IMPLEMENTACJA I ANALIZA EFEKTYWNOŚCI ALGORYTMU GENETYCZNEGO DLA WYBRANEGO PROBLEMU OPTYMALIZACJI

Projektowanie efektywnych algorytmów

AUTOR: MACIEJ BRONIKOWSKI 248838
GRUPA PROJEKTOWA: ŚRODA 13:15
PROWADZĄCY: ANTONI STERNA

1 WSTĘP TEORETYCZNY

Przedmiotem projektu było zaprojektowanie Algorytmu Genetycznego. Algorytm ten jest metaheurystyką służącą do generowania dobrych jakościowo przybliżeń globalnej optymalizacji problemów, dla których przegląd pełnej przestrzeni stanów dla dużych instancji jest zbyt czasochłonny. Przeszukiwanie możliwych rozwiązań odbywa się za pomocą operatorów genetycznych takich krzyżowanie i mutacja oraz selekcji których działanie zaczerpnięte zostało z biologicznych mechanizmów selekcji naturalnej. Standardowy algorytm genetyczny można przedstawić za pomocą listy kroków:

1. Generowanie populacji początkowej
2. Ocena przystosowania chromosomów populacji
3. Dopóki warunki zatrzymania nie został osiągnięty
    3.1 Selekcja chromosomów populacji macierzystej
    3.2 Krzyżowanie
    3.3 Mutacja
    3.4 Ocena przystosowania
    3.5 Utworzenie nowej populacji
4. Wyprowadzenie najlepszego wyniku

1.1 GENEROWANIE POPULACJI POCZĄTKOWEJ

Wybór populacji początkowej jest ważnym elementem działania algorytmu. Należy zapewnić różnorodność między wygenerowanymi osobnikami. Dlatego dobrym rozwiązaniem jest generowanie całkowicie losowej populacji początkowej, jednak możliwe jest, że dodanie pojedynczych rozwiązań problemu według prostych heurystyk dających dobre początkowe rozwiązania mogą pozytywnie wpłynąć na rozwiązanie końcowe, jednak ten element został pominięty podczas implementacji, ponieważ wymagał by wielu dodatkowych testów implementacji.

1.2 OCENA PRZYSTOSOWANIA

Ocena przystosowania chromosomów odbywa się z użyciem funkcji przystosowania. Dla problemów maksymalizacji funkcją przystosowania jest zwykle optymalizowana funkcja. Dla problemów minimalizacji należy problem sprowadzić do problemu maksymalizacji.

1.3 WARUNEK ZATRZYMANIA

Dla problemów optymalizacji warunkiem zatrzymania może być upłynięcie określonego czasu działania algorytmu, osiągnięcie zadanego progu ilości iteracji, brak poprawy wartości optymalnej, lub jeżeli znana jest wartość optymalizowanej funkcji, osiągnięcie jej pewnego przybliżenia.

1.4 SELEKCJA

Selekcja chromosomów polega na wybraniu osobników do populacji macierzystej zgodnie z zasadami selekcji naturalnej. Oznaczać to będzie, że pewne ich cechy przetrwają w kolejnym pokoleniu (iteracji algorytmu). Metodą selekcji zaimplementowaną podczas projektu była selekcja turniejowa, dlatego tylko ona zostanie opisana. Innymi metodami selekcji mogą być selekcja rankingowa oraz metoda ruletki.

1.4.1 Selekcja turniejowa

Selekcja turniejowa polega na k – elementowej próbie losowej z populacji (k >= 2). Dla każdej próby wybierany jest osobnik z lepszą wartością przystosowania, który zostaje dodany do listy osobników użytych później do reprodukcji. W zaimplementowanym algorytmie wybrany został wariant dla k = 2.

1.5 Operator krzyżowania

Operator krzyżowania ma za zadanie połączenie cech obiecujących osobników wybranych podczas selekcji. Z populacji macierzystej należy dobrać dwóch rodziców, na których zostanie wykonana operacja krzyżowania w losowo wygenerowanym punkcie. Dla reprezentacji ścieżkowej problemu można wyróżnić pięć operatorów krzyżowania: PMX, OX, EX, SXX, PX. Poniżej zostanie opisane działanie operatorów PMX oraz OX, które zostały zaimplementowane podczas realizacji projektu.

1.5.1 Operator PMX

Proces działania operatora krzyżowania PMX zostanie przedstawiony na przykładzie dwóch osobników macierzystych. Na ich podstawie zostaną wygenerowane dwa osobniki potomne.

- p = (1 2 3 4 5 6 7 8 9)
- q = (5 3 6 7 8 1 2 9 4)

Pierwszym etapem zastosowania operatora jest dobór dwóch losowych punktów krzyżowania, gdzie drugi z nich musi być większy, lub równy pierwszemu. Dla przykładu operator PMX zostanie przedstawiony dla punktów krzyżowania [3, 6], gdzie indeksowanie rozpoczyna się od wartości 1.

Kolejnym etapem jest utworzenie chromosomów osobników potomnych r oraz s. Dla osobnika r należy przenieść wszystkie geny zawierające się w przedziale punktów krzyżowania z osobnika macierzystego q. Analogicznie należy przenieść cechy z rodzica p do potomka s. Osobniki potomne będą miały wtedy następujący wygląd:

- r = (_ _ | 6 7 8 1 | _ _ _)
- s = (_ _ | 3 4 5 6 | _ _ _)

Następnie wszystkie niewystępujące jeszcze cechy w osobnikach potomnych należy skopiować z osobników macierzystych z nieużytego jeszcze rodzica w tych miejscach na odpowiadających im indeksach.

