,62	0,94	0,64
$p_{step}w_{step}$	1,41	1,65	0,91	0,78	1,15	0,93
$p_{isf}w_{isf}$	1,09	1,06	1,1	1,1	1,15	1,16
$p_{hand\_usage}w_{hand\_usage}$	0,0004	0,93	0,1	0,01	0,02	0,09

Ponieważ podczas projektowania układu QWERTY nie były uwzględniane kryteria ergonomiczne, należy spodziewać się że nie będzie on znacząco lepszy od losowo wygenerowanego układu klawiszy. Rzeczywiście, funkcja oceny nie różni się istotnie w tych przypadkach, a nawet jest gorsza dla QWERTY. Wyraźnie lepsze



Rysunek **3.4**. Najlepszy układ klawiszy dla języka angielskiego znaleziony przez algorytm

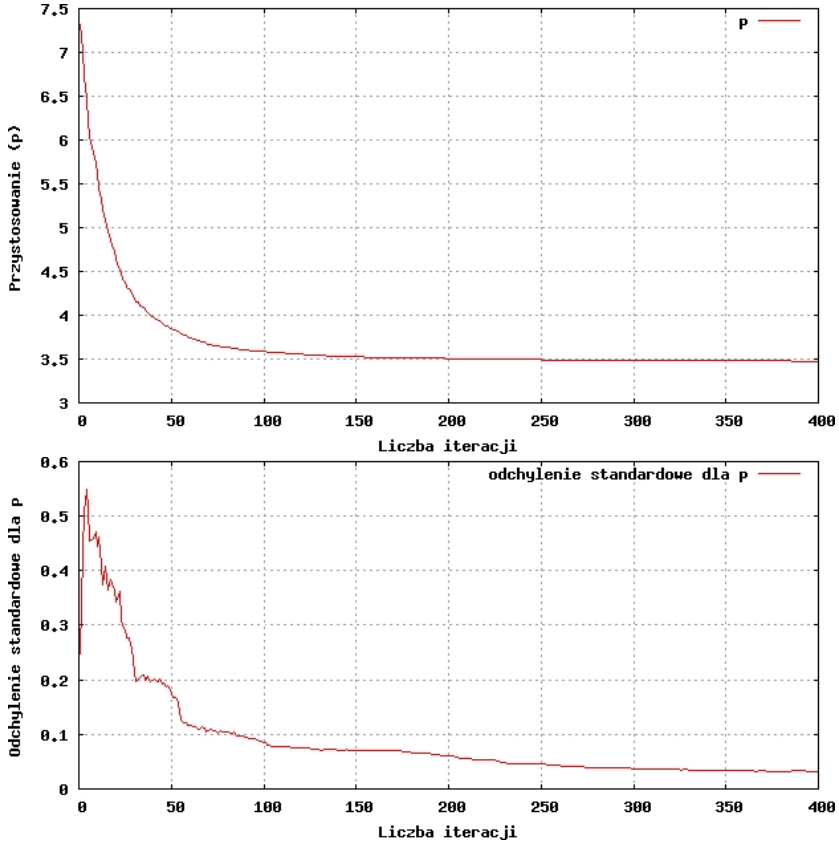


Rysunek **3.5**. Najlepsze rozwiązania algorytmów SW (na górze) oraz AG2 (na dole)

wyniki osiągnął układ Dvoraka, co nie jest zaskoczeniem biorąc pod uwagę fakt, że był on tworzony z myślą o języku angielskim, a kryteria stosowane przy jego projektowaniu w dużym stopniu pokrywają się z tymi użytymi w funkcji przystosowania. Najlepszy wynik znaleziony przez algorytm posiada ocenę o 12% korzystniejszą od najlepiej ocenionego układu wśród pozostałych, jakim jest klawiatura Dvoraka. Algorytmy AG2 oraz SW uzyskały ocenę lepszą od QWERTY, ale też wyraźnie gorszą od Dvoraka. Jednym z powodów takiego stanu może być zastosowanie prostszych funkcji oceny w tych algorytmach - nie został w nich uwzględniony na przykład czynnik *inboard stroke flow*. W przypadku algorytmu AG2 wpływ mógł mieć rów-

nież dobór zbioru tekstów; największą jego część stanowiła Biblia Króla Jakuba, napisana językiem różniącym się znacznie od współczesnego angielskiego.

