Informatyka techniczna i sztuczna inteligencja
Technical informatics and Artificial Intelligence
Engineer of XXI Century'2024

Worktime scheduling with genetic algorithms

Tomasz Steblik[1,*], Mirosław Kordos [2]

[1] *University of Bielsko-Biala, Poland, ts055953@student.ubb.edu.pl*
[2] *University of Bielsko-Biala, Poland, mkordos@ubb.edu.pl*
**corresponding author*

Abstract: *The paper presents an automatic work scheduling system. The presented solution is based mostly on genetic algorithms, along with an analysis of various possible solutions. We discuss possible methods of encoding the work schedule in the chromosome, possible crossover and mutation operators, and possible forms of fitness functions. The possibility of combining genetic algorithms with local search methods and constraint programming is also analyzed.*

Keywords: worktime scheduling; genetic algorithms;

Układanie harmonogramów pracy z wykorzystaniem algorytmów genetycznych

Tomasz Steblik[1,*], Mirosław Kordos [2]

[1] *Uniwersytet Bielsko-Bialski, Polska, ts055953@student.ubb.edu.pl*
[2] *Uniwersytet Bielsko-Bialski, Polska, mkordos@ubb.edu.pl*
* *Corresponding author*

Streszczenie: *Artykuł przedstawia systemem automatycznego układania harmonogramów pracy. Prezentowane rozwiązanie jest oparte w większości o algorytmy genetyczne, wraz z analizą różnych rozwiązań. Omówiono różne metody kodowania harmonogramu pracy w chromosomie, możliwe do zastosowania operatory krzyżowania i mutacji, różne postacie funkcji dopasowania. Przeanalizowano także możliwość połączenia algorytmów genetycznych z metodami przeszukiwania lokalnego i programowaniem z ograniczeniami.*

Słowa kluczowe: harmonogramy pracy; algorytmy genetyczne

1. Wstęp

Optymalizacja harmonogramu pracy pracowników w zakładzie jest powszechnym problemem, z którym mierzą się wszystkie firmy. Odpowiednie dopasowanie ilości osób o konkretnych kwalifikacjach na każdą godzinę pracy zakładu przekłada się na znaczącą poprawę wydajności. Również ważnym elementem w takim scenariuszu jest odpowiednie dopasowanie dni roboczych każdego pracownika zgodnie z jego życzeniami. Elastyczny grafik jest jednym z ważniejszych czynników który wpływa na zadowolenie pracowników. Kolejnym problemem jest sytuacja, w której pracownik mający pracować na maszynie np. rozchoruje się tego samego dnia albo maszyna ulegnie awarii. Takie przypadki powodują znaczący spadek wydajności zakładu. W takiej sytuacji kluczowym jest ponowne utworzenie harmonogramu tak aby zredukować czas postoju maszyny lub braku pracownika do minimum. Tworzenie harmonogramu od podstaw jest bardzo trudnym i złożonym zadaniem, zwłaszcza kiedy musimy mieć na uwadze życzenia pracowników.

Ręczne układanie harmonogramów pracy jest czasochłonne, kosztowne, często nieoptymalne oraz podatne na błędy ludzkie. Prowadzi to do niezadowolenia pracowników, niskiej efektywności tworzonych grafików oraz kar

DOI: https://doi.org/10.53052/9788367652292.22

finansowych wynikających z błędów ludzkich związanych z przepisami Kodeksu Pracy. Automatyzacja tego procesu powinna rozwiązać te problemy, a dodatkowo zwiększyć zadowolenie załogi z ułożonych harmonogramów poprzez spełnienie większej liczby życzeń pracowników, zapewnić bardziej optymalne obsadzenie stanowisk i wyeliminować pojawiające się czasami wśród załogi poczucie niesprawiedliwości, że innym pracownikom ułożono lepszy harmonogram.

Ponieważ ułożenie optymalnego harmonogramu pracy (czyli takiego który spełnia wszystkie koniecznie wymagania i maksymalnie dużo opcjonalnych życzeń pracowników i kierowników) jest zagadnieniem NP-zupełnym, gdzie liczba możliwych harmonogramów nawet w średniej wielkości firmy jest tak duża, że nie jest możliwe sprawdzenie przez program komputerowy wszystkich rozwiązań i wybranie najlepszego z nich, a tym bardziej nie jest więc możliwe ułożenie takiego grafiku przez człowieka.

Pełna automatyzacja, oparta na metodach sztucznej inteligencji może zapewnić rozwiązanie tego problemu. Jednakże nie jest to zagadnieniem prostym ze względu na niezwykle skomplikowane zależności występujące w harmonogramach pracy. W niniejszej pracy skupiamy się na dostosowaniu algorytmów genetycznych [1] do układania harmonogramów pracy [2,3]. Dwa główne problemy polegają na tym, że harmonogramy pracy są zagadnieniem trójwymiarowym (te wymiary to: pracownik, stanowisko, czas), natomiast istniejące metody optymalizacyjne przeznaczone są do zagadnień dwuwymiarowych oraz na występowaniu licznych ograniczeń i interakcji w takim systemie. Dodatkowo analizujemy też wykorzystanie przeszukiwania lokalnego zaimplementowanego w operatorze krzyżowania w algorytmie genetycznym oraz programowania z ograniczeniami.

W ramach przeprowadzonych prac badawczych dokonaliśmy:

- analizy różnych metod reprezentacji problemu w chromosomie wraz z wynikłymi z tego możliwościami i ograniczeniami dobru i adaptacji operatorów krzyżowania, mutacji, operatorów rozwiązywania konfliktów przy krzyżowaniu, liczby rodziców w operatorach krzyżowania, częstości występowania przeszukiwania lokalnego,
- dobór funkcji dopasowania
- zaprojektowania operatorów przeszukiwania lokalnego,
- porównania z próbami rozwiązania problemu przy pomocy programowania z ograniczeniami (ang. constraint programming).
- analizy dodatkowych zagadnień występujących w tym problemie.

