Problem przydziału dla algorytmów genetycznych

Paulina Gryszczyk 250337

16 czerwca 2020

1 Algorytmy genetyczne

W przypadku metaheurystyk takich jak: symulowane wyżarzanie, przeszukiwanie z tabu, czy przeszukiwanie lokalne operuje się na jednym rozwiązaniu, które chcemy, aby było jak najlepsze. W algorytmach genetycznych przyjmuje się inną taktykę. Tworzona jest tzw. „populacja", o wybranym rozmiarze i przy każdej iteracji modyfikuje się ją tak, aby do następnego pokolenia przechodziły najlepsze osobniki. W ten sposób każde następne pokolenie posiada coraz to lepsze cechy, co pozwala na wybranie najlepszego osobnika z końcowej populacji. Modyfikacja populacji następuje poprzez krzyżowanie oraz mutację. Krzyżowanie polega na wyborze dwóch rodziców i ich paru genów, a następnie złączeniu ich. Nowo powstały chromosom nazywany jest dzieckiem dwóch krzyżowanych osobników, i dodawany do populacji. Do mutacji wybierany jest tylko jeden osobnik i dokonywane są pewne zmiany jego genotypu. Może to być zmiana jednego genu lub paru z nich. Do krzyżowania i mutacji wybierana jest określona liczba osobników, a wyniki tych operacji dodawane są do populacji początkowej danej iteracji. Następnie funkcja celu wybiera tyle najlepszych osobników ile wynosi rozmiar populacji. W niektórych przypadkach dla zachowania różnorodności genetycznej można też zachować w populacji osobniki o małej wartości funkcji celu lub osobniki niedopuszczalne, niedozwolone w zadaniu.

2 Problem przydziału

Problem przydziału został zdefiniowany, aby zwiększyć efektywność pracy oraz zminimalizować koszty i marnowany czas lub zasoby. Jednym z bardziej znanych jest „Job-shop Scheduling Problem"(JSP). Polega on na tym, że jest do wykonania dana ilość zadań, składających się z określonej ilości operacji. Każda operacja musi zostać wykonana na jednej maszynie. Maszyny mogą wykonywać tylko jedną operację w tym samym czasie. Każdej maszynie są też przypisane operacje, które muszą zostać na niej wykonane. Załóżmy, że każdą operacje można opisać listą krotek (m, t), gdzie m to numer maszyny, a t czas potrzebny na wykonanie danej operacji na tej maszynie. Wtedy przykładowe dane wejściowe do zadania wyglądałyby tak: job0=[(0,3),(1,2),(2,2)], job1=[(0,2),(2,1),(1,4)], job2=[(1,4),(2,3)].

3 Źródło 1

Pierwszym omówionym tekstem będzie „A genetic algorithm for the Flexible Job-shop Scheduling Problem". Autorzy zamieszczonych tam badań F.Pezella, G.Morganti, G.Ciaschetti stworzyli algorytm rozwiązujący „Flexible Job-shop Scheduling Problem". FJSP jest trudniejszą wersją opisanego wyżej JSP, w którym każda operacja miała przypisaną maszynę, na której powinna zostać wykonana. W tym przypadku algorytm musi również dopasowywać maszyny do operacji. Algorytm opisany w tym tekście jest algorytmem genetycznym.

3.1 Dane wejściowe

Jako dane wejściowe do algorytmu należy podać zbiór zadań, z których każde składa się z określonej liczby operacji O. Operacje danego zadania muszą być wykonywane w takiej kolejności, w jakiej zostaną podane. Dodatkowo należy jeszcze podać zbiór maszyn M. W poniższej tabeli przedstawione zostały przykładowe podziały zadań na operacje O_{ij}, gdzie i to numer zadania, a j to numer operacji danego zadania. W środku tabeli na przecięciu się konkretnej operacji z maszyną znajduje się czas wykonania tej operacji na danej maszynie.

	M_1	M_2	M_3	M_4
O_{11}	7	6	4	5
O_{12}	4	8	5	6
O_{13}	9	5	4	7
O_{21}	2	5	1	3
O_{22}	4	6	8	4
O_{23}	9	7	2	2
O_{31}	8	6	3	5
O_{32}	3	5	8	3

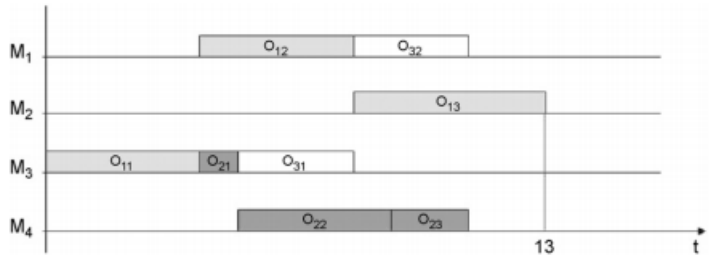
Rysunek 1: Tabela z przykładowymi danymi wejściowymi

3.2 Cel

Celem zadania jest dopasować każdą operację do właściwej maszyny oraz ułożyć w czasie wykonanie się zadań przypisanych do każdej maszyny, tak aby zminimalizować czas wykonywania się zadań tzw. „makespan" oraz aby operacje danego zadania były wykonywane po kolei.