Większość układów klawiatury znalezionych przez algorytm przejawia wyraźną tendencję do umieszczania samogłosek po tej samej, lewej lub prawej stronie i na środkowym rzędzie. Pomysł Augusta Dvoraka, aby umieścić wszystkie samogłoski z lewej strony zaprojektowanego przez niego układu, znajduje tym samym niezależne potwierdzenie empiryczne (należy zwrócić uwagę, że algorytm nie posiada żadnej informacji o tym, które znaki są samogłoskami, a które spółgłoskami).



Rysunek **3.6**. Ewolucja maksymalnego przystosowania w populacji oraz jego odchylenia standardowego. Wyniki uśredniono dla 30 wykonań algorytmu.

Różnice między poszczególnymi układami są szczególnie widoczne w przypadku wykorzystania rzędów klawiszy. Szczegółowe różnice przedstawiono w poniższej tabeli:

rząd klawiszy	losowy	QWERTY	Dvorak	AG	SW	AG2
górny	40,5%	49,1%	22,9%	22,0%	35,9%	18,6%
środkowy	36,5%	33,5%	68,3%	70,9%	41,7%	62,1%
dolny	23%	17,4%	8,8%	7,1%	22,4%	19,3%

W przypadku układu QWERTY zdecydowanie zbyt duży nacisk położony jest na górny rząd klawiszy, podczas gdy najkorzystniejszy środkowy rząd jest wykorzystany w zaledwie jednej trzeciej. Jest to jedna z największych wad tego układu; nietrudno jest wygenerować losowo układ z lepszym rozkładem. Dobrze widoczne są zalety klawiatury Dvoraka, która posiada rozkład zbliżony do optymalnego, wykorzystując w bardzo dużym stopniu rząd środkowy. Mimo to, układ klawiatury znaleziony przez algorytm poprawił ten wynik jeszcze bardziej.

Poniższa tabela przedstawia różnice w stopniu użycia poszczególnych palców pomiędzy testowanymi układami klawiszy:

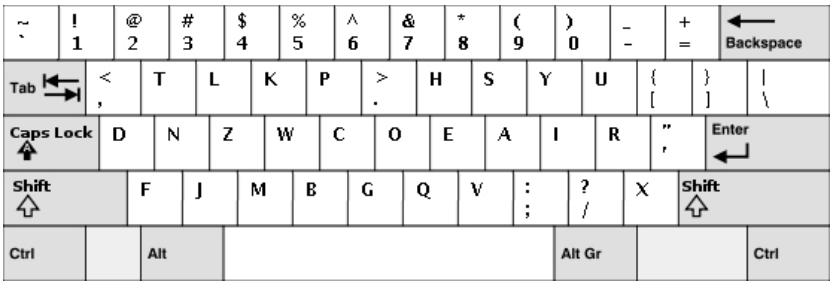
układ	lewy mały	lewy serd.	lewy śr.	lewy wsk.	prawy wsk.	prawy śr.	prawy serd.	prawy mały
losowy	14%	2%	10%	21%	16%	24%	9%	4%
QWERTY	8%	9%	18%	21%	20%	9%	13%	2%
Dvorak	8%	9%	13%	14%	18%	13%	13%	10%
AG	6%	9%	15%	19%	18%	15%	10%	8%
SW	8%	4%	10%	24%	31%	10%	12%	0%
AG2	8%	9%	12%	22%	23%	11%	9%	8%

Tym razem układ QWERTY sprawdza się lepiej od losowo wygenerowanego. Rozkłady dla klawiatury Dvoraka oraz układu znalezionego przez algorytm są jednak bardziej równomierne.