Doświadczenia z naszych poprzednich prac nad algorytmami optymalizacji w produkcji i handlu [4,5], a także opracowania dostępne w Internecie wskazują, że w tego typu problemach kombinatorycznych rozwiązania znalezione przez algorytmy z wykorzystaniem metod sztucznej inteligencji cechują się przeciętnie o 10-15% lepszym wynikiem (skrócenie, czasu, drogi, innej wielkości optymalizowanej) w porównaniu do rozwiązań ułożonych ręcznie przez człowieka. W przypadku układania harmonogramów pracy możliwa do uzyskania poprawa może być podana przez różne funkcje kryterialne i może się różnić w zależności od przyjętej funkcji kryterialnej, a jej podanie procentowe wymaga przeprowadzenia analizy porównawczej na wielu rzeczywistych przypadkach układania harmonogramu w różnych firmach. Aktualnie jesteśmy na etapie gromadzenia i analizowania tych przypadków. F. M. Howard w swoim opracowaniu [6] podaje, że po zastosowanie automatyzacji układania harmonogramów w jednym ze szpitali ponad 2-krotnie uległo zmniejszeniu występowanie konfliktów wśród załogi oraz zwiększyło się zadowolenie pracowników o średnio 0,8 pkt na 5- stopniowej skali Likerta w porównaniu z ręcznie układanymi harmonogramami.

Możliwe jest również zastosowanie elementów programowania z ograniczeniami (constraint programming) i to przeanalizowaliśmy w niniejszej pracy korzystając z opracowanego przez Google solwera OR Tools [18]. W ostatnich czasach również pojawiły się w literaturze próby wykorzystania różnych struktur sieci neuronowych jako pomocniczego mechanizmu w układaniu harmonogramów pracy [8-10]. Na aktualnym etapie mamy jednak zgromadzone zbyt mało danych, by próbować z takich rozwiązań korzystać.

2. Metody i algorytmy

W tej sekcji przedstawiamy analizowane przez nas rozwiązania oparte na algorytmach genetycznych i ich połączeniu z przeszukiwaniem lokalnym. W końcowej części rozdziału (sekcja 2.8) omówimy także wykorzystanie programowania z ograniczeniami.

W algorytmie genetycznym [1] najpierw kodujemy rozwiązanie problemu w chromosomie (sekcja 2.1). Następnie losowo tworzymy populację N osobników (np. N=100). Od tej pory iteracyjnie prowadzimy następujące operacje:

- wyznaczenie funkcji dopasowania wszystkich osobników (sekcja 2.3),
- selekcji osobników na rodziców (sekcja 2.6),
- krzyżowania (sekcja 2.4.) wraz z opcjonalnie dodanym przeszukiwaniem lokalnym (sekcja 2.8),
- eliminacji (sekcja 2.7).

Proces ten trwa tak długo, aż uznamy, że zostało znalezione odpowiednie rozwiązanie, lub upłynęła założona liczba iteracji i wtedy wybieramy najlepsze z istniejących rozwiązań.

Algorytmy genetyczny są metodą doskonale nadającą się do poszukiwania optymalnego rozwiązania w bardzo dużych i złożonych zagadnieniach. Przykładowo, jeśli zagadnienie jest zakodowane na 100 pozycjach binarnych (zera i jedynki) to liczba możliwych rozwiązań wynosi $2^{100} \approx 10^{30}$ i tyle rozwiązań trzeba by sprawdzić, żeby znaleźć najlepsze z nich (tzw. metoda brute force). Natomiast algorytmy genetyczne dzięki inteligentnemu procesowi przeszukiwania są w stanie znaleźć najlepsze rozwiązanie przeszukując zaledwie kilka tysięcy różnych rozwiązań, czyli często znaleźć rozwiązanie w czasie rzędu sekund lub minut (w zależności od złożoności funkcji celu), podczas gdy metodą brute force zajęło by to wiele trylionów lat, czyli w praktyce byłoby niewykonalne.

Zaletą algorytmów genetycznych nad metodami przeszukiwania lokalnego (które w nielicznych wypadkach można sprowadzić do metod gradientowych) jest to, że w praktycznych złożonych zagadnieniach funkcja celu ma wiele (kilkadziesiąt, kilkaset) argumentów i jest niezwykle złożona, pełna minimów lokalnych. W tych minimach lokalnych utykają metody przeszukiwania lokalnego, nie będąc w stanie zbliżyć się do żadnego minimum globalnego (bo w złożonych problemach tych równoważnych minimów globalnych jest też często więcej niż jedno). Natomiast algorytmy genetyczne całkiem dobrze sobie radzą ze znalezieniem jednego z minimów globalnych lub zbliżeniem się do niego na tyle blisko, że w praktycznych zastosowaniach nie robi to żadnej różnicy. Jednak jeśli już proces optymalizacji się zbliży do minimum globalnego, to można czasem w końcowej fazie wykorzystać również metody przeszukiwań lokalnych celem przyśpieszenia znalezienia końcowego rozwiązania.

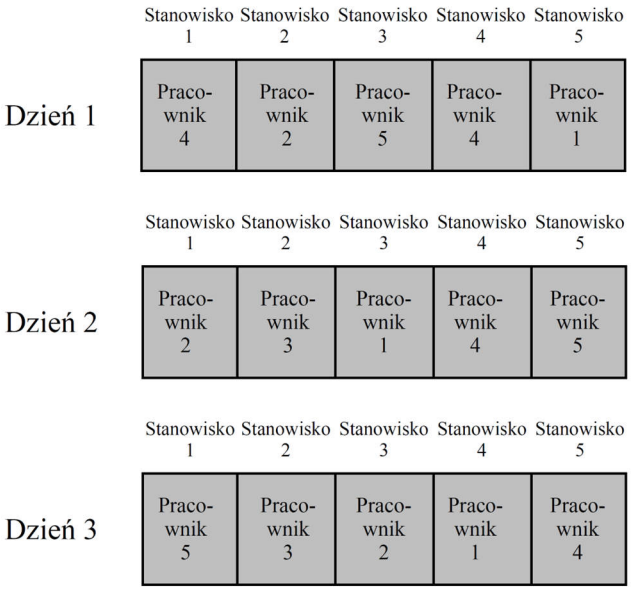
Układanie harmonogramów pracy (niezależnie jaką metodą) powinno być prowadzone w najmniejszych niezależnych od siebie jednostkach firmy, ponieważ łatwiej wtedy znaleźć optymalne rozwiązanie, pod warunkiem, że pracownicy pracujący w różnych działach firmy nie mają możliwości zastępowania siebie nawzajem na danych stanowiskach.