- r = (_ 2 | 6 7 8 1 | _ _ 9)
- s = (_ _ | 3 4 5 6 | 2 9 _)

Dla nieużytych jeszcze cech należy utworzyć tabele odwzorowań, w której każdej z nieużytych cech zawartych w przedziale tworzonym przez punkty krzyżowania zostaje przypisana cecha z drugiego rodzica na tym samym indeksie:

- [3 => 6], [4 => 7], [5 => 8] dla potomka r
- [7 => 4], [8 => 5], [1 => 6] dla potomka s

Należy następnie wpisanie pozostałych cech według tabeli odwzorowań, to znaczy dla każdego z potomków r,s należy wstawić element po lewej stronie odwzorowania w miejsce elementu po prawej stronie odwzorowania. Daje to wynik:

- r = (_ 2 | 6 7 8 1 | 4 5 9)
- s = (8 _ | 3 4 5 6 | 2 9 7)

Jak widać na powyższym przykładzie, dla potomka r oraz s nie jest możliwe wstawienie jeszcze jednej cechy. Jeżeli taka sytuacja wystąpi, należy wygenerować kolejny poziom tabeli odwzorowań dla cech po prawej stronie jak zostało to zrobione za pierwszym razem. Tabela odwzorowań dla pozostałych cech będzie wyglądać w następujący sposób:

- [3 => 6 => 1]
- [1 => 6 => 3]

Takie odwzorowania należy powtarzać aż do uzyskania pełnej informacji o potomkach. Końcowo osobniki potomne będą wyglądać w następujący sposób:

- r = (3 2 | 6 7 8 1 | 4 5 9)
- s = (8 1 | 3 4 5 6 | 2 9 7)

Tak skonstruowany algorytm byłby jednak bardzo nieefektywny. Złożoność obliczeniowa wyniosłaby O(n^2) (nawet O(n^3) jeżeli wymagałoby to również wyszukiwania elementów). Jednak istnieje pewne uproszczenie tego algorytmu. Należy zaimplementować dodatkową tablicę przechowującą odwzorowanie cechy na indeks w której znajduje się dana cecha. Przykład zostanie przedstawiony już tylko na potomku r, w analogiczny sposób można utworzyć potomka s zamieniając argumenty funkcji.

Początkowo należy zainicjalizować potomka r wartościami z rodzica p. Tablice m należy zainicjować odwzorowaniem wartości w indeks. Zainicjowane struktury będą wyglądać w następujący sposób:

- r = (1 2 | 3 4 5 6 | 7 8 9)
- m = (1 2 3 4 5 6 7 8 9)

Następnie kolejno dla każdego indeksu i w przedziale krzyżowania należy zamienić element na indeksie i z cechą o tej samej wartości, która jest u rodzica q w miejscu indeksu i. Tą samą operację należy wykonać na tablicy odwzorowań wartości w indeksy, tak aby zawierała aktualne dane. Tablica odwzorowań skraca czas wyszukiwania elementów w tablicy permutacji potomka. Kolejne operacje będą wyglądać w następujący sposób:

Potomek r	Tablica odwzorowań m	Opis
(1 2 | 3 4 5 6 | 7 8 9)	(1 2 3 4 5 6 7 8 9)	Inicjalizacja
(1 2 | 6 4 5 3 | 7 8 9)	(1 2 6 4 5 3 7 8 9)	Zamiana cech 6 oraz 3
(1 2 | 6 7 5 3 | 4 8 9)	(1 2 6 7 5 3 4 8 9)	Zamiana cech 4 oraz 7
(1 2 | 6 7 8 3 | 4 5 9)	(1 2 6 7 8 3 4 5 9)	Zamiana cech 5 oraz 8
(3 2 | 6 7 8 1 | 4 5 9)	(6 2 1 7 8 3 4 5 9)	Zamiana cech 3 oraz 1

1.5.2 Operator OX

Proces działania operatora krzyżowania OX zostanie przedstawiony na takich samych przykładach jak dla operatora PMX.

- p = (1 2 3 4 5 6 7 8 9)
- q = (5 3 6 7 8 1 2 9 4)

Początkowa faza jest identyczna jak dla operatora PMX. Należy wybrać losowe dwa punkty krzyżowania oraz przenieść elementy w przedziale krzyżowania z rodzica do potomka. Osobniki potomne będą wyglądać w następujący sposób:

- r = (_ _ | 6 7 8 1 | _ _ _)
- s = (_ _ | 3 4 5 6 | _ _ _)

Następnie należy przejrzeć wszystkie cechy z rodzica jeszcze nie użytego i kolejne nieużyte jeszcze cechy należy wstawiać na kolejnych indeksach potomka. Utworzone w ten sposób dzieci będą miały następującą postać:

- r = (4 5 | 6 7 8 1 | 9 2 3)
- s = (8 1 | 3 4 5 6 | 2 9 7)

1.6 Operator mutacji

Mutacja ma za zadanie wprowadzenie pewnej dywersyfikacji w Algorytmie Genetycznym tak, aby zmniejszyć możliwość wpadania w maksima lokalne. Mutacja na każdym z osobników pokolenia potomnego może odbyć się z pewnym małym prawdopodobieństwem. Dla reprezentacji permutacyjnych problemu można wyróżnić min. operatory mutacji takie jak: inversion, insertion, displacement oraz transposition. W projekcie zostały zaimplementowane insertion oraz transposition, dlatego tylko one zostaną opisane poniżej.

1.6.1 Operator mutacji insertion

Działanie operatora mutacji insertion zostanie pokazane na poniższym przykładzie:

- p = (1 2 3 4 5 6 7 8 9)

Operacja insertion rozpoczyna się od doboru dwóch punktów mutacji. Przykład zostanie zademonstrowany na indeksach (3, 7). Elementy w przedziale [3, 6] należy przesunąć o 1 indeks w prawo (lub w lewo, jeżeli indeks pobieranego elementu jest mniejszy od indeksu miejsca do wstawienia), a następnie cechę o wartości 7 wstawić w miejsce o indeksie 3.

- p = (1 2 7 3 4 5 6 8 9)

1.6.2 Operator mutacji transposition

Działanie operatora zostanie przedstawione na tym samym przykładzie jak dla operatora insertion:

- p = (1 2 3 4 5 6 7 8 9)

Dla wygenerowanych losowo dwóch punktów mutacji (w tym wypadku jak w przykładzie powyżej będą to punkty (3, 7), należy dokonać zamiany elementów na tych indeksach.

Osobnik po tak przeprowadzonej mutacji będzie wyglądał w następujący sposób:

- p = (1 2 7 4 5 6 3 8 9)

2 IMPLEMENTACJA

Kod programu został podzielony na segmenty (filtry)

- Algorithms – algorytmy zaimplementowane w programie
- States – stany w których aplikacja może się znajdować (różne poziomy menu)
- Structures – struktury wykorzystywane podczas działania programu

2.1 STATES

W filtrze Satates znajdują się wszystkie możliwe stany działania aplikacji. Każdy poziom menu aplikacji (jak menu główne, lub stan uruchamiania algorytmu) zaimplementowany jest jako osobna klasa dziedzicząca po klasie AState. Dzięki użyciu polimorfizmu oraz stosu, umożliwia to łatwe przejścia między kolejnymi stanami aplikacji oraz powrót do wcześniejszych stanów.