3.4. Wyniki dla języka polskiego

Porównanie wartości funkcji przystosowania oraz jej składników dla różnych układów klawiatury:

kryterium	losowy	QWERTY	Dvorak	AG
p	7,25	8,44	3,64	1,06
$p_{finger}w_{finger}$	0,72	0,77	0,94	0,05
$p_{row}w_{row}$	4,66	5,64	1,07	0,03
$p_{hand\_alter}w_{hand\_alter}$	0,35	0,37	0,28	0,23
$p_{finger\_alter}w_{finger\_alter}$	0,35	0,31	0,22	0,14
$p_{step}w_{step}$	0,52	0,62	0,46	0,23
$p_{isf}w_{isf}$	0,36	0,36	0,38	0,37
$p_{hand\_usage}w_{hand\_usage}$	0,30	0,37	0,28	0,002



Rysunek **3.7**. Najlepszy układ klawiszy dla języka polskiego znaleziony przez algorytm

Układ klawiatury Dvoraka był zaprojektowany z myślą o języku angielskim, dlatego należy się spodziewać że wynik dla języka polskiego nie będzie optymalny. Rzeczywiście, układ Dvoraka jest nadal lepszy od losowo wygenerowanego lub QWERTY, ale bardzo daleki od optimum. Układ klawiszy wygenerowany przez algorytm jest natomiast wyraźnie lepszy od wszystkich pozostałych, co daje znaczącą przewagę w porównaniu z wynikami dla języka angielskiego.

Różnice między poszczególnymi układami ze względu na wykorzystanie rzędów klawiszy przedstawiono w poniższej tabeli:

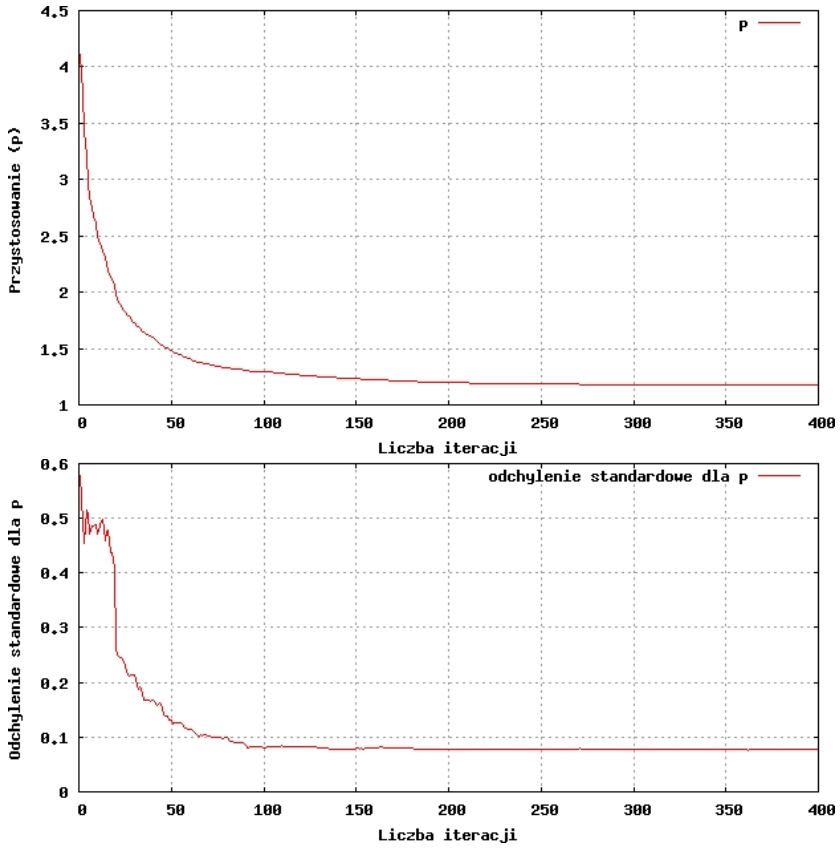
rząd klawiszy	losowy	QWERTY	Dvorak	AG
górny	33,8%	44,7%	25,0%	29,0%
środkowy	35,5%	30,3%	54,0%	61,9%
dolny	30,7%	25,0%	21,0%	9,1%