2.1. Reprezentacja harmonogramu pracy w chromosomie

Biorąc pod uwagę, że każde stanowisko pracy musi mieć przypisanego jakiegoś pracownika do obsługi, chromosom dal jednej zmiany możnaby zdefiniować jako tablicę jednowymiarową której indeks będzie odpowiadał stanowisku pracy a wartość będzie zawierać przypisanego pracownika. Natomiast, gdy celem algorytmu jest ułożenie optymalnego harmonogramu pracy obejmującego wiele zmian i wiele dni (a w praktyce tak jest zawsze) to problem można zakodować w chromosomie na kilka innych sposobów. Jednym z nich jest użycie tablicy dwuwymiarowej, gdzie dodatkowym wymiarem będzie dzień roboczy, co przedstawia rys. 1. Przy pracy w systemie wielozmianowym w powyższej reprezentacji dzień zostaje zastąpiony zmianą. W praktyce często jest tak, że dany pracownik może pracować tylko na jednym stanowisku (choć może na nim pracować jednocześnie kilku pracowników, np. w sklepie, restauracji, itp.), co zmienia nieco sposób kodowania.

Oprócz schematu przedstawionego na rys. 1 możliwe są też inne sposoby zakodowania problemu. Przykładowo, jeśli układamy tygodniowy (7dni) harmonogram pracy i mamy 20 pracowników, 3 zmiany w każdym dniu (0-dzień wolny, 1-pierwsza zmiana, 2-druga, 3-trzecia) to można to zakodować w chromosomie o długości 7*20=140, gdzie na każdej pozycji będzie zmiana danego pracownika w danym dniu (0 do 4) [2,11,12]. Takie kodowanie daje nam jednowymiarowy chromosom, jednak dalej zostają zależności między jego pozycjami (np. danego dnia na danej zmianie wymaganych jest 5 pracowników). Skutkiem tego złożoność problemu nie maleje, lecz w inny sposób jest sprawdzana zgodność harmonogramu.

DOI: https://doi.org/10.53052/9788367652292.22



Rysunek 1. Przykładowa reprezentacja chromosomu dla 3 dni z 5 pracownikami i 5 stanowiskami.

2.2. Ograniczenia

W zagadnieniu układania harmonogramów pracy występuje cały szereg wymagań, które będziemy również nazywać ograniczeniami. Można je podzielić obowiązkowe (twarde) i opcjonalne (miękkie). Ograniczenia obowiązkowe muszą być spełnione, aby harmonogram był poprawny i akceptowalny. Ograniczenia opcjonalne nie muszą, ale czym więcej ich jest spełnionych, tym harmonogram należy uznać za lepszy. Więc dążymy do spełnienia tak wielu ograniczeń opcjonalnych, jak to możliwe.

Wymagania obowiązkowe to warunki które muszą zostać spełnione, aby harmonogram mógł zostać uznany za prawidłowy:

- Pracownik może znajdować się tylko na takim stanowisku, które potrafi obsługiwać,
- Pracownik musi pracować w sposób zgodny z kodeksem pracy, np. nie może mieć dwóch zmian pod rząd,
- Wymagane do obsadzenia stanowiska muszą zostać obsadzone wymaganą liczbą pracowników.

W tej pracy przyjęliśmy następujące wymagania opcjonalne:

- Czy pracownik znajduje się na preferowanej przez niego maszynie,
- Czy pracownik pracuje w preferowane przez niego dni,
- Czy pracownik pracuje preferowaną ilość dni.

Mając na uwadze zdefiniowane wymagania twarde oraz miękkie zaproponowaliśmy kilka operatorów krzyżowania omówionych w sekcji 2.5

2.3. Funkcja dopasowania

Funkcja dopasowania określa wartość każdego osobnika (czyli każdego rozwiązania stanowiącego potencjalny harmonogram) w populacji. Dla problemu komiwojażera będzie to długość drogi, którą musi przebyć bo odwiedzić wszystkie miasta z listy. Natomiast dla niektórych problemów funkcja dopasowania może być trudna lub wręcz nie możliwa do zdefiniowania za pomocą prostego wzoru. Dotyczy to najczęściej przypadków, gdy należy przeprowadzić pewne symulacje, których wynik stanowi wartość funkcji dopasowania. Funkcję dopasowania można maksymalizować lub minimalizować w procesie optymalizacji.

Przeanalizowano trzy podejścia do skonstruowania funkcji dopasowania do układania harmonogramów pracy:

1. Funkcja dopasowania zawiera tylko miękkie wymagania, czyli życzenia pracowników oraz inne ewentualne parametry, które lepiej jak są spełnione, ale spełnienie ich nie jest konieczne. Twarde wymagania, które muszą być spełnione, np. liczba pracowników na danej zmianie (czasem może ty być konkretna liczba, czasem zakres od do) lub założenia wynikłe z kodeksu pracy odnośnie tego kiedy ile i w jakich odstępach pracownik może pracować natomiast są kontrolowane najpierw podczas inicjalizacji populacji, a następnie w operatorze krzyżowania i operator ten nie dopuszcza do powstania osobników, które tych parametrów nie spełniają.

2. Funkcja dopasowania zawiera zarówno twarde jak i miękkie wymagania Przykładem takiej funkcji dopasowania jest funkcja w postaci: suma ilości pracowników na preferowanych przez nich stanowiskach (oczywiście w obrębie stanowisk, na których mogą pracować), ilości pracowników pracujących w wybrane przez nich dni oraz różnicy wybranej przez pracownika ilości dni i faktycznej ilości dni pracujących. Tak zdefiniowana funkcja dopasowania podlega maksymalizacji w procesie optymalizacyjnym. Funkcja dopasowania zawiera zarówno twarde (z większą wagą), jak i miękkie (z mniejszą wagą) wymagania. W tym przypadku brak jest kontroli w operatorze krzyżowania i dopuszcza się tymczasowe powstanie osobników, który mogą nie spełniać wymaganych założeń (wymagana liczba pracowników, kodeks pracy), a zadaniem funkcji dopasowania jest ich stopniowe eliminowanie podczas optymalizacji.

3. Połączenie obu powyższych podejść poprzez stosowanie w początkowej fazie optymalizacji podejścia nr 2 celem zapewnienia odpowiedniej różnorodności populacji, a następnie przełączenie się na podejście nr 1, aby już nie tworzyć niepoprawnych osobników w końcowej fazie. Kwestia zapewnienia różnorodności populacji jest omawiana w dalszej części artykułu.

2.4. Operatory krzyżowania

Celem operatora krzyżowania jest łączenie informacji z dwóch lub więcej różnych chromosomów (rodziców) w jeden chromosom (dziecko), który może reprezentować lepsze rozwiązanie niż jego rodzice.