3.3 Parametry algorytmu

1. Kodowanie
   Sposób reprezentacji rozwiązań to tabela składająca się z krotek (i, j, k). I odnosi się do numeru zadania, j do numeru operacji, a k maszyny. Przykładowa tabela mogłaby wyglądać w następujący sposób: [(1,1,3),(1,2,1),(2,1,3),(2,2,4), (3,1,3),(1,3,2),(3,2,1),(2,3,4)]. Jest to sposób reprezentacji poniższego rozwiązania.



Rysunek 2: diagram Gantta

2. Populacja początkowa
   Wyznaczanie populacji od której zaczyna się działanie algorytmu polegało na początku na znalezieniu dla każdej operacji maszyny z najmniejszym czasem pracy i dodaniu do reszty operacji przypisanych do tej maszyny, tego najmniejszego czasu. Na poniższym obrazku w kwadracie znajduje się wybrany obiekt, natomiast koszty operacji, które powinny zostać zwiększone zostały pogrubione.

	M_1	M_2	M_3	M_4
O_{11}	7	6	4	5
O_{12}	4	8	**9**	6
O_{13}	9	5	**8**	7
O_{21}	2	5	**5**	3
O_{22}	4	6	**12**	4
O_{23}	9	7	**6**	2
O_{31}	8	6	**7**	5
O_{32}	3	5	**12**	3

Rysunek 3: diagram Gantta

Jednak sposób ten był bardzo uzależniony od kolejności w jakiej operacje i maszyny zostały wpisane do tabeli dlatego twórcy zmodyfikowali jeszcze ten sposób. Zmienili sposób wyboru najmniejszego czasu, na taki aby znajdować globalne minimum(w całej tabeli), a nie dla danej operacji. Drugą zmianą było to iż wprowadzili losową permutację zadań i maszyn w tabeli. Dzięki drugiej zmianie przestrzeń poszukiwań była lepiej eksplorowana. Połączenie tych dwóch sposobów zostało wykorzystane do zmieniania przydziału operacji do maszyn. Do zmiany sekwencjonowania zostały wykorzystane 3 inne metody: losowego wyboru zadania, MWR wstawiająca do harmonogramu jako pierwsze operacje, które pochodzą od zadania,

któremu zostało jeszcze do wykonania najwięcej pracy, oraz MOR, który wybiera najpierw operacje, które pochodzą od zadania, któremu została do wykonania największa ilość operacji.

3. Funkcja celu
   Do oceniania chromosomu została użyta funkcja zwracająca makespan, czyli całkowity czas wykonywania się operacji, w konfiguracji danego osobnika.

4. Selekcja
   Wybór osobników do reprodukcji następował na 3 sposoby. Pierwszym z nich był „binary tournament", polegający na wyborze 2 losowych osobników z populacji, a następnie przekazaniu dalej do reprodukcji tego z lepszą funkcją przystosowania. „n-size tournament" to druga z metod, w której chromosom do reprodukcji jest wybierany spośród n losowych. Ostatnia metoda to „linear ranking". Aby jej użyć należało najpierw posortować wszystkie osobniki według funkcji celu, przydzielić każdemu rangę od 1 do n(rozmiaru populacji). Najlepsze osobniki dostają rangę n, a te najsłabsze 1. Na końcu obliczane jest prawdopodobieństwo wyboru danego osobnika według wzoru:
   $$p_i = \frac{2r_i}{n(n+1)}, i = 1, ..., n. \tag{1}$$
   gdzie r_i to ranga danego ocenianego osobnika, a n to rozmiar populacji. Poprzez te 3 metody tworzony jest zbiór potencjalnych rodziców, z których w sposób losowy wybierane są osobniki do tworzenia następnego pokolenia.