W przypadku układu QWERTY, sytuacja wygląda jeszcze gorzej niż dla języka angielskiego: udział bardzo niekorzystnego rzędu dolnego wzrósł, a rzędu środkowego spadł do mniej niż jednej trzeciej. Wynik klawiatury Dvoraka również się pogorszył, co nie jest zaskakujące, ale nadal jest lepszy od losowo wygenerowanego rozwiązania. Przewaga rozwiązania znalezionego przez algorytm, niewielka w przypadku języka angielskiego, tym razem jest bardziej wyraźna.

Poniższa tabela przedstawia różnice w stopniu użycia poszczególnych palców pomiędzy testowanymi układami klawiszy:

układ	lewy mały	lewy serd.	lewy śr.	lewy wsk.	prawy wsk.	prawy śr.	prawy serd.	prawy mały
losowy	10%	6%	8%	20%	19%	13%	17%	9%
QWERTY	16%	9%	16%	11%	17%	13%	14%	3%
Dvorak	10%	10%	12%	20%	11%	12%	9%	16%
AG	5%	11%	14%	18%	19%	15%	12%	6%

Ze względu na różnice w częstotliwości występowania liter, wyniki dla układów QWERTY oraz Dvorak są znacznie gorsze niż w przypadku języka angielskiego. QWERTY kładzie zbyt duży nacisk na lewy mały palec, który służy do wpisywania popularnych w języku polskim liter *a* oraz *z*. Z kolei klawiatura Dvoraka przeciąża prawy mały palec, obsługujący litery *l*, *s* oraz *z*, w zbyt małym stopniu natomiast wykorzystując prawy palec wskazujący. Problemy te nie dotyczą układu wygenerowanego przez algorytm.



Rysunek **3.8**. Ewolucja maksymalnego przystosowania w populacji oraz jego odchylenia standardowego. Wyniki uśredniono dla 30 wykonań algorytmu.

Zakończenie

3.5. Konkluzje

Projektowanie optymalnego układu klawiatury (KAP - Keyboard Arrangement Problem) jest złożonym problemem, który wymaga uwzględnienia wielu czynników wpływających na ergonomię pisania. Rozwiązywanie go bez pomocy komputera jest zajęciem bardzo czasochłonnym i nie gwarantuje znalezienia optymalnego rozwiązania. Co więcej, dla każdego języka naturalnego problem musi być podejmowany od nowa, ze względu na różnice w częstotliwości występowania liter oraz ich par pomiędzy językami. Jak pokazują wyniki, dobór tekstów do których pisania ma być przystosowany dany układ klawiszy ma bardzo duży wpływ na jego ocenę.

Ogromna ilość potencjalnych rozwiązań oraz trudność w optymalizowaniu wielu czynników jednocześnie sprawiają, że KAP jest dobrym kandydatem na zastosowanie algorytmu genetycznego. Naturalnym sposobem reprezentacji chromosomu jest permutacja, co pozwala na stosowanie operatorów genetycznych wypróbowanych na innych zagadnieniach optymalizacji kombinatorycznej, takich jak problem komiwojażera.

Uzyskane wyniki pokazują jasno, że istnieją układy klawiszy lepsze od tych pozostających w powszechnym użyciu i algorytmy genetyczne potrafią znajdywać je stosunkowo szybko. Najlepszym spośród powstałych dotychczas układów okazał się układ Dvoraka. Dla języka angielskiego, najlepszy układ znaleziony przez algorytm uzyskał korzystniejszą ocenę w stosunku do klawiatury Dvoraka pod względem wszystkich czynników branych pod uwagę w funkcji oceny. W przypadku wyników dla języka polskiego, różnice są jeszcze bardziej widoczne. Pewnym zaskoczeniem są słabe wyniki osiągnięte przez inne algorytmy (SW, AG2), szczególnie w porównaniu do klawiatury Dvoraka. Może to wynikać z zastosowania prostszych kryteriów oceny rozwiązań oraz mało różnorodnych zbiorów tekstów przez ich autorów.