Sposób zakodowania problemu w chromosomie determinuje rodzaj operatorów krzyżowania, który możemy wykorzystać. Przy sposobie kodowania jak na rys. 1. nie można wprost wykorzystać najprostszego operatora krzyżowania polegającego na tym, że wybieramy losowo dwóch rodziców i losowy punkt podziału, a następnie pierwsze dziecko (pierwszy potomek) otrzymuje pierwszą część z pierwszego rodzica, a drugą część z drugiego, zaś drugie dziecko na odwrót. Będzie to bowiem bardzo często prowadzić do sytuacji pokazanych na rys. 2. Przy czym w przypadku dłuższych chromosomów znacznie częściej będą generowane nieprawidłowe dzieci, niż prawidłowe.

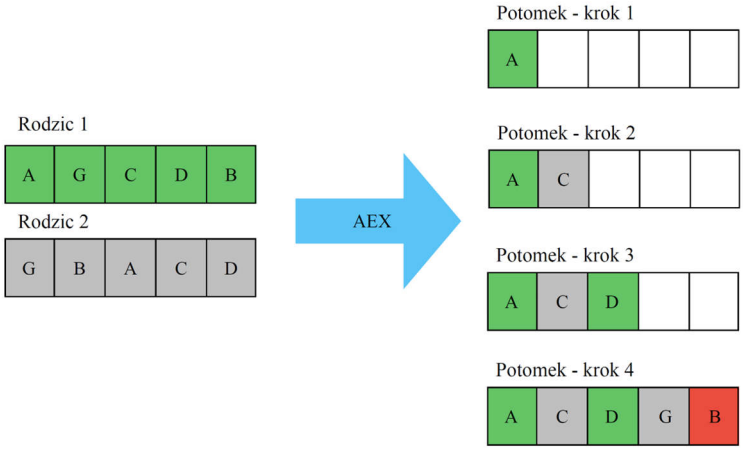
Zastosowaliśmy kilka operatorów krzyżowania:

- Operator dzielący połowicznie, oparty na dniach pracy,
- Operator dzielący połowicznie, oparty na zajętych stanowiskach,
- Operator dzielący połowicznie, oparty na zajętych stanowiskach dokonujący naprawy błędnego chromosomu,
- Operator dzielący połowicznie, oparty naprzemiennie na dniach pracy i zajętych stanowiskach,
- Operator AEX,
- Operatory z serii HgreX.

Na rys. 2 przedstawiony został operator krzyżowania oparty o podział połowiczny oparty na dniach pracy. Przy pomocy tego operatora krzyżowania poprawiamy dopasowanie wybranych przez pracowników konkretnych dni pracy oraz ilości dni pracy. W celu ulepszenia dystrybucji preferowanych stanowisk należy użyć operatora opartego na zajętych stanowiskach. Najlepsze dopasowanie osiągniemy poprzez naprzemienne stosowanie operatorów opartych na dniach oraz na stanowiskach. Ze względu na dużą ilość błędnych chromosomów które powstają przy takim podejściu, należy dodać naprawę takich chromosomów poprzez wyszukanie wolnych pracowników i wstawienie ich w miejsce błędnego genu. Naprawa ta może być wykonywana na różne sposoby. W niektórych operatorach krzyżowania jest ona ich nieodłączną częścią.

Rysunek 2. Operacja krzyżowania chromosomu której wynikiem jest jeden prawidłowy potomek oraz jeden nieprawidłowy

Zostały opracowane różne operatory krzyżowania implementujące w sobie mechanizm naprawy błędnych genów (zwany też mechanizmem unikania kolizji) [13,14]. Jeden z najpopularniejszych to operator AEX (ang. alternating edges crossover). Operator ten wybiera jeden element z pierwszego rodzica a następnie z drugiego rodzica ten element, który następuje po elemencie ostatnio wybranym z rodzica pierwszego. Operacja ta wykonywana jest na przemian między dwoma rodzicami, aż do wygenerowania całego potomka, albo do wystąpienia kolizji. Kolizja występuje, gdy zgodnie z tą procedurą został w pewnym momencie wybrany element, który w potomku już występuje. W takim przypadku zamiast wybranego elementu do potomka trafia losowo wybrany element z elementów jeszcze w nim nieobecnych. Następnie podstawowa procedura jest kontynuowana. Na rys. 3 przedstawione zostało działania operatora AEX.



Rysunek 3. Przykładowe działanie operatora AEX (rysunek nie uwzględnia możliwego wystąpienia kolizji)

Szczególnie ciekawe są stanowiące rozwinięcie operatora AEX operatory z serii H…X, jako że zawierają one w sobie elementy przeszukiwania lokalnego [15-17]. Jednakże ich użycie wymaga zdefiniowania pojęcia sąsiedztwa w postaci kosztu przejścia między dwoma elementami. Koszt ten jest bardzo dobrze zdefiniowany np. w zagadnieniu komiwojażera (ang. Traveling Salesmen Problem), gdzie jest on po prostu odległością między miastami reprezentowanymi przez dane elementy. Operator HRndX zamiast wybierać elementy z rodziców naprzemiennie, losowo decyduje z którego rodzica zostanie wybrany kolejny element. Operator HProX z kolei dokonuje wyboru

rodzica z prawdopodobieństwem większym dla mniejszego kosztu przejścia między elementem z danego rodzica, a ostatnim elementem dodanym do potomka. Istnieje również operator w pełni deterministyczny – HGreX. Operator ten zawsze wybierze rodzica, dla którego koszt przejścia do ostatniego elementu potomka jest mniejszy. W przypadku, gdy pojawi się konflikt (wybrany element, który chcemy dodać istnieje już w potomku) należy wstawić zamiast niego losowy element, którego jeszcze nie ma w nowym chromosomie (dokładnie tak samo, jak w operatorze AEX).

Istotny problem w zagadnieniu układania harmonogramów pracy stanowi zdefiniowanie kosztu przejścia. Dlatego naszą ideą było zastąpienie wyboru minimalizującego koszt przejścia wyborem rozwiązania lokalnie bardziej optymalnego. Przykładowo, jeśli w pewnym dniu pracy danego pracownika możliwa do wyboru pozycja z jednego rodzica reprezentuje zmianę pierwszą, a z drugiego zmianę drugą, to wybieramy z większym prawdopodobieństwem tą pozycję, która jest zgodna z preferencjami pracownika odnośnie zmiany, na której chciałby w tym dniu pracować.

Nie jest celowe wykorzystanie jedynie samego przeszukiwania lokalnego w oparciu o wiedzę domenową do ułożenia optymalnego harmonogramu dla dużej organizacji, dlatego, że w ten sposób znalezione by został najprawdopodobniej jedynie minimum lokalne, gorsze od tego, które jest w stanie znaleźć algorytm genetyczny. Samo przeszukiwanie lokalne ma bowiem bardzo małe możliwości eksploracji przestrzeni rozwiązań [7].