5. Generowanie potomstwa
   Tworzenie nowych osobników następuje poprzez krzyżowanie oraz mutację. Każda z tych dwóch operacji została jeszcze podzielona na 2 kategorie: zmieniająca przydział operacji do maszyn, ale zachowująca sekwencję zdarzeń, i zmieniająca kolejność zadań. Krzyżowanie zmieniające przydział polegało na zamianie fragmentów przydziału pomiędzy rodzicami. Natomiast mutacja w tej kategorii zmieniała kawałek przydziału pojedyńczemu rodzicowi. Podczas zmiany sekwencjonowania zmieniana jest kolejność zadań, lecz tak aby zachować kolejność wykonywania się operacji tego samego zadania. Autorzy algorytmu przedstawili też 2 nowe sposoby krzyżowania i mutacji: Precedence Preserving Order-based crossover(POX) oraz Precedence Preserving Shift mutation(PPS). POX polega na wybraniu u pierwszego rodzica losowej operacji, skopiowaniu do 1 dziecka wszystkich operacji z zadania, do którego należała wybrana na początku operacja. Reszta zadań jest uzupełniana w takiej kolejności i takimi wartościami jak u drugiego rodzica. Analogicznie tworzone jest drugie dziecko, zaczynając od drugiego rodzica. Natomiast PPS polega na wybraniu operacji od rodzica i przesunięciu jej na inną pozycję. Oczywiście podczas tego należy uważać, aby nie naruszyć kolejności wykonywania się operacji danego zadania. Została też wprowadzona tzw. inteligentna mutacja, polegająca

na wyborze operacji z maszyny obciążonej największą ilością pracy, i przypisaniu tej wybranej operacji do maszyny o najmniejszym obciążeniu.

3.4 Wyniki

Algorytm był testowany dla różnych wielkości parametrów, a najlepsze wyniki zostały osiągnięte dla populacji wielkości 5000, 1000 generacji, przypisania w populacji początkowej tworzonego w 10% metodą globalnego minimum, a w 90% metodą losowej permutacji. Sekwencje populacji początkowej najlepiej wyznaczać w 20% w sposób losowy, w 40% metodą MWR oraz też w 40% metodą MOR. Jeśli chodzi o wyznaczanie potomstwa, to najlepsze wyniki zostały osiągnięte przy następującym podziale: 45% POX, 45% krzyżowanie przydziałów, 2% PPS, 2% mutacja przydziału oraz 6% inteligentna mutacja przydziału. Poniższa tabela obrazuje porównanie wyników algorytmu przedstawionego w tekście do 3 innych znanych algorytmów rozwiązujących ten sam problem.

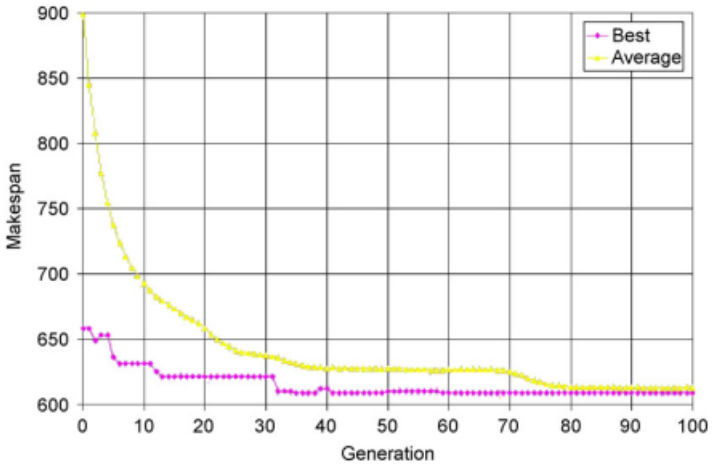
Name	*n*	*M*	LB	GA	Chen	*dev* (%)	GENACE	*dev* (%)	Jia	*dev* (%)
Mk01	10	6	36	40	40	0	41	+2.44	40	0
Mk02	10	6	24	26	29	+10.34	29	+10.34	28	7.14
Mk03	15	8	204	204	204	0	204	0	204	0
Mk04	15	8	48	60	63	+4.76	67	+10.45	61	1.64
Mk05	15	4	168	173	181	+4.42	176	+1.70	176	1.70
Mk06	10	15	33	63	60	−5.0	68	+7.35	62	−1.61
Mk07	20	5	133	139	148	+6.08	148	+3.38	145	4.14
Mk08	20	10	523	523	523	0	523	0	523	0
Mk09	20	10	299	311	308	−0.97	328	+5.18	310	−0.32
Mk10	20	15	165	212	212	0	231	+8.23	216	1.85
Average improvement						+1.96		+5.18		+1.45

Rysunek 4: Porównanie wyników

Kolumna n zawiera liczbę prac, M numer maszyn, LB najlepszą dolną granice, a GA to najlepszy „makespan" jaki udało się osiągnąć po 5 uruchomieniach. Kolumny Chen, Genace oraz Jia zawierają najlepsze wyniki 3 różnych algorytmów do porównania. Po każdej z tych kolumn znajduje się kolumna nazwana „dev", która zawiera informacje o odchyleniu w porównaniu do algorytmu omówionego w tekście. Odchylenie zostało wyliczone według wzoru:

$$dev = [(MK_{comp} - MK_{GA})/MK_{comp}] \cdot 100\%. \tag{2}$$

gdzie MK_{GA} to makespan omawianego algorytmu, a MK_{comp} to makespan drugiego algorytmu, do którego porównujemy. Na ostatnim załączonym wykresie możemy zauważyć, że średni makespan poprawia się bardzo szybko, a najlepszy makespan osiąga optymalną wartość 609 po 35 iteracjach.