2.2 STRUCTURES

W filtrze Structures zawarte są wszystkie klasy przechowujące dane potrzebne do działania algorytmu:

- WeightedTradiness – w klasie zawarte są informacji o aktualnie wybranych parametrach algorytmu, oraz wszystkie potrzebne informacje o instancji problemu. Dodatkowo klasa implementuje metodę obliczającą wartość sumarycznego ważonego opóźnienia według podanej kolejności oraz metodę generującą losową permutację problemu.
- Job - zawarte są tu informacje na temat pojedynczego zadania instancji problemu. Dodatkowo implementowana jest tu metoda obliczająca ważone opóźnienie dla jednego zadania, gdyby zostało wykonane w podanym w argumencie czasie.
- JobsOrder – klasa ta wykorzystywana jest do zapisu aktualnie najlepszego znalezionego rozwiązania. Zawiera informacje na temat permutacji zadań oraz całkowitą ważoną sumę opóźnień tej kolejności. Dodatkowo implementuje metodę wyświetlającą w przejrzysty sposób dane w instancji klasy zawarte.

2.3 ALGORITHMS

W filtrze Algorithms znajduje się klasa GeneticAlgorithm, implementująca działanie algorytmu Genetycznego. Zawarte są w niej min. metody:

- Konstruktor – generowanie informacji początkowych potrzebnych do prawidłowego działania algorytmu
- runAlgorithm – metoda uruchamiająca pętle główną algorytmu
- PMX – statyczna metoda implementująca krzyżowanie typu PMX
- OX - statyczna metoda implementująca krzyżowanie typu OX
- insertMutation - statyczna metoda implementująca mutacje typu insert
- transpositionMutation – statyczna metoda implementująca mutację typu transposition

2.4 Zmiana parametrów algorytmu

W klasie WeightedTradiness przechowywane są wszystkie parametry algorytmu. Zmiana czasu działania, wielkości populacji, współczynnik mutacji oraz współczynnik krzyżowania są wartościami liczbowymi (typu unsigned lub float).

Informacja o typ, jaki typ mutacji lub krzyżowania ma być zastosowany w algorytmie, przechowywany jest za pomocą wskaźnika do funkcji posiadającej odpowiednie parametry. Ułatwia to szybką zmianę funkcji i pozwala na uniknięcie potrzeby sprawdzania z każdym przebiegiem pętli algorytmu, który z typów mutacji/krzyżowania powinien zostać użyty.

3 Eksperyment

3.1 Plan eksperymentu

Dla 10 instancji problemu wielkości 100 uruchomione zostały wykonane zawarte poniżej testy:

- Wpływ wielkości populacji na wynik dla populacji wielkości (20, 100, 600). Współczynnik mutacji został ustawiony na wartość 0.1, a współczynnik krzyżowania na 0.7
- Wpływ współczynnika mutacji (0.01, 0.05, 0.1) dla najlepszej wielkości populacji (100) i współczynnika krzyżowania o wartości 0.8.
- Wpływ współczynnika krzyżowania (0.5, 0.7, 0.9) dla najlepszej wielkości populacji (100) i współczynnika mutacji 0.01.

Każdy z testów został wykonany dla czterech wariantów algorytmu:

- Krzyżowanie: PMX, mutacja: insertion
- Krzyżowanie: PMX, mutacja: transposition
- Krzyżowanie: OX, mutacja: insertion
- Krzyżowanie: OX, mutacja: transposition

Podczas każdego testu algorytm był uruchomiony przez 30 sekund. Oprócz wyniku ostatecznego pobierane były również wyniki dla czasu działania algorytmu przez kolejno 1s, 2s, 5s, 10s oraz 20s. Ponieważ dla kolejnych czasów działania algorytm nie był inicjalizowany od nowa, pozwoliło to uniknąć przypadków, gdzie generator losowy mógł wskazywać wartości pozwalające na osiągnięcie lepszych wyników podczas krótszego czasu działania algorytmu.

Po wykonaniu testów dla 10 instancji dla każdego przypadku testowego obliczona została wartość błędu względnego z użyciem wzoru:

$$\delta = \frac{\left| f - f_{opt} \right|}{f_{opt}}$$

a następnie została obliczona wartość średniego błędu względnego 10 instancji dla każdego przypadku testowego. Z uwagi na dużą ilość danych w kolejnych punktach zostaną przedstawione tylko wartości średnie dla każdego przypadku testowego.

3.2 Sposób pomiaru czasu

Pomiar czasu odbywa się z użyciem std::chrono::high_resolution_clock. Klasa ta pozwala na wyznaczanie bardzo małych odstępów czasu. Przykład wyznaczania odstępu czasu:

```
std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now();
//obliczenia, których czas chcemy zmierzyć
std::chrono::high_resolution_clock::time_point end = std::chrono::high_resolution_clock::now();
long long time = std::chrono::duration_cast<std::chrono::microseconds>(end - this->start).count();
```

Wynikiem jest czas podany w mikrosekundach.

3.3 Wyniki eksperymentu

3.3.1 Wpływ wielkości populacji na wynik

Czas	PMX insertion 20	PMX insertion 100	PMX insertion 600	PMX transposition 20	PMX transposition 100	PMX transposition 600
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	1.45927	0.61425	0.30931	1.05494	0.78211	0.77080
2	1.21099	0.20425	0.30931	1.00688	0.78211	0.77080
5	0.33479	0.18934	0.25139	0.87651	0.78211	0.77080
10	0.30742	0.18934	0.15482	0.69375	0.78211	0.74664
20	0.30742	0.14501	0.12603	0.69375	0.66598	0.74664
30	0.13291	0.14501	0.09041	0.69375	0.66598	0.56966

Tabela 1 Wpływ wielkości populacji na wynik dla operatora krzyżowania typu PMX

Czas	OX insertion 20	OX insertion 100	OX insertion 600	OX transposition 20	OX transposition 100	OX transposition 600
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	0.15099	57.43835	59.34896	0.14071	57.93646	60.32140
2	0.10666	55.35884	57.36592	0.07329	57.29881	57.11954
5	0.02911	52.40484	56.03096	0.04600	54.03485	55.99871
10	0.02172	51.10139	55.15798	0.04600	49.40818	54.00675
20	0.00658	48.75100	53.83356	0.04600	45.55575	53.36719
30	0.00658	48.12991	53.16995	0.04600	45.32539	52.12310