Istotnym zagadnieniem jest tu również optymalny dobór proporcji przeszukiwania lokalnego i globalnego. Zbyt duży udział przeszukiwania lokalnego jest niewskazany, nie prowadzi bowiem do uzyskania dobrych wyników, jednak mały udział może pomóc znaleźć lepsze rozwiązanie. Analizowaliśmy szczegółowo to zagadnienie w naszej poprzedniej pracy „Local Search in Selected Crossover Operators"[4] w odniesieniu do optymalizacji rozmieszczenia towarów w magazynie i problemu komiwojażera. W tamtej pracy analizowaliśmy także wykorzystanie przeszukiwania lokalnego w innych operatorach krzyżowania, jak AEX i KPoint. Operatory te nie zawierają same w sobie komponentu przeszukiwania lokalnego, ale można go użyć w różnych scenariuszach, np. jako sposobu rozwiązywania konfliktów przy budowaniu potomka z rodziców. Aktualnie analizujemy możliwość wykorzystania opisanych tam wyników w układaniu harmonogramów pracy.

Należy jeszcze wspomnieć, że do stworzenia potomka można wykorzystać więcej niż dwóch rodziców. Zwiększenie tej liczby poprawia szybkość zbieżności algorytmu w większości operatorów krzyżowania, a w szczególności w operatorze przedstawionym na rysunku 2. Opis tego problemu przekracza ramy niniejszego artykułu. Zainteresowanych odsyłamy to naszej publikacji „Optimization of Warehouse Operations with Genetic Algorithms" [5].

2.5. Operatory mutacji

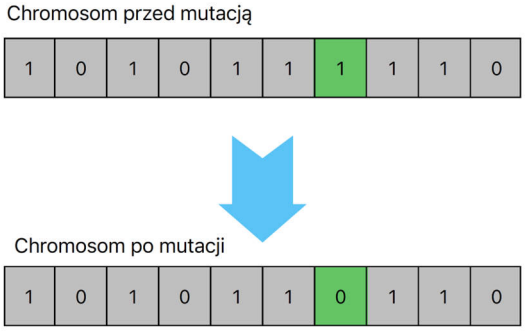
Celem mutacji jest przeprowadzenie zmiany w obrębie jednego chromosomu. Stosuje się ją po to, aby uniknąć takiej sytuacji, że żaden chromosom w populacji nie zawiera takich genów na wybranych pozycjach, jakie prowadzą do znalezienia najlepszego rozwiązania problemu i nie ma skąd tego genu wziąć w operacjach krzyżowania.

Mutacja zazwyczaj jest implementowana jako losowa zmiana w losowych miejscach losowych chromosomach. W pierwszej kolejności wybrane zostaje x losowych chromosomów nie zależnie od ich wartości funkcji dopasowania, a następnie jest wykonywana losowa operacja na ich genach. Przy czym operator mutacji musi być dostosowany do sposobu reprezentacji problemu w chromosomie.

W przypadku niezależnej reprezentacji genów (gdzie zmiania jednego genu nie wymusza zmiany innego genu) najczęściej używanymi operatorami mutacji są to zamiana genów miejscami albo odwrócenie bitów. Na rysunku 4 przedstawiony został operator mutacji stosujący odwrócenie bitu.

Natomiast operator oparty na losowej zamianie dwóch genów w chromosomie iteruje po całej populacji i zgodnie ze zdefiniowanym prawdopodobieństwem dokonuje mutacji. Niestety, ponieważ operacja taka może spowodować uszkodzenie chromosomu, przed dokonaniem permanentnej zmiany w chromosomie należy sprawdzić, czy będzie on nadal poprawny po tej zamianie.

DOI: https://doi.org/10.53052/9788367652292.22



Rysunek 4. Mutacja poprzez odwrócenie bitu

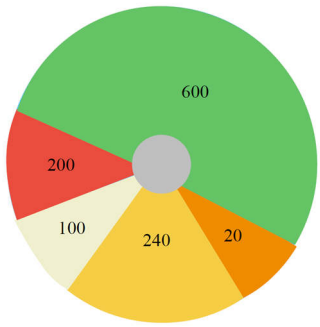
W przypadku zależnej reprezentacji genów opracowano kilka specjalnych operatorów mutacji. Na podstawie przeprowadzonych testów wynika, że jednym z najlepszych jest RSM (Reverse Sequence Mutation). Operator ten wyznacza losowo dwa punkty w chromosomie, a następnie odwraca kolejność elementów zawartych na pozycjach pomiędzy tymi dwoma punktami.

Stosowane prawdopodobieństwa wykonywania mutacji są z reguły niskie. Bowiem jej działaniem ubocznym jest dezorganizacja porządku, który stara się zbudować operacja krzyżowania, a tego staramy się uniknąć (nie dotyczy pewnych szczególnych wersji algorytmów genetycznych, których tu jednak nie używamy).

2.6. Operatory selekcji

Celem operatora selekcji jest wybranie osobników które następnie zostaną wykorzystanie podczas działania operatora krzyżowania w celu wygenerowania potomków. Wybór ten może odbywać się w sposób deterministyczny, częściowo losowy z preferencją lepszych osobników (selekcja turniejowa, ruletkowa i ich warianty) lub w pełni losowy.

Wybór deterministyczny. Metoda ta polega na wybraniu pewnej liczby osobników z najlepszym wynikiem funkcji dopasowania. Jako że jest to rozwiązanie całkowicie deterministyczne, metoda ta odznacza się szybką zbieżnością, lecz słabą eksploracją rozwiązań. Czyli w początkowych epokach otrzymujemy szybki wzrost jakości osobników, ale rozwiązanie możne utknąć (i przy większych problemach prawie zawsze utknie) w minimum lokalnym. Dzieje się tak na wskutek przedwczesnego wyeliminowania osobników o gorszej wartości funkcji dopasowania (w zależności, czy funkcję dopasowania minimalizujemy, czy maksymalizujemy może to być jej wartość wyższa lub niższa), których fragmenty chromosomu mogą jednak zawierać dobre rozwiązania częściowe nieobecne w lepszych osobnikach.



Rysunek 5. Dystrybucja prawdopodobieństwa wyboru podczas selekcji. Etykieta zawiera wartość osobnika.