Rysunek 5: Spadek najlepszego i średniego makespan'u

4 Źródło 2

Drugim analizowanym algorytmem będzie „Genetic Algorithms For Flowshop Scheduling Problems", którego autorami są Tadahiko Murata, Hisao Ishibuchi oraz Hideo Tanaka. W tym tekście autorzy rozważają problem, w którym mamy określoną ilość zadań i każde z nich składa się z m etapów, gdzie m to liczba maszyn. Każdy etap musi zostać wykonany na innej maszynie, dodatkowo kolejność wykonywania się tych etapów jest taka sama dla każdej maszyny. Celem zadania jest ustalić taką kolejność zadań, aby czas wykonywania się ich wszystkich był jak najkrótszy.

4.1 Parametry algorytmu

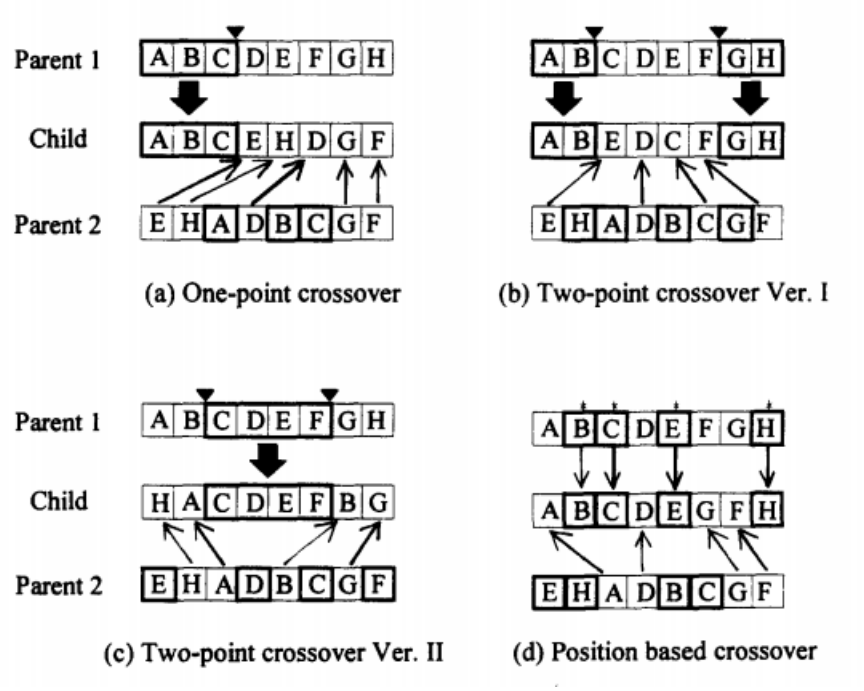
1. Kodowanie
   Do zapisu rozwiązań autorzy algorytmu użyli stringów, które przechowywały dane o tym, w jakiej kolejności powinny być wykonywane zadania. Na przykład string „ABCDEF" mówi nam o tym, że najpierw wykona się zadanie A, następnie B, C, D, E i na końcu F.

2. Selekcja
   W każdej iteracji jest wybieranych N_{pop} rodziców do reprodukcji, gdzie N_{pop} to rozmiar populacji. Rodzice ci są wybierani na podstawie następującej funkcji prawdopodobieństwa:

$$P_s(x_t^i) = \frac{[f_M(\Psi_t) - f(x_t^i)]^2}{\sum_{x_t^j \in \Psi_t}[f_M(\Psi_t) - f(x_t^j)]^2} \tag{3}$$