Istnieje niewiele badań empirycznych na temat czynników wpływających na ergonomię pisania, a te istniejące najczęściej przeprowadzane były na niewielką

skalę. Dlatego też kryteria przyjęte podczas oceny rozwiązań przedstawionego algorytmu z pewnością nie stanowią ostaniego słowa w tej dziedzinie. Stanowią one jednak pewien punkt wyjścia i można je w łatwy sposób zmodyfikować w miarę pojawiania się kolejnych danych w dziedzinie ergonomii.

3.6. Plany dalszych badań

Użyty algorytm był stworzony z myślą o technice pisania bezwzrokowego. Interesującym problemem byłoby przystosowanie go do techniki pisania jednym palcem, która znajduje zastosowanie między innymi podczas pisania na urządzeniach posiadających ekran dotykowy. W tym celu można by zmodyfikować funkcję oceny w taki sposób, żeby algorytm minimalizował długość ścieżki jaką musi przebyć palec (lub rysik albo podobne urządzenie) podczas przepisywania zadanego tekstu. Tego typu zadanie można sprowadzić do zagadnienia Quadratic Assignment Problem (QAP), będącego jednym z klasycznych problemów optymalizacji kombinatorycznej [23]. Algorytmy genetyczne były już wykorzystywane z pewnymi sukcesami do rozwiązywania QAP [24].

Podczas projektowania funkcji oceny ograniczono się do kryteriów ergonomicznych mających na celu ułatwienie pisania zwykłego tekstu. Możliwe byłoby wprowadzenie dodatkowych kryteriów mających znaczenie dla klawiatury komputerowej. Na przykład, układ klawiatury Colemak został stworzony z myślą o łatwym dostępie do standardowych skrótów klawiszowych stosowanych w oprogramowaniu, takich jak Ctrl-C, Ctrl-X i Ctrl-V.

Warto byłoby również zbadać wpływ rodzaju zbioru tekstów na wynikowe układy klawiatury. W tym przypadku wykorzystano teksty literackie w dwóch językach. Innym przykładem byłoby zastosowanie kodu źródłowego w różnych językach programowania, w celu stworzenia układu klawiatury przystosowanego specjalnie dla programisty.

Algorytm genetyczny jaki został użyty z pewnością mógłby zostać rozwinięty i ulepszony. Korzystne byłoby przetestowanie większej ilości operatorów genetycznych oraz zaawansowanych technik, takich jak niszowość lub zmienność rozmiaru populacji w czasie. W szczególności, warto byłoby porównać otrzymane wyniki z

wynikami uzyskanymi przez zastosowanie specjalistycznych wariantów algorytmów genetycznych zaprojektowanych z myślą o optymalizacji wielokryterialnej, takich jak NSGA, NSGA-II lub VEGA.

DODATEK **A**

Część programowa

Do pracy dołączono program w języku Java, stanowiący implementację opisanego w niej algorytmu.

Sposób uruchomienia programu:
`java -cp build/classes mdettla.keyboard.ga.GAKeyboard [parametry] PLIK...`

Jako argument `PLIK...` należy podać ścieżki do plików tekstowych, na podstawie których zostaną obliczone statystyki używane w funkcji oceny. Teksty dla języka angielskiego znajdują się w katalogu *src/mdettla/keyboard/ga/resources/en*, a dla języka polskiego w katalogu *src/mdettla/keyboard/ga/resources/pl*.