Selekcja Ruletkowa. W selekcji ruletkowej na wirtualną tarczę, na której znajdują się identyfikatory każdego osobnika rzucamy wirtualną kulką do gry. Na pole którego osobnika wypadnie, tego wybieramy. Pole odpowiadające każdemu osobnikowi zajmuje tym większą część koła im wartość funkcji dopasowania (fitness) tego osobnika jest większa (zakładamy tu maksymalizację funkcji dopasowania), co zwiększa prawdopodobieństwo wyboru lepszych osobników. W podstawowym wariancie selekcji ruletkowej zależność między wartością funkcji dopasowania danego osobnika, a przydzielonym mu polem jest liniowa. Może ona jednak być wyrażona dowolną funkcją monotoniczną, zależną od tego jak bardzo (lub ja mało) chcemy premiować lepsze osobniki. Choć selekcja turniejowa i ruletkowa działają na innej

zasadzie, to z praktycznego punktu widzenia są niemal równoważne przy odpowiednim doborze liczby kandydatów w selekcji turniejowej i postaci funkcji odwzorowującej fitness osobnika na jego pole na kole ruletki. Graficzna reprezentacja selekcji ruletkowej znajduje się na rys. 5.

Selekcja Turniejowa. Selekcja turniejowa polega na tym, że najpierw z całej populacji losujemy pewną liczbę kandydatów, osobno na każdego rodzica, a następnie wybieramy tego z nich, który ma najlepszą wartość funkcji dopasowania. Jeśli liczba kandydatów będzie równa liczbie osobników w populacji, to jest to równoważne wyborowi deterministycznemu. Wraz z malejącą liczbą kandydatów coraz większą szansę zostania rodzicem mają słabsze osobniki. Optymalną liczbą jest często kilka kandydatów (4-8). Przy dwóch kandydatach selekcja jest już słaba, a przy jednym kandydacie jest to równoważne wyborowi w pełni losowemu.

2.7. Operatory eliminacji

Operator ten eliminuje osobniki, które nie przechodzą do kolejnej epoki. Można więc go też traktować jako operator selekcji osobników przechodzących do kolejnej epoki. Są tu stosowane trzy podstawowe rozwiązania:

1. Do następnej epoki przechodzą tylko dzieci (wówczas musi ich być tyle ile wynosi liczebność populacji)
2. Sortujemy razem dzieci i rodziców i z tej całości do następnej opoki przechodzi tyle osobników, ile wynosi liczebność populacji
3. Do następnej epoki przechodzi tylko kilku najlepszych rodziców i całą resztę uzupełniamy dziećmi (tzw. elityzm).

Ważne jest, żeby tzw. selection pressure, czyli siła, z jaką preferujemy lepsze osobniki była łącznie rozpatrywana dla operatora selekcji rodziców oraz dla operatora eliminacji osobników które nie przejdą do kolejnej epoki. Ta siła powinna być optymalna. Zbyt słaba niepotrzebnie wydłuża proces. Zbyt mocna możne powodować, że optymalne rozwiązanie nie zostanie znalezione na skutek uszczuplenia materiału genetycznego populacji. Co więcej, można stosować zmienną siłę selekcji wzrastającą wraz z postępem optymalizacji. Wynika to z tego, że z postępująca optymalizacją maleją różnice między osobnikami i warto wtedy mocniej promować najlepsze osobniki.

2.8. Przeszukiwanie lokalne

Wiedzę domenową w ramach przeszukiwania lokalnego może zostać wbudowana w działanie algorytmu genetycznego na różne sposoby, a pierwszym wyborem jest jej zaimplementowanie w operatorze krzyżowania. Najprostszym przykładem wiedzy domenowej jest w tym wypadku sytuacja, że jeżeli dany pracownik chce mieć jakiś konkretny czas wolny, to należy tak pokierować proces poszukiwań, by mu to umożliwić (nie patrząc w tym momencie na całą resztę harmonogramu).

Nie jest możliwe wykorzystanie jedynie samego przeszukiwania lokalnego w oparciu o wiedzę domenową do ułożenia optymalnego harmonogramu, dlatego, że w ten sposób znaleziono by się jedynie w minimum lokalnym, gorszym od tego, które jest w stanie znaleźć algorytm genetyczny. Samo przeszukiwanie lokalne ma bowiem bardzo małe możliwości eksploracji przestrzeni rozwiązań.

W końcowym etapie próbowaliśmy wykorzystać algorytm n-opt (a dokładnie 2-opt) do poprawy otrzymanego rozwiązania. Aczkolwiek nie został on zastosowany w końcowym rozwiązaniu ze względu na to, że konieczność uwzględnia ograniczeń (np. pracownik musi mieć określoną przerwę pomiędzy dwoma zmianami i innych) przy każdych zmianach dokonywanych przez n-opt czyniła ten proces bardzo skomplikowanym.

2.9. Programowanie z ograniczeniami

Programowanie z ograniczeniami (ang. constraint programming) jest metodą rozwiązywania problemów kombinatorycznych mającą na celu znalezienie wykonalnych rozwiązań z bardzo dużego zbioru kandydatów, gdzie problem może być opisany poprzez dowolne ograniczenia. Często wykorzystuje techniki backtrackingu i propagacji ograniczeń. Programowanie z ograniczeniami odnosi się do opracowania planu, w tym przypadku harmonogramu pracy (a nie programowania w języku komputerowym). Opiera się na wykonalności (znalezieniu wykonalnego rozwiązania) i koncentruje się na ograniczeniach i zmiennych, a nie na funkcji celu. Na tym polega istotna różnica między programowaniem z ograniczeniami, a algorytmami genetycznymi. Przykładem problemu, który nadaje się do programowania z ograniczeniami jest harmonogramowanie czasu pracy pracowników.

Warto tu porównać programowanie z ograniczeniami z algorytmami genetycznymi. Zaletą algorytmów genetycznych jest ich duża zdolność eksploracji przestrzeni rozwiązań; mogą eksplorować dużą przestrzeń rozwiązań i znajdować dobre rozwiązania nawet w przypadku złożonych problemów. Kolejną zaletą jest elastyczność: mogą wykorzystywać różne funkcje celu i ograniczenia. Wadą algorytmów jest czas potrzebny do znalezienia rozwiązania. Mogą wymagać wielu pokoleń (iteracji), aby zbiec się do optymalnego lub zadowalającego rozwiązania, dodatkowo następuje najczęściej spowolnienie procesu optymalizacji przy zbliżaniu się do minimum.