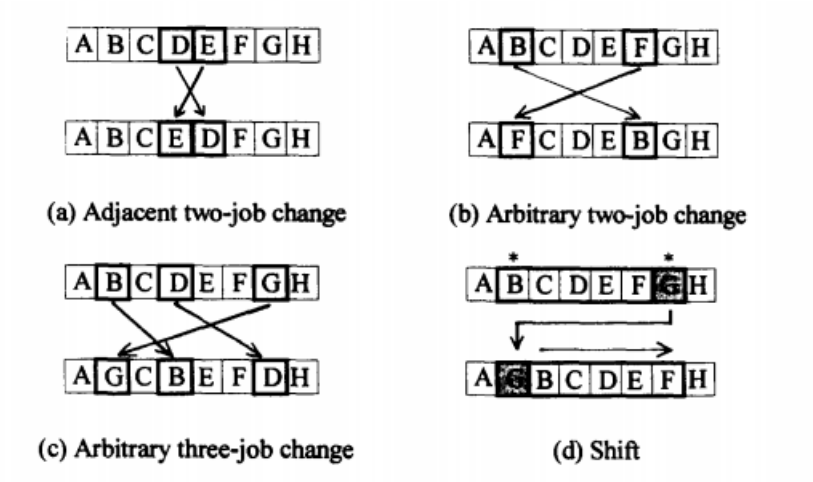
$f(x_t^i)$ to funkcja celu od danego rozwiązania (makespan), a $f_M(\Psi_t)$ to największy makespan w całej generacji.

3. Krzyżowanie
   Każde z 4 sposobów krzyżowania, omówionych tutaj uważają na to aby powstające nowe osobniki były rozwiązaniami dopuszczalnymi, czyli zawierały wszystkie zadania dokładnie raz. Pierwszym sposobem jest „jednopunktowe krzyżowanie", które polega na losowym wyborze jednego punktu u 1 rodzica, a następnie na przepisaniu do dziecka wszystkiego co jest po lewej stronie u 1 rodzica od wylosowanego punktu, a reszta pozycji jest przepisywana od 2 rodzica. „dwu-punktowe krzyżowanie" działa podobnie do jedno-punktowego, z tą różnicą, że w dwu-punktowym losujemy 2 punkty i na ich podstawie wyznaczamy od którego rodzica kopiujemy, który fragment. Jest też II wersja dwu-punktowego krzyżowania, która różni się od I wersji kierunkiem przepisywania od rodziców. Ostatnie krzyżowanie to „krzyżowanie oparte na pozycjach", polegające na wybraniu losowej ilości losowych genów u 1 rodzica, przepisaniu ich do tworzonego dziecka, a następnie uzupełnienie reszty genów genami 2 rodzica, których nowo tworzony osobnik jeszcze nie ma. Poniżej przedstawione są schematy tych 4 operatorów krzyżowania.



Rysunek 6: Sposoby krzyżowania

4. Mutacja
   Celem mutacji była zamiana kolejności zadań w każdym stringu utworzonym w procesie krzyżowania. Do mutacji również zostały utworzone 4 sposoby, które polegały na zamianie 2 sąsiadujących ze sobą zadań, 2 lub 3 losowych zadań lub usunięcie jednego zadania i dodanie go na nowe, losowe miejsce. Do mutacji również zostały zamieszczone schematy działania.



Rysunek 7: Sposoby mutacji

4.2 Zarys algorytmu

Na samym początku jest tworzona populacja początkowa z losowych permutacji wszystkich zadań. Następnie wybieranych jest N_{pop} (rozmiar populacji) par stringów, według funkcji prawdopodobieństwa. Do każdej z par jest zastosowana jedna z metod krzyżowania, lub jeśli nie, to jeden ze stringów z pary jest dodawany do następnego pokolenia w niezmienionym stanie. Po krzyżowaniu, każda z par jest poddawana również jednemu ze sposobów mutacji, a pod koniec usuwany jest jeden losowy osobnik z nowo utworzonej populacji, i jest zastępowany najlepszym stringiem z poprzedniej populacji. Algorytm kończy się w momencie, w którym zostanie spełniony warunek stopu.

4.3 Porównanie wyników dla różnych parametrów

W celu znalezienia parametrów dających najlepsze wyniki, autorzy uruchamiali testy dla różnych kombinacji tych parametrów. Sprawdzali również wpływ sposobów krzyżowania i mutacji na osiągane wyniki. Po przeprowadzeniu 100 testów z 20 zadaniami i 10 maszynami, doszli do wniosków, iż najlepiej jest ustawić wielkość populacji na 10, prawdopodobieństwo krzyżowania i mutacji na 1.0, przeprowadzać „dwu-punktowe krzyżowanie" oraz mutację polegającą na

zamianie miejsca losowemu zadaniu. Poniższy tabela przedstawia porównanie wyników dla różnych sposobów krzyżowania.

Table 1. Comparison of various crossover operators

	Number of evaluations		
Crossover operator	10,000	20,000	50,000
One-point crossover	100.81	100.53	100.15
Two-point: Ver. I	100.44*	100.19*	100.00*
Two-point: Ver. II	100.85	100.55	100.19
Two-point: Ver. III	100.66†	100.35†	100.03†
Position based: Ver. I	101.10	100.79	100.30
Position based: Ver. II	100.81	100.50	100.24
Edge recombination	104.95	103.91	102.52
Enhanced E.R.	105.14	104.12	102.99
Partially matched	100.69	100.45	100.17
Cycle crossover	100.84	100.50	100.15

*,† show the best result and the second best result in each column.