Tabela 2 Wpływ wielkości populacji na wynik dla operatora krzyżowania typu OX

3.3.2 Wpływ współczynnika mutacji na wynik

Czas	PMX insertion 0.01	PMX insertion 0.05	PMX insertion 0.1	PMX transposition 0.01	PMX transposition 0.05	PMX transposition 0.1
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	3.91084	1.97138	0.73418	1.39433	1.04577	0.62251
2	3.30528	1.49408	0.73418	1.22064	1.00144	0.57818
5	2.76517	0.98681	0.61312	0.86367	1.00144	0.57818
10	2.76517	0.21570	0.31349	0.68351	0.97248	0.57623
20	2.76517	0.17542	0.03306	0.68351	0.97029	0.57623
30	2.64765	0.15843	0.02600	0.66976	0.97029	0.57623

Tabela 3 Wpływ współczynnika mutacji na wynik dla operatora krzyżowania typu PMX

Czas	OX insertion 0.01	OX insertion 0.05	OX insertion 0.1	OX transposition 0.01	OX transposition 0.05	OX transposition 0.1
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	61.37483	62.57550	62.23646	61.84488	61.90882	61.81593
2	60.81361	61.53725	61.32098	59.66838	60.78635	59.89916
5	59.50965	59.34966	60.58072	58.37920	59.06284	57.98495
10	56.99639	58.64649	58.00430	57.74263	56.73349	56.67905
20	55.98932	56.82297	57.26911	57.19534	56.12977	56.15732
30	54.95361	56.17094	56.51606	55.76207	55.72803	55.34682

Tabela 4 Wpływ współczynnika mutacji na wynik dla operatora krzyżowania typu OX

3.3.3 Wpływ współczynnika krzyżowania na wynik

Czas	PMX insertion 0.5	PMX insertion 0.7	PMX insertion 0.9	PMX transposition 0.5	PMX transposition 0.7	PMX transposition 0.9
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	3.05899	2.64102	3.13347	1.47515	1.24284	1.40383
2	2.30949	2.56619	2.89257	1.43016	1.15783	0.91423
5	2.11515	2.16292	2.31186	0.88210	1.09496	0.88673
10	2.11515	2.04326	0.95386	0.72686	0.61304	0.83356
20	1.37801	2.04326	0.79090	0.67296	0.59775	0.83356
30	1.37801	2.04103	0.65428	0.67296	0.53967	0.83356

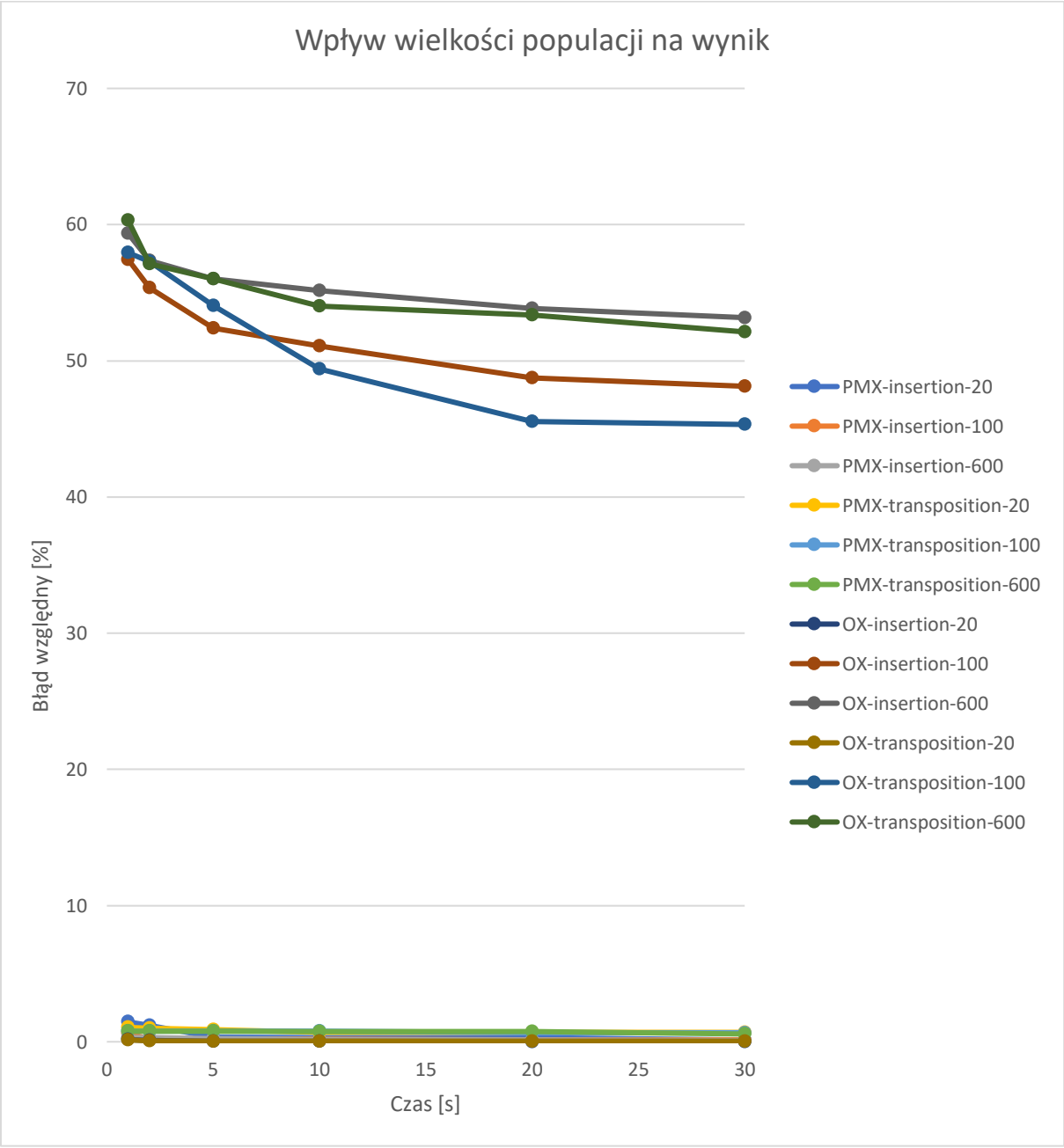
Tabela 5 Wpływ współczynnika krzyżowania na wynik dla operatora krzyżowania typu PMX