Zaletami programowania z ograniczeniami jest możliwość znalezienia optymalnego rozwiązania dobrze zdefiniowanych problemów, dzięki czemu nadaje się ono do sytuacji, w których precyzja jest niezbędna. Kolejna zaletą jest to, że metoda ta dobrze radzi sobie ze złożonymi ograniczeniami i zależnościami między zmiennymi. Jest efektywna w przypadku małych i średnich problemów.

Wadami programowania z ograniczeniami są problemy skalowalnością: tj. ze wzrostem wymiarowości i złożoności zagadnienia skuteczność spada. Wykazały to dokładnie nasze testy z pakietem OR Tools przy układaniu harmonogramów pracy. Ułożenie harmonogramu dla dużych organizacji trwało tak długo, że nie doczekaliśmy się końca i skutkiem tego metoda ta okazała się niepraktyczna do tych zagadnień. Kolejną wadą jest to, że napisanie od podstaw programu implementującego programowanie z ograniczeniami jest bardziej skomplikowane niż napisanie odpowiadającego programu implentującego algorytmy genetyczne.

Uwzględniając wady i zalety obu tych metod, warto spróbować ich połączenia. Choć nie jest to proste w odniesieniu do problemu harmonogramowania czasu pracy, to będzie to stanowiło element naszych dalszych prac. Jednak próby takie zostały już opisane w literaturze [11,12].

2.10. OR Tools

Narzędzie Google OR Tools [18] jest biblioteką implementującą programowanie z ograniczeniami. OR Tools dostarcza interfejs w najpopularniejszych językach programowania, w tym w C#.

W celu opracowania modelu układającego harmonogramy czasu pracy zgodnie z założeniami i ograniczeniami przeanalizowaliśmy możliwości wykorzystania biblioteki Google OR Tools. Cały proces rozpoczyna się od stworzenia obiektu CpModel i wprowadzeniu do niego wszystkich przygotowanych na wcześniejszych etapach założeń i ograniczeń.

```
var model = new CpModel();
var work = new BoolVar[constraints.EmployeeCount,
                                        constraints. constraints.DataRanges.Shifts.Count,
                                        constraints.DataRanged.Days.Count];
```

Pierwszymi definiowanymi ograniczeniami jest przypisanie maksymalnie jednej zmiany każdego dnia.

```
foreach (var shift in contstraints.DataDanged.Shifts)
{
      work[employee.Value, shift.Value, day.Value] = Model.NewBoolVar($"work{employee.Value}
                                                    _{shift.Value}_{day.Vaule}");
      temp[shift.Value] = work[employee.Value, shift.Value, day.Value];
}
AddContsraint(model, assumptions, "One shift per day", LinearExpr.Sum(temp) == 1);
```

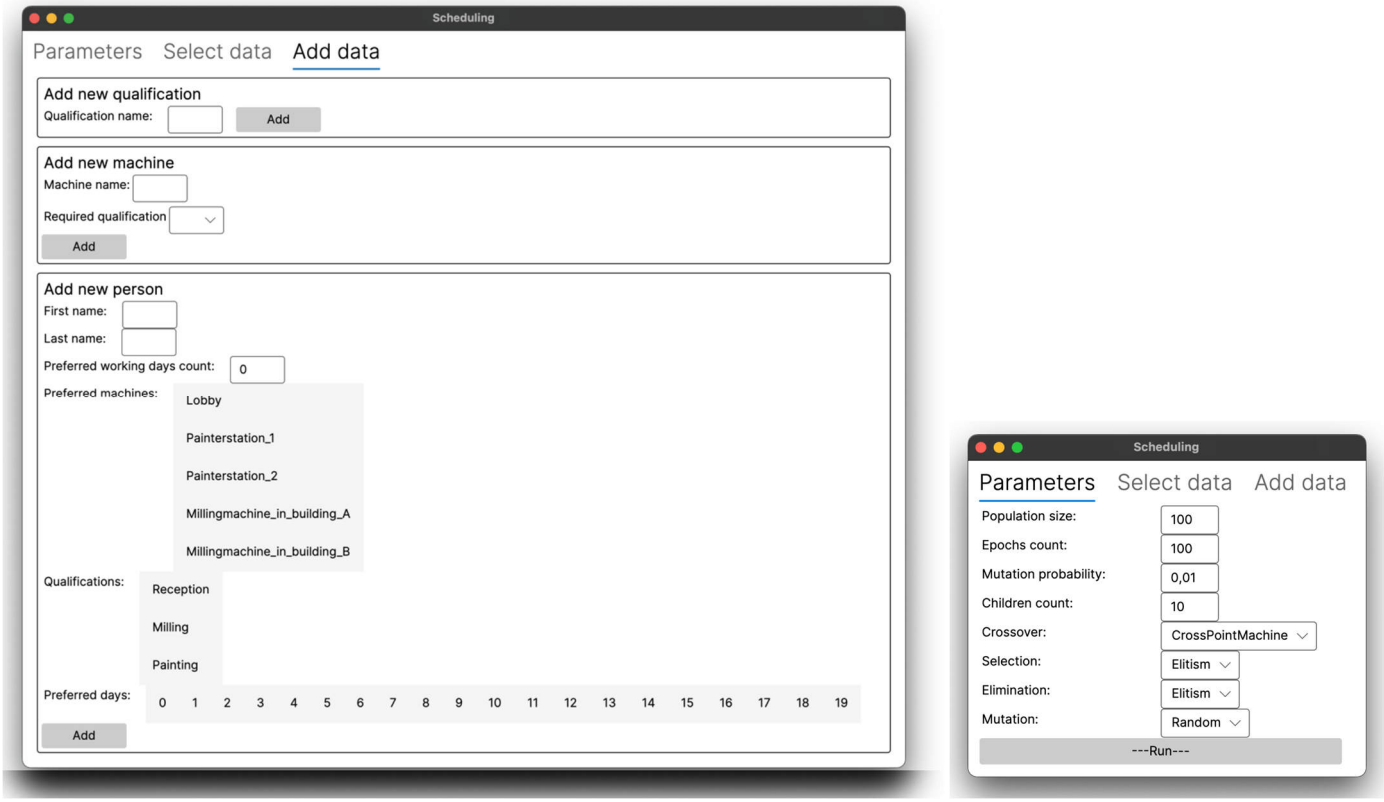
W celu dodania ograniczenia korzystamy z funkcji pomocniczej AddConstraint, która dodaje je zarówno do modelu jak i do zmiennej pomocniczej wykorzystywanej do późniejszej identyfikacji konkretnych ograniczeń w przypadku niepowodzenia znalezienia rozwiązania problemu. Kolejnym krokiem jest dodanie wszystkich dodanych już przez użytkownika grafików w danym okresie. Podaliśmy tylko przykładowe fragmenty kodu celem zobrazowania jego stylu. Osoby zainteresowane znajdą więcej szczegółów na ten temat w dokumentacji OR Tools [18].