Rysunek 8: Porównanie operacji krzyżowania

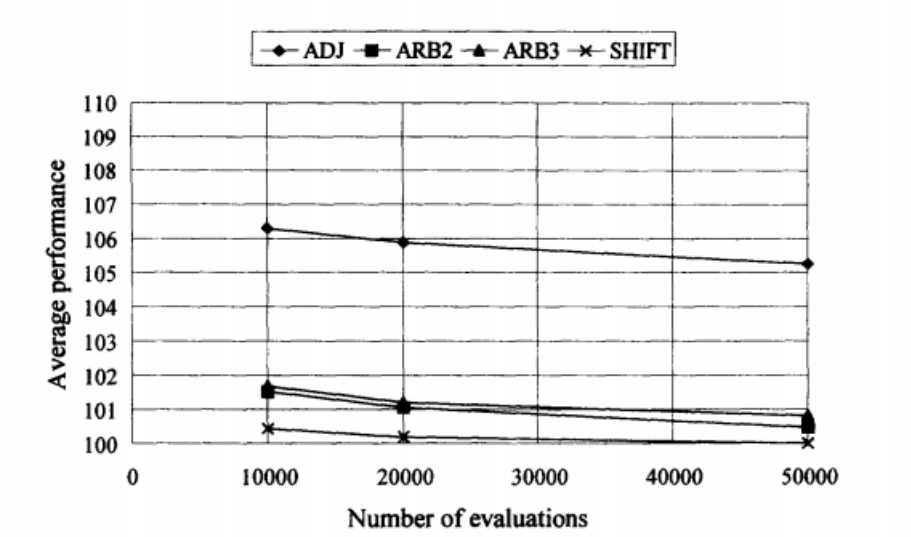
Możemy zauważyć, że najlepiej sprawdza się krzyżowanie dwu-punktowe. Dlatego właśnie autorzy algorytmu zdecydowali się na zastosowanie go. W drugiej tabeli znajdują się wyniki porównania już tylko 4 sposobów krzyżowania, na tych samych testach co w tabeli powyżej uruchamianych po 10 razy. Po połączeniu wyników z obydwu tabel, nadal krzyżowanie dwu-punktowe pozostaje najlepszym wyborem.

Table 2. Comparison of four crossover operators

	50,000 evaluations	
Crossover operator	Average	Standard deviation
One-point crossover	100.05	0.6835
Two-point: Ver. I	100.00	0.7047
Position based: Ver. I	100.19	0.7214
Edge recombination	102.33	1.1952

Rysunek 9: Porównanie 4 sposobów krzyżowania

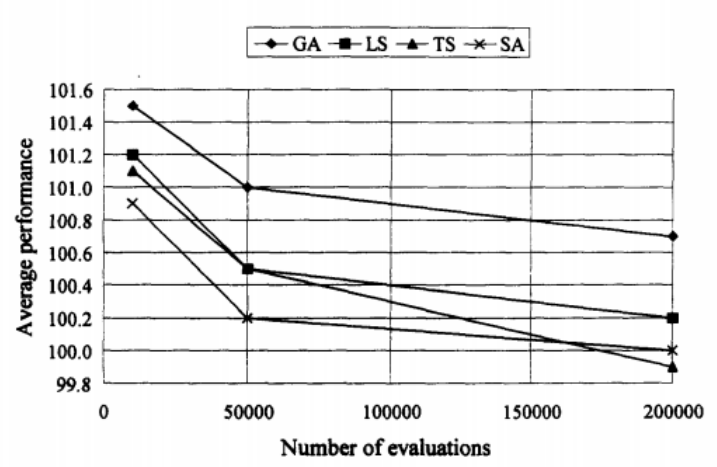
Testy przeprowadzane w celu ustalenia najlepszego sposobu mutacji, wykazały że sposób nazwany „Shift change" zapewnia najlepsze rezultaty. Pokazuje to poniższy wykres, na którym sposób ten nazwano „SHIFT" i przy jego użyciu osiągnięto najkrótszy całkowity czas wykonywania się zadań.



Rysunek 10: Porównanie 4 sposobów mutacji

4.4 Porównanie z innymi algorytmami

Opisywany algorytm genetyczny został porównany do algorytmów opartych na innych podejściach heurystycznych, takich jak: przeszukiwanie lokalne, przeszukiwanie z tabu oraz symulowane wyżarzanie. Na poniższym wykresie znajdują się wyniki porównania osiąganych najkrótszych czasów, dla każdego z tych algorytmów.