Czas	PMX insertion 0.5	PMX insertion 0.7	PMX insertion 0.9	PMX transposition 0.5	PMX transposition 0.7	PMX transposition 0.9
[s]	[%]	[%]	[%]	[%]	[%]	[%]
1	3.49809	3.27870	62.25894	1.28221	1.41711	63.93705
2	1.94206	3.01231	60.70664	1.09825	0.54280	62.76768
5	1.78415	1.42742	59.18423	1.06304	0.53794	60.52852
10	1.35796	1.42742	58.76186	0.41314	0.51099	59.98096
20	1.17288	0.38920	57.86322	0.41314	0.44357	59.10235
30	1.13627	0.38920	57.72981	0.41314	0.44357	58.25732

Tabela 6 Wpływ współczynnika krzyżowania na wynik dla operatora krzyżowania typu OX

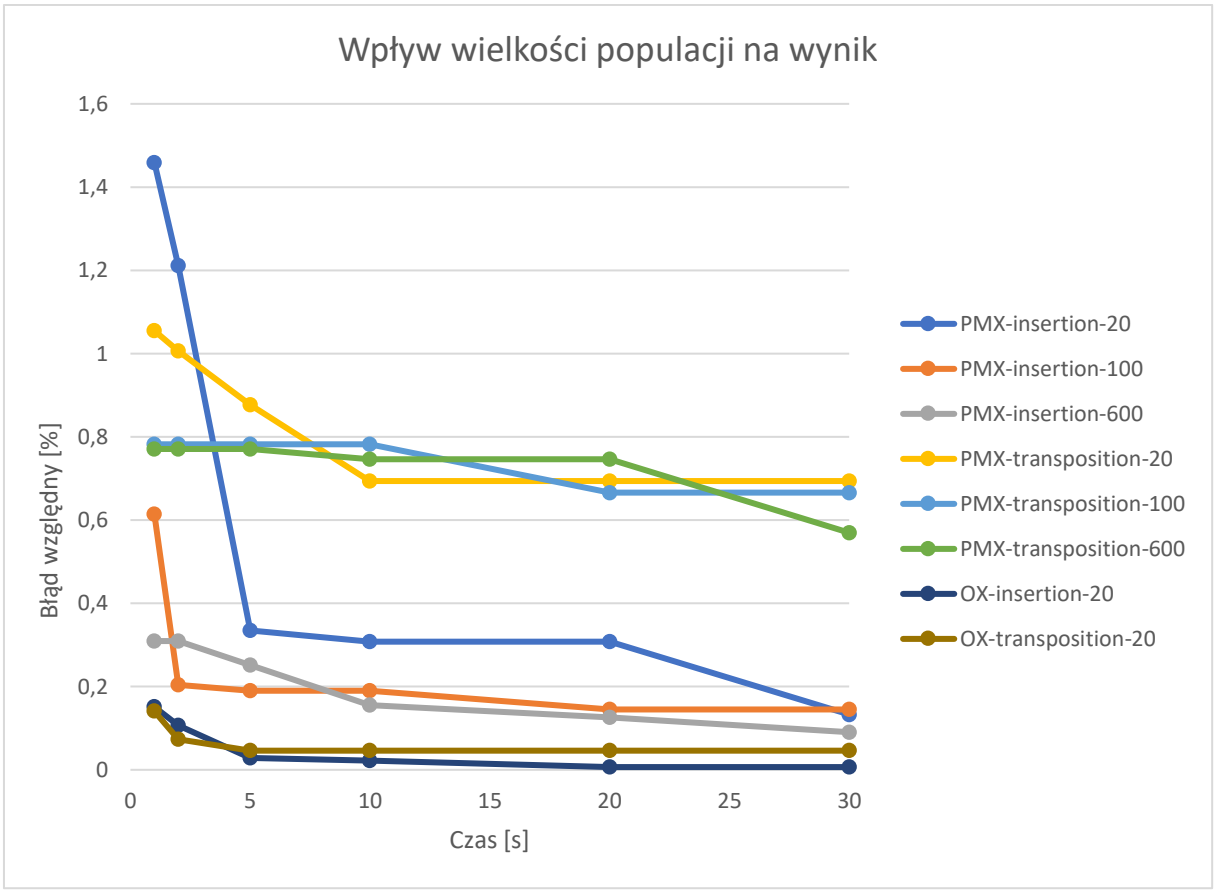
3.4 ANALIZA WYNIKÓW

3.4.1 Wpływ wielkości populacji na wynik



Wykres 1 Wpływ wielkości populacji na wynik

Na przedstawionym wykresie powyżej można zauważyć duży wpływ wielkości populacji na operator krzyżowania typu OX. Dla większych wielkości populacji dawał on o wiele gorsze wyniki. Aby umożliwić większą przejrzystość lepszych wyników, na kolejnym wykresie zostaną pominięte wyniki dla operatora krzyżowania OX z wielkościami populacji równymi 100 oraz 600.

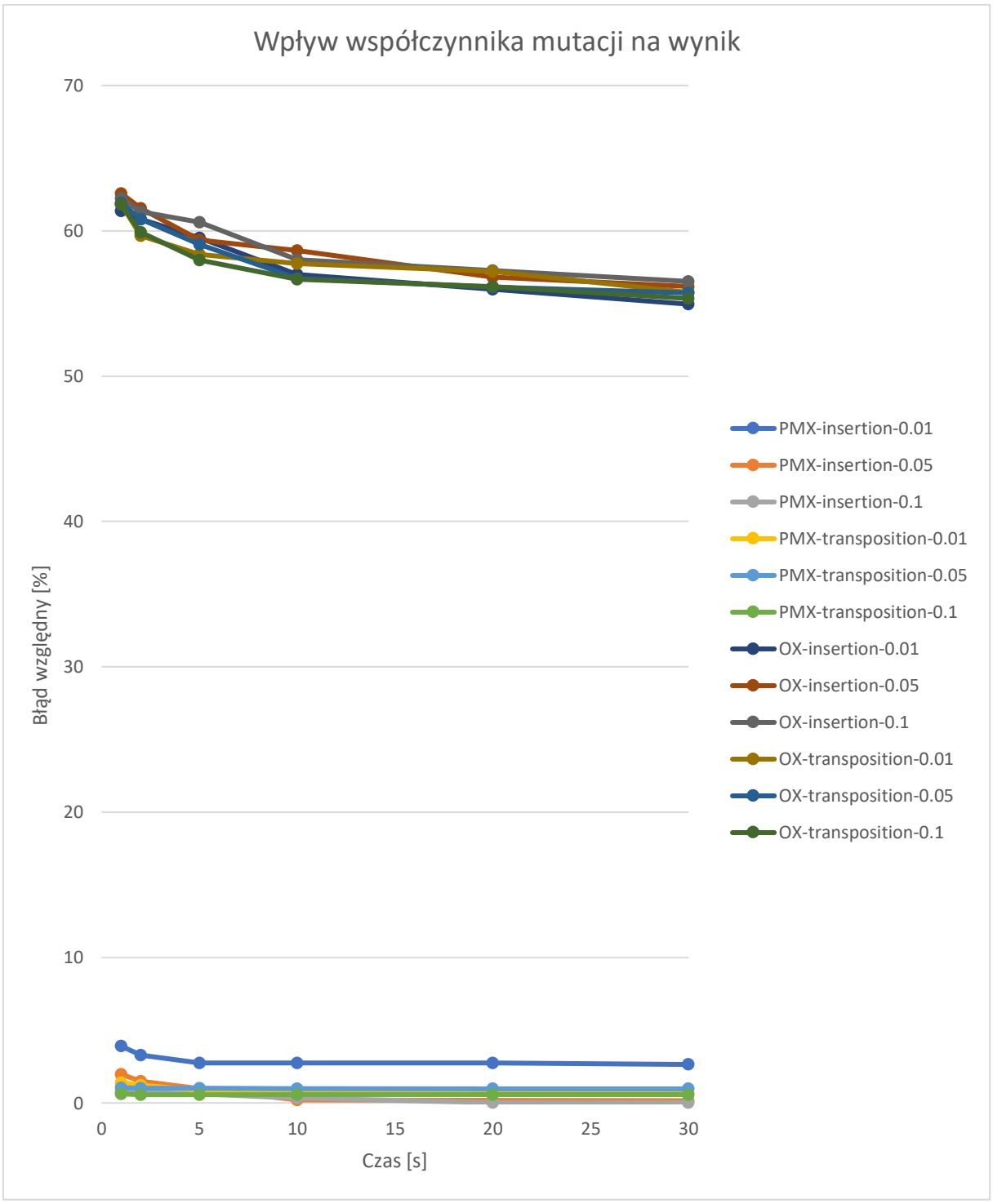


Wykres 2 Wpływ wielkości populacji na wynik - wykres zredukowany

Z powyższego wykresu nie da się jednoznacznie wywnioskować, która z wielkości populacji dawała lepsze wyniki dla operatora krzyżowania typu PMX. W tabeli nr 1 można zauważyć, że większa populacja przynosiła lepszy rezultat dla dłuższego działania algorytmu, jednak różnica ta jest na tyle mała, że mogła wyniknąć z wartości pseudolosowych losowych wygenerowanych przez program.

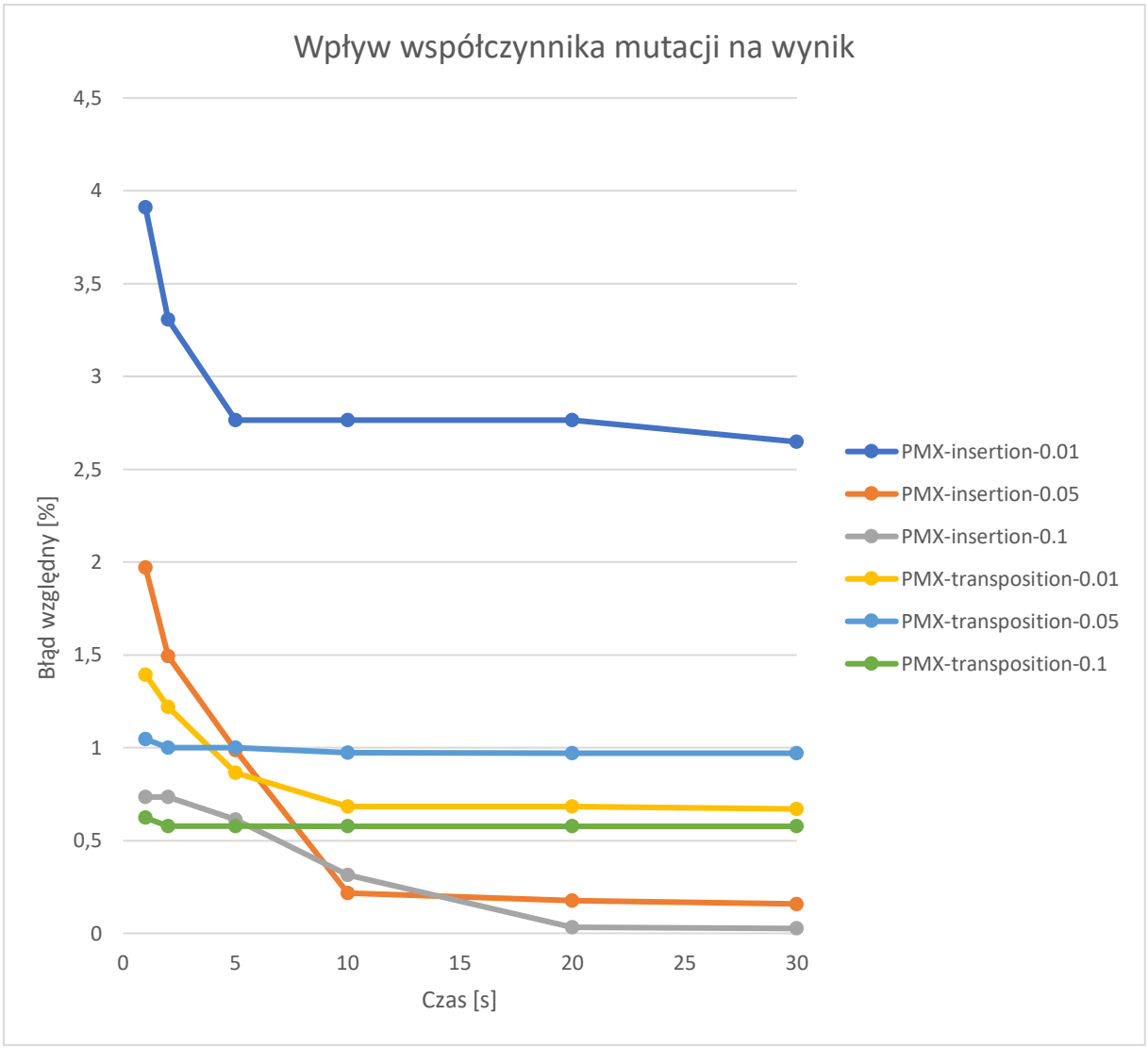
Do dalszych testów wybrana została populacja wielkości 100, która średnio w czasie dawała lepsze wyniki dla operatora PMX. Pozwoliło to również na pokazanie, jak duży wpływ dla operatora krzyżowania OX może mieć współczynnik krzyżowania. Najlepsza wielkość populacji dla operatora krzyżowania OX równa 20 została również pominięta z tego powodu, że tak dobre wyniki mimo małej populacji mogły zostać osiągnięte przez zły dobór typu selekcji.

3.4.2 Wpływ współczynnika mutacji na wynik



Wykres 3 Wpływ współczynnika mutacji na wynik

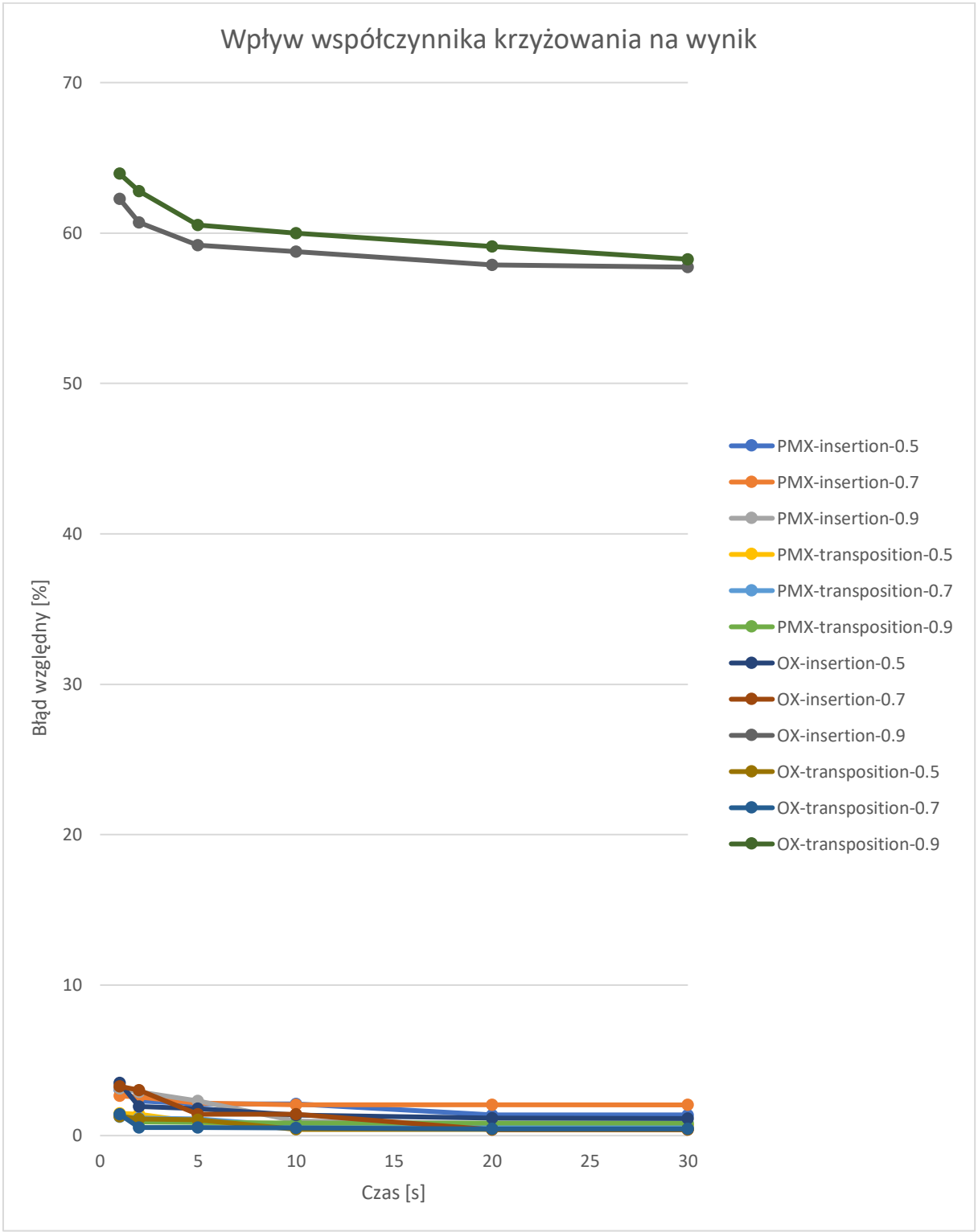
W tym wypadku wszystkie wyniki dla operacji krzyżowania typu OX są o wiele gorsze od krzyżowania typu PMX. Oznacza to, że współczynnik mutacji nie miał tak dużego wpływu na ten operator. Jednak współczynnik równy 0.1 dał końcowo minimalnie lepszy wynik. W kolejnej tabeli zostaną przedstawione tylko dane operatora PMX, ponieważ tylko on w tym wypadku dał bardzo dobre wyniki.



Wykres 4 Wpływ współczynnika mutacji na wynik – wykres zredukowany

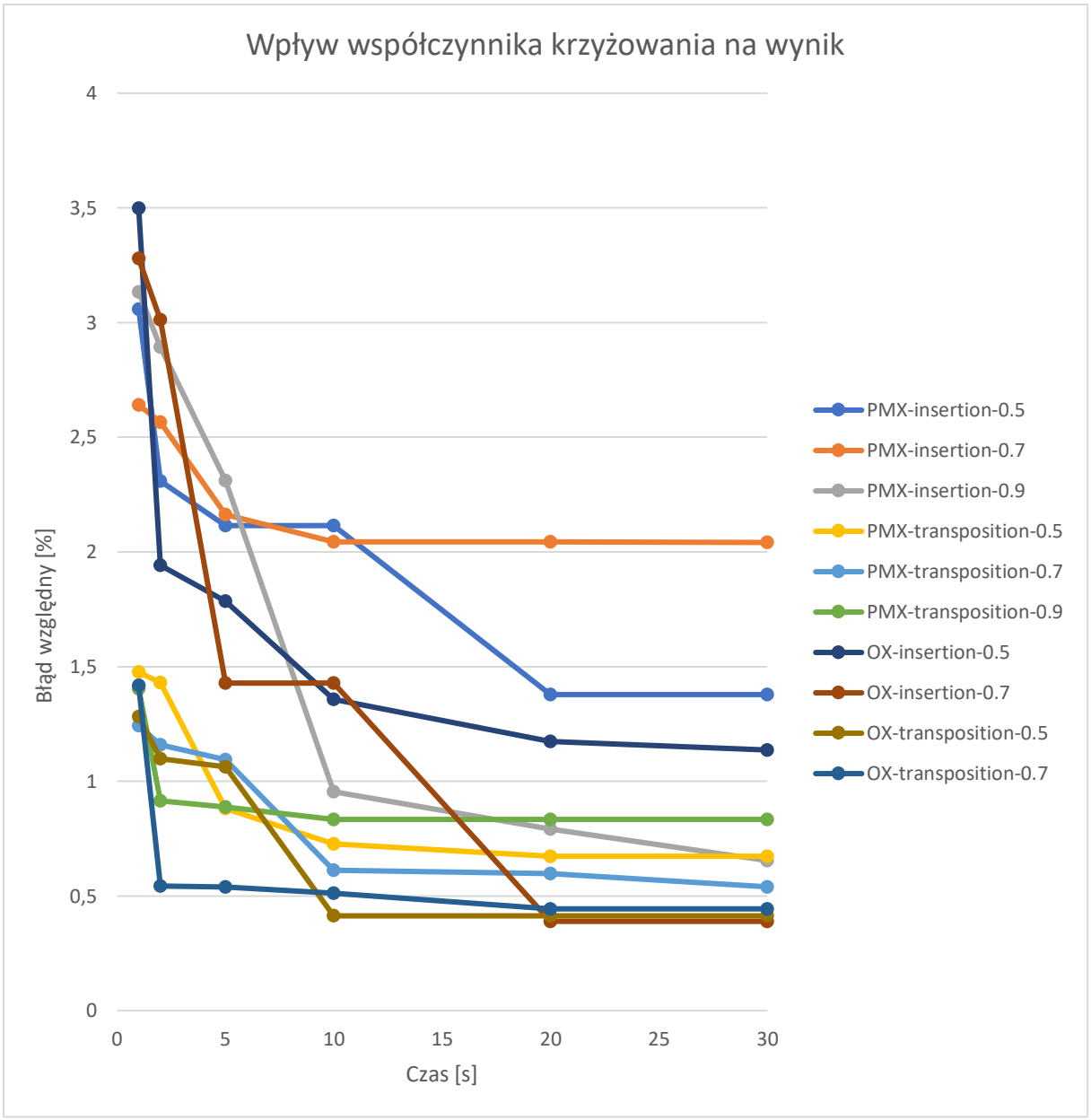
W tym wypadku widać duży wpływ współczynnika mutacji na mutacje typu insertion. Zwiększenie jego wartości dało najlepszy końcowo dostępny wynik. Dla mutacji typu transposition ten wpływ nie jest już tak widoczny. Współczynnik mutacji o wartości 0.1 oraz 0.01 są bardzo do siebie zbliżone.

3.4.3 Wpływ współczynnika krzyżowania na wynik



Wykres 5 Wpływ współczynnika krzyżowania na wynik

Na powyższym wykresie można zauważyć jak duży wpływ miało zmniejszenie współczynnika krzyżowania dla operatora krzyżowania typu OX. Zmniejszenie go spowodowało zrównanie się wyników do operatora typu PMX. Na poniższym wykresie, aby zwiększyć widoczność wartości, pominięte zostały wyniki dla operatora krzyżowania typu OX dla wartości współczynnika mutacji równego 0.9.



Wykres 6 Wpływ współczynnika krzyżowania na wynik - wykres zredukowany

W tym wypadku również widać, że zmniejszony współczynnik krzyżowania miał dobry wpływ na końcowy wynik dla operatora krzyżowania typu OX, szczególnie na jego wersję z mutafcą typu insertion. Dla operatora krzyżowania typu PMX oraz mutacji typu insertion, lepsze wyniki dawał zwiększony współczynnik krzyżowania. Dla operatora krzyżowania PMX oraz mutacji typu transposition dane te nie są już tak jednoznaczne.

4 WNIOSKI

Po ustawieniu odpowiednich parametrów zaimplementowany Algorytm Genetyczny dawał bardzo dobre wyniki końcowe. Szczególnym przypadkiem jest operator krzyżowania OX, który dla małej wielkości populacji, oraz niskiego współczynnika krzyżowania dawał średnie wyniki bardzo bliskie 0%. Zmniejszony współczynnik krzyżowania zwiększa wpływ selekcji turniejowej, która jest w stanie przez więcej iteracji zachować najlepsze dostępne permutacje. Może to również oznaczać, że selekcja turniejowa nie jest dobrym wyborem dla operatora OX i możliwe lepsze rozwiązania mogłyby pojawić się dla selekcji koła ruletki.

Zwiększanie współczynnika mutacji miało dobry wpływ na wyniki algorytmu. Pozwala on unikać wartości optymalnych tylko lokalnie. Należy jednak pamiętać, aby nie zwiększać go zbytnio, aby potencjalnie dobra populacja macierzysta nie została zbyt mocno zmodyfikowana.

Zwiększanie populacji początkowej ma dobry wpływ na algorytm, jeżeli zostanie on uruchomiony przez odpowiednio długi czas. Dla operatora krzyżowania typu PMX jest to widoczne w tabeli nr 2, jednak różnica ta może wynikać z losowości algorytmu. Mała populacja początkowa pozwala na szybsze osiąganie dobrych wyników, kosztem ilości informacji o przestrzeni problemu.

Podczas testów współczynnik mutacji miał minimalny wpływ na wyniki końcowe. Pozwala on jednak na zwiększenie dywersyfikacji algorytmu. Dane testowe były generowane w bardzo podobny sposób (przewidywane czasy zakończenia zadania były generowane z rozkładu jednostajnego). Większa różnorodność instancji problemów mogłaby pokazać zaletę doboru współczynnika mutacji oraz jej typu.