Następnie tworzymy obiekt funkcji liniowej dla ograniczeń opcjonalnych, a następnie je do niego dodajemy zaczynając od preferencji pracowników i ich wniosków. Idąc dalej podajemy ograniczenia ilości następujących po sobie tych

samych zmian. Na koniec wszystkie funkcje ograniczeń są minimalizowane i tworzony jest solver, który odpowiada za rozwiązanie danego problemu z uwzględnieniem parametrów jego wywołania. W parametrach można zdefiniować m.in. ile czasu ma funkcja na znalezienie rozwiązania, czy szukać pierwszego spełniającego warunki rozwiązania czy tego, który jest najbardziej optymalny.

3. Komputerowa implementacja przedstawionego pomysłu

Prowadziliśmy testy i próbne implementacji poszczególnych metod w kilku systemach i środowiskach. Program, w którym zaimplementowaliśmy główne funkcjonalności harmonogramowania czasu pracy został stworzony w środowisku .NET, a fragment jego interfejsu jest pokazany na rys. 6. Kod źródłowy programu dostępny jest na stronie: https://drive.google.com/drive/folders/1QJ4UnhdZbfqnDVHJRpmydhhh7pBAZGtS



Rysunek 6. Interfejs programu, w którym zaimplementowaliśmy główne funkcjonalności harmonogramowania czasu pracy

4. Wnioski

Układanie harmonogramów pracy jest bardzo skomplikowanym zagadnieniem ze względu na dużą złożoność problemu, wiele ograniczeń i silne wzajemne powiązania między poszczególnymi zmiennymi. Jednakże automatyzacja tego procesu ma wiele korzyści. Przygotowane algorytmy są w stanie w bardzo krótkim czasie przeanalizować wiele założeń i parametrów i na ich podstawie ułożyć harmonogram pracy. Tym samym zostaje uwolniona znaczna część cennego czasu pracy osób, które to robiły ręcznie do tej pory, a które teraz będą mogły skupić się na innych kluczowych aspektach swojej pracy, podnosząc efektywność całej firmy. System układa harmonogramy pracy starając się w maksymalnym możliwym stopniu spełnić zarówno potrzeby organizacji, jak i potrzeby jej pracowników, stosownie do wag przypisanych poszczególnym potrzebom. Opisany tu system wymaga dalszego rozwoju, celem zaimplementowania zarówno bardziej złożonych warunków, jak i celem przyśpieszenie i poprawienia jego działania. Jednym z dalszych kierunków rozwoju jest opisane już w poprzednich sekcjach połączenie algorytmów genetycznych, przeszukiwania lokalnego i programowania z ograniczeniami.

DOI: https://doi.org/10.53052/9788367652292.22

Podziękowania

Praca została dofinansowa przez grant NCBiR projekt nr POIR.01.01.01-00-0302/22

Literatura

1. Affenzeller, M., Wagner, S., Winkler, S., Beham. A. Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications. CRC Press (2018)
2. Mudra S. Gondane, D. R. Zanwar. Staff Scheduling in Health Care System, OSR Journal of Mechanical and Civil Engineering (IOSRJMCE). Volume 1, Issue 6, PP 28-4 (2012)
3. Martin Fakt. A Genetic Algorithm for Personnel Scheduling in Vacation Seasons. LiTH-MAT-EX_2021 (2021)
4. M. Kordos, R. Kulka, T. Steblik, R. Scherer. Local Search in Selected Crossover Operators, LNCS, vol. 13352, pp. 369–382, ICCS (2022)
5. Kordos, M., Boryczko, J., Blachnik, M., Golak, S. Optimization of warehouse operations with genetic algorithms. Applied Sciences 10(14), 4817 (2020)
6. F. M. Howard, et. al. Implementation of an automated scheduling tool improves schedule quality and resident satistaction. PLoS One. 2020 Aug 11;15(8):e0236952. doi: 10.1371/journal.pone.0236952 (2020)
7. Tommy Clausen. Airport Ground Staff Scheduling. PhD Thesis, 2010
8. Benjamin Platten1, Matthew Macfarlane, David Graus, Sepideh Mesbah. Automated Personnel Scheduling with Reinforcement Learning and Graph Neural Networks. RecSys in HR'22: The 2nd Workshop on Recommender Systems for Human Resources (2022)
9. Ziyi Chen, Yajie Dou, Patrick De Causmaecker. Neural networked-assisted method for the nurse rostering problem. Computers & Industrial Engineering Volume 171, 108430 September (2022)
10. Fred N. Kiwanuka, Louay Karadsheh, Ja'far alqatawna, Anang Hudaya Muhamad Amin. Modeling Employee Flexible Work Scheduling As A Classification Problem. 25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (2021)
11. Stefano Di Alesio, Lionel C. Briand, Shiva Nejati, Arnaud GotliebAuthors Info & Claims.ombining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 25, Issue 1, Article No.: 4, Pages 1 – 37, https://doi.org/10.1145/2818640
12. Su Nguyen, Dhananjay Thiruvady, Yuan Sun, Mengjie Zhang. Genetic-based Constraint Programming for Resource Constrained Job Scheduling.arXiv:2402.00459v1 (2024)
13. Puljić, K., Manger, R. Comparison of eight evolutionary crossover operators for the vehicle routing problem. Mathematical Communications 18, 359–375 (2013)
14. Hassanat, A.B.A., Alkafaween, E. On enhancing genetic algorithms using new crossovers. Int. Journal of Computer Applications in Technology 55 (2017)
15. Al-Furhud, M.A., Ahmed, Z.H. Experimental study of a hybrid genetic algorithm for the multiple travelling salesman problem. Mathematical Problems in Engineering 20, 3431420 (2020)
16. Lin, B.L., Sun, X., Salous, S. Solving travelling salesman problem with an improved hybrid genetic algorithm. Journal of computer and communications 4(15), 98–106 (2016)
17. Santos, J., Ferreira, A., Flintsch, G. An adaptive hybrid genetic algorithm for pavement management. International Journal of Pavement Engineering 20(3) (2019)
18. Google OR Tools https://developers.google.com/optimization