Rysunek 11: Porównanie 4 podejść heurystycznych

Na powyższym Rysunku 11 jako GA został oznaczony algorytm genetyczny, LS - przeszukiwanie lokalne, TS - przeszukiwanie z tabu, a SA - symulowane wyżarzanie. Aby polepszyć początkowy algorytm genetyczny autorzy połączyli go z innymi podejściami, tworząc tzw. algorytmy hybrydowe. Dwa z nich zostaną omówione w następnych podrozdziałach.

4.5 Algorytm oparty na przeszukiwaniu lokalnym i podejściu ewolucyjnym

Jednym z większych problemów podczas uruchamiania tego algorytmu był bardzo długi czas przeszukiwania lokalnego, co prowadziło do małej ilości utworzonych generacji. Dlatego autorzy zmodyfikowali algorytm tak aby nie sprawdzać wszystkich sąsiadów, ale tylko ich część np. 10%. Po tej zmianie algorytm składa się z następujących kroków: utworzenie populacji początkowej, przeszukiwanie lokalne dla aktualnej generacji (ale tylko w ustalonej liczbie procentów), wybór osobników do reprodukcji, krzyżowanie, mutacja i jeżeli nie został spełniony warunek stopu to powrót do przeszukiwania lokalnego.

4.6 Algorytm oparty na symulowanym wyżarzaniu i podejściu ewolucyjnym

Kroki tego algorytmu są takie same jak algorytmu opartego na przeszukiwaniu lokalnym, z tą różnicą, że tutaj jako 2 krok uruchamiane jest symulowane wyżarzanie. Autorzy postanowili ustawić stałą temperaturę dla każdego rozwiązania z aktualnej populacji. Aby ulepszyć algorytm podczas symulowanego wyżarzania wybieranych jest losowych k sąsiadów aktualnego rozwiązania, a najlepszy z nich jest ustawiany jako rozwiązanie w następnej iteracji algorytmu.

4.7 Podsumowanie algorytmów hybrydowych

Obydwa wyżej opisane algortmy hybrydowe były sprawdzane na 100 różnych testach, składających się z 20 zadań i 10 maszyn. Poniższa tabela zawiera wyniki tych testów, dla różnych α - ile procent sąsiadów jest sprawdzanych w procedurze przeszukiwania lokalnego, oraz dla różnych k - ilu losowych sąsiadów jest wybieranych z sąsiedztwa aktualnego rozwiązania podczas symulowanego wyżarzania.

Search algorithms	Number of evaluations		
	10,000	50,000	200,000
GA	101.48	101.03	100.71
GLS (100%)	101.14	100.14	99.85†
GLS (75%)	101.05	100.12*	99.82*
GLS (50%)	101.04†	100.18	99.92
GLS (25%)	101.02*	100.19	99.97
GLS (10%)	101.16	100.28	100.01
GLS (5%)	101.25	100.52	100.24
GSA ($k = 1$)	101.25	100.25	99.92
GSA ($k = 2$)	101.18	100.20	99.93
GSA ($k = 4$)	101.21	100.22	99.96
GSA ($k = 6$)	101.10	100.12†	99.87
GSA ($k = 8$)	101.26	100.23	99.92
GSA ($k = 10$)	101.27	100.17	99.92
GSA ($k = 20$)	101.51	100.20	99.94

*,† show the best result and the second best result in each column.

Rysunek 12: Wyniki algorytmów hybrydowych z różnymi parametrami

Jak możemy zauważyć GLS(Genetic local search) najlepiej się sprawdza dla α ustawionego na 75%, a przy GSA(Genetic simulated annealing) najlepiej jest ustawić k na 6. Jeśli porównamy wyniki zwykłego algorytmu genetycznego do wcześniejszych dwóch hybrydowych to poprawa wyników będzie zauważalna, co potwierdza dodatkowo poniższy wykres.

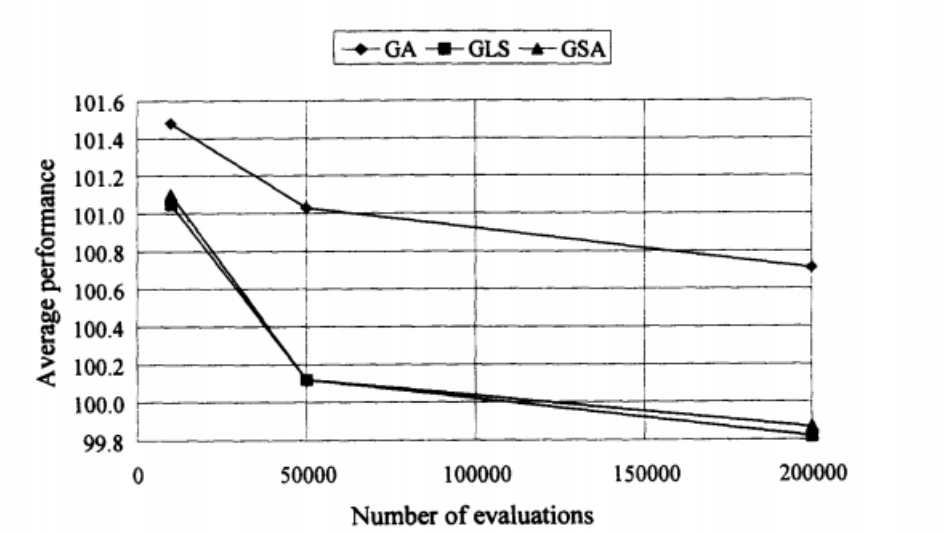


Fig. 6. Simulation results by GA, GLS ($\alpha = 75\%$) and GSA ($k = 6$).

Rysunek 13: Porównanie algorytmu genetycznego do hybrydowych

5 Główne problemy 1 algorytmu

Podczas rozwiązywania FJSP jednym z problemów był fakt, iż nie był to zwykły JSP i trzeba było dodatkowo ustalać przydział operacji do maszyn. Drugim napotkanym problemem był sposób wyznaczania populacji początkowej. Okazało się, że sposób wybrany przez twórców algorytmu na samym początku był za bardzo uzależniony od kolejności wpisywania operacji i maszyn, dlatego musiał on zostać zmieniony. Aby pozbyć się tej zależności, zastosowano tzw. globalne minimum, które w żaden sposób nie było zależne od wpisywania. Do generowania potomstwa zostało utworzonych bardzo wiele sposobów, co stworzyło kolejny problem z ustaleniem, które z nich wykorzystać oraz jaka część nowych osobników będzie tworzona poprzez, który z nich. Aby to ustalić trzeba było przeprowadzić wiele testów, na podstawie których ustalono najlepsze proporcje.

6 Główne problemy 2 algorytmu

W tym algorytmie autorzy zdecydowali się na zastosowanie tylko jednego sposobu krzyżowania i jednego sposobu mutacji w docelowym algorytmie rozwiązującym zadany problem. Dlatego spośród 10 sposobów krzyżowania, oraz 4 mutacji trzeba było wybrać po jednym najlepszym. W tym celu przeprowadzili testy, które pokazały, które sposoby najlepiej zastosować. Aby osiągnąć jeszcze lepsze wyniki, początkowy algorytm genetyczny został połączony z innymi podejściami heurystycznymi. Podczas implementacji „local search'a" okazało się, że czas wykonywania się go przy każdej iteracji jest tak duży, że zmniejsza w znacznym stopniu ilość generacji. Dlatego przeszukiwanie lokalne zostało zmienione w taki sposób aby znajdować tylko 10% sąsiadów. Przy drugim podejściu wstrzykniętym do algorytmu - symulowanym wyżarzaniu, także zostały wprowadzone modyfikacje. K sąsiadów aktualnego rozwiązania było wybieranych losowo, a następnie najlepszy z nich zostawał kandydatem na rozwiązanie w następnej iteracji algorytmu. Po testach, k zostało ustawione na 6.

7 Podsumowanie

Każdy z omówionych algorytmów rozważał inną modyfikację problemu przydziału, dlatego też przy każdym z nich pojawiły się różne problemy i zostały wprowadzone różne parametry. Autorzy każdego z nich przeprowadzili wiele testów, tak aby dobrać jak najlepsze parametry, lub jak w przypadku 2 omawianego algorytmu, aby połączyć 2 różne podejścia heurystyczne. Wnioski wyciągnięte, z każdego z tych testów, zostały wykorzystane podczas implementacji końcowych algorytmów, które następnie były porównywane z innymi sposobami rozwiązania tego samego problemu. W przypadku FJSP okazało się, że przedstawiony algorytm daje lepsze wyniki, niż 3 inne znane algorytmy genetyczne. Aby osiągnąć zadowalające wyniki podczas rozwiązywania 2 problemu - FSP, trzeba było połączyć 2 różne podejścia. Poskutkowało to znaczącą poprawą wyników w porównaniu do algorytmu początkowego.

Bibliografia

[1] F. Pezzellaa, G. Morgantia, G. Ciaschettib. A genetic algorithm for the Flexible Job-shop Scheduling Problem, 2008

[2] Tadahiko Murata, Hisao Ishibuchi and Hideo Tanaka. Genetic Algorithms for Flowshop Scheduling Problems, 1996

[3] Zbigniew Michalewicz, David B. Fogel. Jak to rozwiązać czyli nowoczesna heurystyka, 2006