Dostępne są następujące parametry pozwalające wpływać na działanie algorytmu:

- `populationSize` - rozmiar populacji
- `generationsCount` - ilość iteracji
- `eliteSize` - rozmiar elity
- `tournamentSize` - rozmiar turnieju
- `mutationProbability` - prawdopodobieństwo zajścia mutacji
- `mutationOperator` - operator mutacji (należy podać nazwę jednej z klas znajdujących się w katalogu *src/mdettla/jga/operators/mutation*)
- `crossoverProbability` - prawdopodobieństwo zajścia krzyżowania
- `crossoverOperator` - operator krzyżowania (należy podać nazwę jednej z klas znajdujących się w katalogu *src/mdettla/jga/operators/crossover*)

- `selectionFunction` - funkcja selekcji (należy podać nazwę jednej z klas znajdujących się w katalogu *src/mdettla/jga/operators/selection*)

Przykład uruchomienia programu:

```
java -cp build/classes mdettla.keyboard.ga.GAKeyboard \
   populationSize=100 mutationProbability=0.5 \
   mutationOperator=mdettla.jga.operators.mutation.SwapMutation \
   src/mdettla/keyboard/ga/resources/pl/*
```

Bibliografia

[1] D. A. Norman. *The Design of Everyday Things*. Basic Books, 1988.

[2] R. C. Cassingham. *The Dvorak Keyboard: The Ergonomically Designed Typewriter Keyboard, Now an American Standard*. Freelance Communications, 1986.

[3] B. A. Julstrom J. S. Goettl, A. W. Brugh. Call me e-mail: Arranging the keyboard with a permutation-coded genetic algorithm. *Proceedings of the 2005 SCM Symposium on Applied Computing*, 2:947–951, 2005.

[4] J. Eggers, D. Feillet, S. Kehl, M. O. Wagner, and B. Yannou. Optimization of the keyboard arrangement problem using an ant colony algorithm. *European Journal of Operational Research*, 148(3):672 – 686, 2003.

[5] P. Yin and E. Su. Cyber swarm optimization for general keyboard arrangement problem. *International Journal of Industrial Ergonomics*, 41(1):43 – 52, 2011.

[6] P. Deshwal and K. Deb. Design of an optimal hindi keyboard for convenient and efficient use. *Technical Report KanGAL Report No. 2003005, Indian Institute of Technology, Kanpur*, 2003.

[7] Y. Li, L. Chen, and R. S. Goonetilleke. A heuristic-based approach to optimize keyboard design for single-finger keying applications. *International Journal of Industrial Ergonomics*, 36(8):695 – 704, 2006.

[8] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictwa Naukowo-Techniczne, third edition, 2003.

[9] R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317:804 – 806, 1985.

[10] T. Stützle, A. Grün, S. Linke, and M. Rüttger. A comparison of nature inspired heuristics on the traveling salesman problem. In *Proceedings of the 6th*

International Conference on Parallel Problem Solving from Nature, PPSN VI, pages 661–670, London, UK, 2000. Springer-Verlag.

[11] P. Larrañaga, C. M. H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.

[12] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[13] S. Luke. *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[14] D.E. Goldberg. *Algorytmy genetyczne i ich zastosowania*. Wydawnictwa Naukowo-Techniczne, second edition, 1998.

[15] D. E. Glover. Solving a complex keyboard configuration problem through generalized adaptive search. In *Genetic Algorithms and Simulated Annealing*, pages 12–27. 1987.

[16] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the tsp. In J. J. Grefenstette, editor, *Genetic algorithms and their applications: Proceedings of the Second International Conference*, pages 224 – 230, Hillsdale, New Jersey, 1987.

[17] D. E. Goldberg and R. Lingle. Alleles, loci and the tsp. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 154 – 159, Hillsdale, New Jersey, 1985.

[18] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 332 – 349. Van Nostrand Reinhold, New York, 1991.

[19] D. B. Fogel. Applying evolutionary programming to selected travelling salesman problems. *Cybernetics and Systems*, 24:27 – 36, 1993.

[20] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221 – 248, 1994.

[21] J. D. Shaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, 1984.

[22] L. W. Light and P. G. Anderson. Typewriter keyboards via simulated annealing. *AI Expert*, September 1993.

[23] Y. Wu P. Ji and H. Liu. A solution method for the quadratic assignment problem (qap). *Lecture Notes in Operations Research*, 6:106 – 117, 2006.

[24] A. Misevicius. An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowledge-Based Systems*, 17(2-4):65 – 73, 2004.

Spis rysunków

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis