Grupa II (Inf): środa, tydzień parzysty, 9:45

Marcin Popławski
106313
marcin.poplawski@student.put.poznan.pl

Tomek Jósiak
121325
tomek.josiak@student.put.poznan.p

Problem szeregowania zadań: Algorytm genetyczny

1 Metoda rozwiązania problemu

Problem 3
Jobshop, liczba maszyn $m = 2$, liczba zadań n,
operacje niewznawialne,
dla drugiej maszyny k okresów konserwacji o losowym czasie rozpoczęcia i trwania (określonym przez generator instancji problemu), $k \geq \frac{n}{8}$,
czas gotowości dla każdej operacji nr 1 każdego zadania, nieprzekraczający $\frac{1}{4}$ sumy czasów wszystkich operacji.
minimalizacja całkowitego czasu wykonania wszystkich operacji

Na rozwiązanie problemu składają się 3 główne komponenty: generator instancji problemu, generator rozwiązania losowego i generator rozwiązania metaheurystyką.

1.1 Generator instancji problemu

Pierwszy z komponentów, mający na celu dostarczenie pozostałym instancji, która będzie podlegać rozwiązaniu, działa w dwóch trybach jeśli biorąc pod uwagę sposób przydzielania czasu poszczególnym operacjom, a także biorąc pod uwagę sposób generowania okresów konserwacji dla drugiej maszyny.

Jeden z trybów generowania zadań, polegający na deklaracji klas długości zadań, opiera się całkowicie na parametrze *ProblemInstanceGenerator.ranges* będącego dwuwymiarową macierzą liczb całkowitych. Wiersze macierzy symbolizują klasy długości zadań, mają rozmiar 3 i składają się z: liczby zadań w tej klasie, minimalnej długości zadania oraz maksymalnej długości zadania.

Aby dodatkowo urozmaicić rodzaje instancji generowane w ten sposób, liczba zadań w danej klasie jest podwajana, a klasy długości są niezależnie przypisywane każdej operacji. W ten sposób przekazując do generatora jako parametr macierz [[15, 1, 10], [10, 20, 40], [5, 80, 100]] możemy się spodziewać wygenerowania instancji problemu, w której 30 operacji będzie miało długość z przedziału <1, 10), 20 z przedziału <20, 40), a 10 z <80, 100). Ma to zapewnia to dużą elastyczność generatora jeśli chodzi o definiowanie klasy instancji problemu.

Drugi tryb generowania zadań zakłada podanie dwóch parametrów: liczby zadań n oraz całkowitego czasu przeznaczonego C na ich wykonanie. Generator najpierw losuje dla każdej operacji czas zbliżony do C/n. Jeśli pozostanie jeszcze nie wykorzystany czas, to generator będzie losowym operacjom przypisywał po jednej jednostce czasu aż do jego wyczerpania.

Podobnie jest w przypadku generowania okresów konserwacji. Jeden tryb zakłada podanie liczy okresów i ich całkowitego czasu trwania, a drugi liczby okresów i maksymalnej

długości okresu. Generator zapewnia, żeby okresy te na siebie nie nachodziły, więc należy zwrócić uwagę, żeby ilości i czasy nie były zbyt długie, gdyż może to uniemożliwić generatorowi wykonanie zadania.

1.2 Generator rozwiązania losowego

Generator rozwiązania losowego ma na celu szybkie dostarczenie rozwiązania nawet jeżeli miałoby to oznaczać cechowania się dużym błędem. Ustawia on w losowej kolejności operacji na pierwszej maszynie i równolegle ustawia operacje na obu maszynach, tak aby zapewnić poprawność generowanego rozwiązania. Na początku napisaliśmy i rozważaliśmy użycie innego generatora rozwiązania losowego, który iterujac po kolejnych jednostkach czasu szukał potencjalnych operacji i układał losową z nich w danym segmencie czasu. Zrezygnowaliśmy z tego rozwiązania, gdyż pomimo generowania rozwiązań losowych o niższej wartości funkcji celu, stosowanie tego generatora prowadziło do pojawiania się gorszych wyników po takim samych czasie optymalizacji.

1.3 Generator rozwiązania metaheurystyka

Jako algorytm metaheurystyczny, który wykorzystaliśmy do rozwiązania problemu wybraliśmy algorytm genetyczny. Poniżej przedstawimy działanie 3 operatorów składających się na ten algorytm czyli: mutacji, krzyżowania i selekcji.

W każdej ewolucji populacji losowe pary rozwiązań w populacji są ze sobą krzyżowane. Powstają z tego po dwa kolejne rozwiązania. Następnie algorytm wykonuje mutacje zgodnie z zadaną szansą (prawdopodobieństwem) zajścia. Tym sposobem liczba rozwiązań w populacji znacznie się zwiększa i należy ją poddać selekcji by sprowadzić populacje to pierwotnego rozmiaru przed rozpoczęciem kolejnej ewolucji.

W naszej implementacji mutacja odbywa się na jednej maszynie poprzez wybranie losowej operacji i przesunięcie jej w przód lub w tył. Odległość o jaką operacja może być przesunięta nazywamy siłą mutacji. Po przesunięciu operacji odbudowywana jest druga maszyna tak by rozwiązanie spełniało założenia problemu.

Selekcja polega na wyborze populacji o wielkości początkowej z większej populacji powstałej w wyniku krzyżowania i mutacji. Najpierw do nowej populacji przechodzą dwa uszeregowania o najlepszym wyniku. Wybór pozostałych rozwiązań przeprowadzony jest na drodze turnieju. Zależnie od liczby turniejów, do każdego z nich wybierana jest losowo odpowiednia liczba rozwiązań, z których wybierany jest ten lub te z najlepszym wynikiem.

Warunkiem stopu działania metaheurystyki jest upłynięcie zadanego czasu pracy nad pojedynczą instancją.

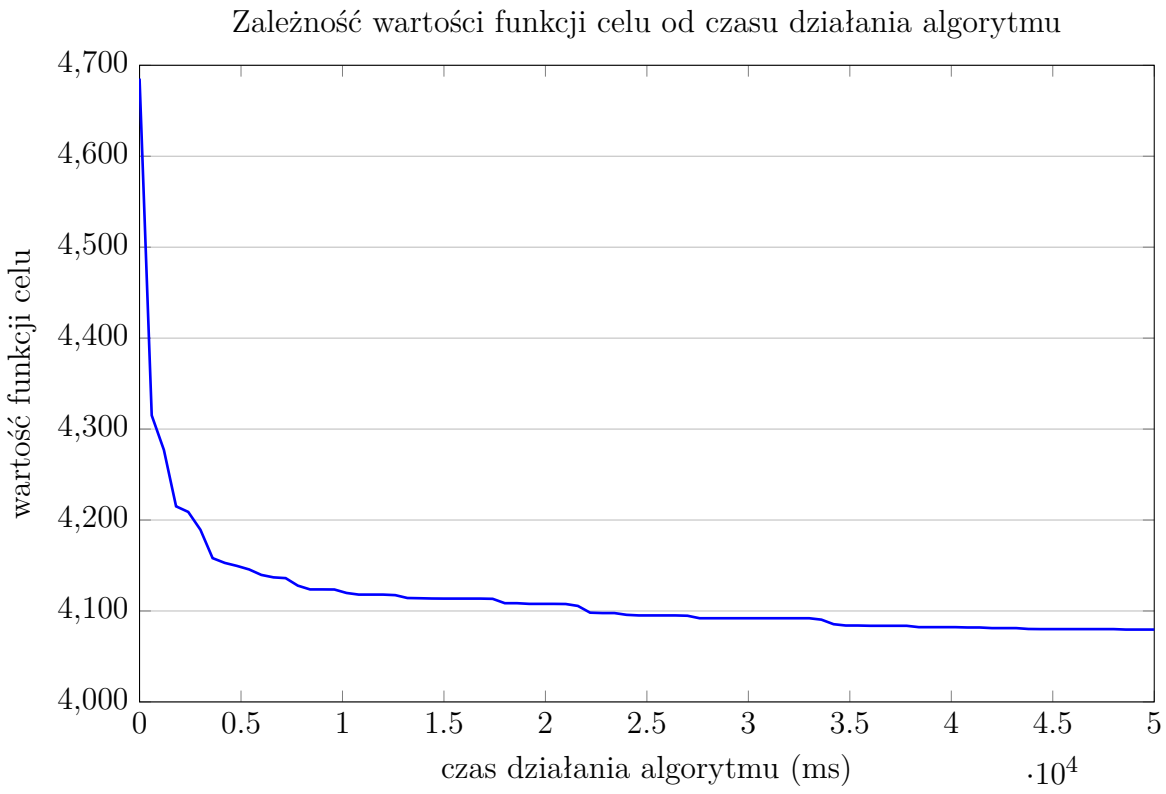
2 Testy parametrów algorytmu

Parametry domyślne użyte podczas testowania algorytmu:

- liczba zadań, minimalny i maksymalny czas trwania:
  [[20, 5, 20], [15, 40, 80], [15, 80, 120]]
- czas gotowości operacji nr 1 każdego zadania: $\frac{1}{20}$ sumy czasów wszystkich operacji
- wielkość populacji: 100
- czas pracy nad 1 instancją: 10 ms
- liczba punktów krzyżowania: 1
- prawdopodobieństwo mutacji: 0,2
- siła mutacji: 2
- liczba turniejów: 10

2.1 Test czasu działania algorytmu

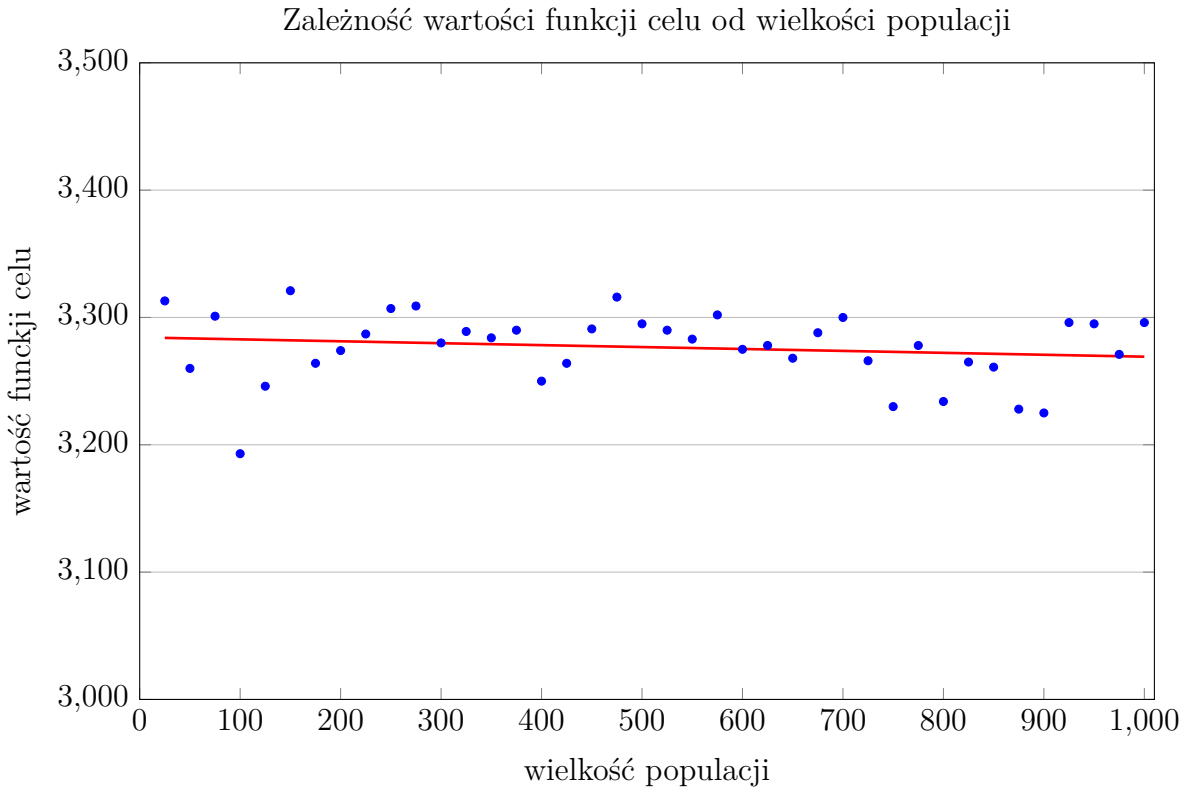
Test czasu działania algorytmu został przeprowadzony dla 100 zadań o sumie czasów wykonania równej 7000.



Wartość funkcji celu maleje gwałtowanie w ciągu pierwszej milisekundy działania algorytmu, gdyż znalezienie lepszego rozwiązania niż te powstałe w sposób losowy pośród powstałych z krzyżowania i mutacji jest stosunkowo łatwe. Dla wyższych wartości czasu widać, że wartości o jakie funkcja celu spada są niewielkie. Oznacza to, że pomimo iż dłuższy czas trwania algorytmu przekłada się na jakość powstałego rozwiązania to przeznaczanie dużej ilości czasu na prace algorytmu nie przynosi wartościowego rezultatu.

2.2 Zmiana wielkości populacji

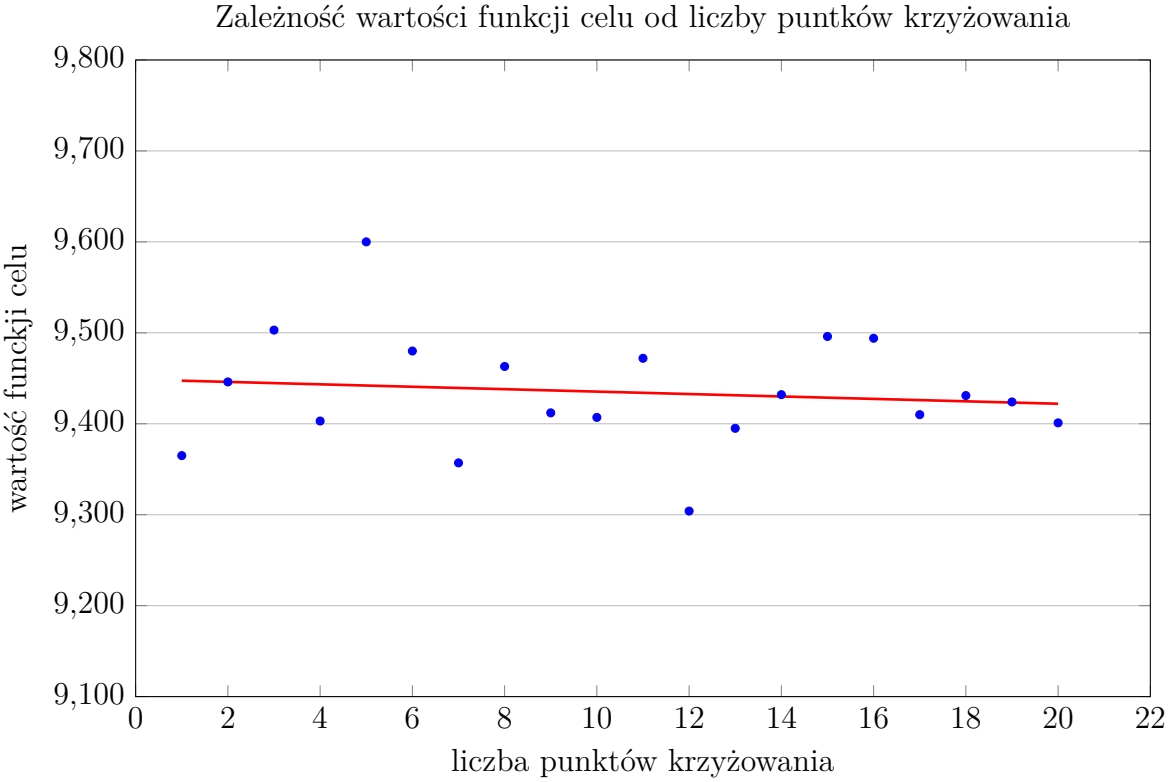
Parametr określający wielkość populacji mierzony był co 25 w zakresie $25 - 1000$.



Widoczna lekka poprawa wartości funkcji celu w raz ze wzrostem wielkości populacji związana jest z tym, iż w większej populacji istnieje większa szansa, że powstanie jak najlepsze rozwiązanie. Jednak wtedy algorytm musi przeprowadzić więcej operacji. Dlatego poprawa rozwiązania nie jest znacząca, co widać na powyższym wykresie. Można przyjąć, że wielkość populacji nie ma znaczącego wpływu na jakość otrzymanego wyniku działania algorytmu.

2.3 Zmiana liczby punktów krzyżowania

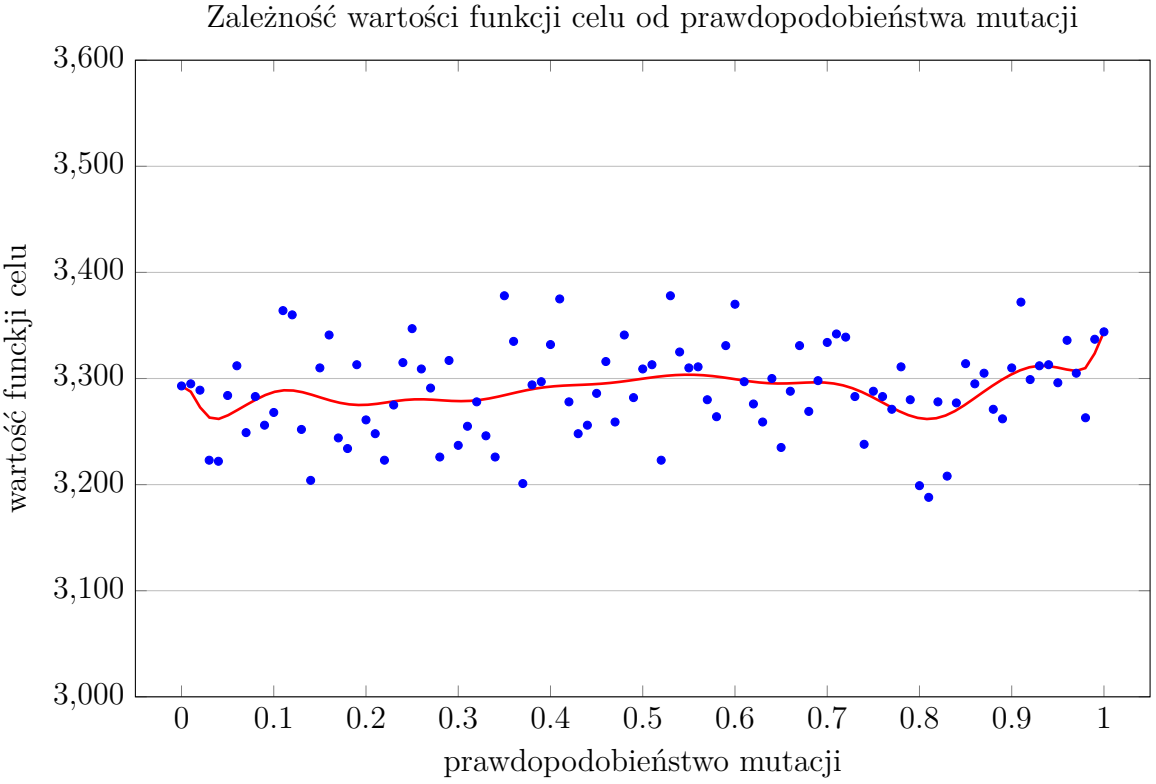
W teście krzyżowania macierz zadań prezentowała się następująco: [[20, 10, 30], [20, 40, 70], [30, 90, 120], [30, 110, 140]]. Badany parametr zmieniał się co 1 w zakresie $1 - 20$.



Powyższy wykres nie wskazuje na wyraźną korelację między liczbą punktów krzyżowania a wartością funkcji celu. Można przybliżyć, iż zależność jest stała. Pomimo iż różnica w wartościach funkcji celu między maksimum, a minimum wydaje się być znaczna, jest ona równa jedynie ok 3% wartości funkcji celu, a ich występowanie, odpowiednio dla liczby punktów krzyżowania 5 i 12, nie wydaje się mieć poparcia w sposobie wykonania krzyżowania. Można więc założyć, że są one wynikiem jedynie przypadku, a nie tendencją wartą zanotowania.

2.4 Zmiana prawdopodobieństwa mutacji

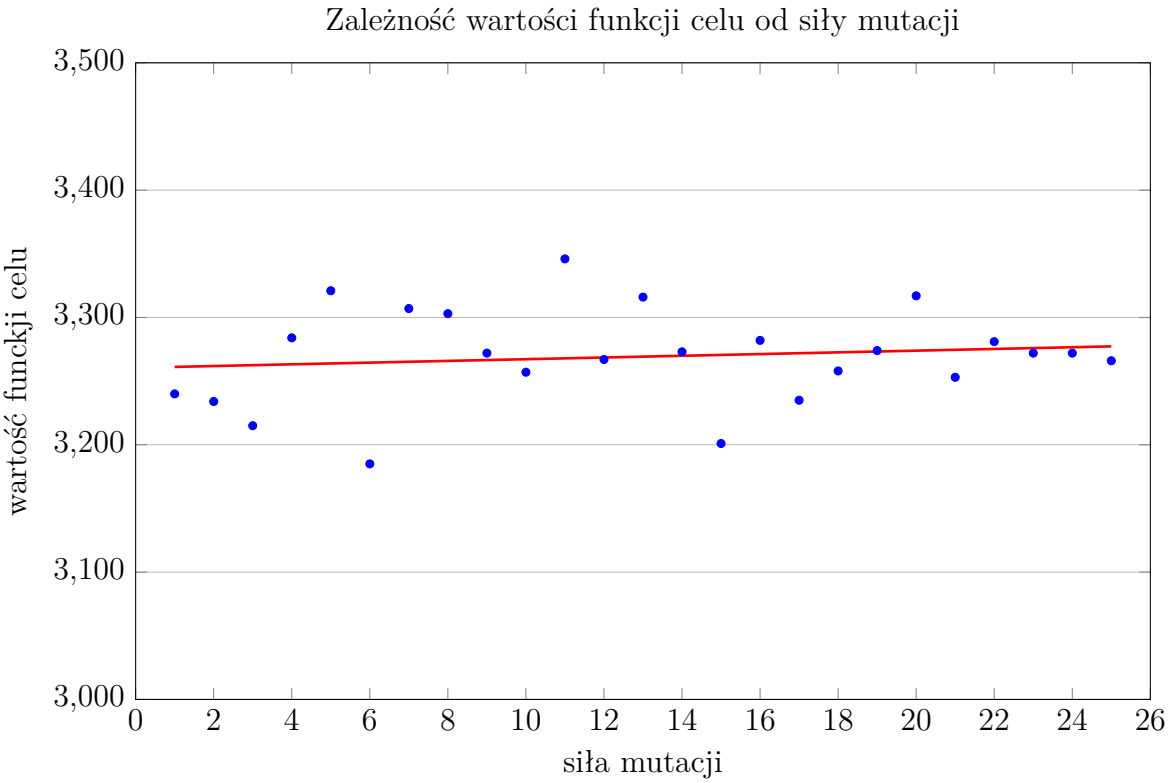
Pomiar zależności funkcji celu od prawdopodobieństwa mutacji został przeprowadzony ze zmianą badanego parametru w zakresie od 0,00 do 1,00 z krokiem co 0,01.



Z powyższego wykresu wynika, że zależność wartości funkcji celu od prawdopodobieństwa mutacji jest w przybliżeniu stała. Warto jednak zauważyć delikatne odchylenie dla prawdopodobieństwa mutacji pomiędzy 0 a 0,1. Ze znalezionych w internecie publikacji wyczytaliśmy, że optymalna wartość prawdopodobieństwa mutacji wynosi między 0.01 a 0,1, co przedstawiony wykres potwierdza. Warte wspomnienia jest również drugie odchylenie wykresu przy wartości prawdopodobieństwa 0,8, w którym także występuje zauważalna poprawa wyników.

2.5 Zmiana siły mutacji

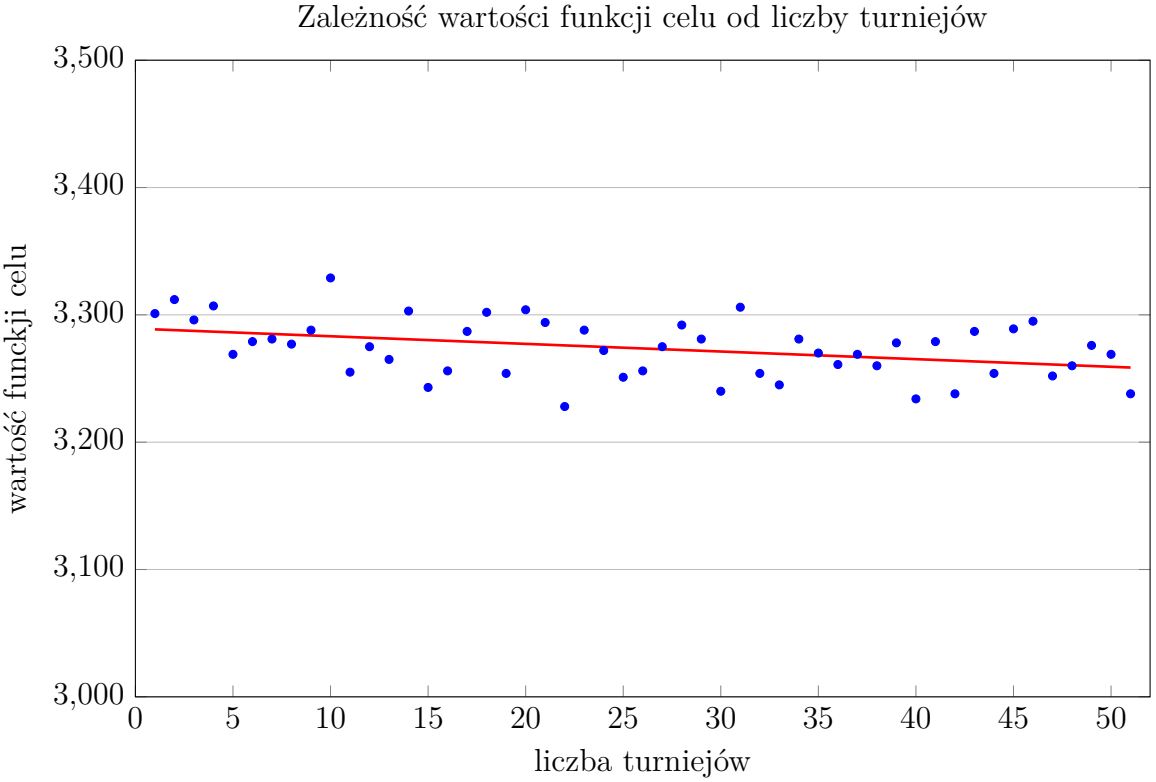
Wartość siły mutacji mierzona była w zakresie od 1 do 25 z krokiem co 1.



Podobnie jak w przypadku zmiany liczby punktów krzyżowania można przyjąć, iż siła mutacji nie ma znacznego wpływu na wartość funkcji celu. Lekki wzrost jej wartości jest prawdopodobnie spowodowany tym, że im większa siła mutacji tym algorytm więcej czasu spędza na naprawie uszeregowania. Ma wtedy mniej czasu na inne operacje i ostatecznie przekazuje trochę gorsze rozwiązanie niż dla mniejszych wartości siły mutacji.

2.6 Zmiana liczby turniejów

Badanie zależności wartości funkcji celu od liczby turniejów zostało przeprowadzone ze skokiem wartości parametru co 1 w zakresie $1 - 50$.



Im większa wartość parametru, tym rozwiązanie jest w niewielkim stopniu lepsze. Wynika to z tego, że dla mniejszych turniejów (więcej turniejów oznacza też, że są mniejsze) jest mniejsze prawdopodobieństwo, że jakieś dobre rozwiązanie nie zostanie wybrane na drodze turnieju do kolejnej ewolucji.

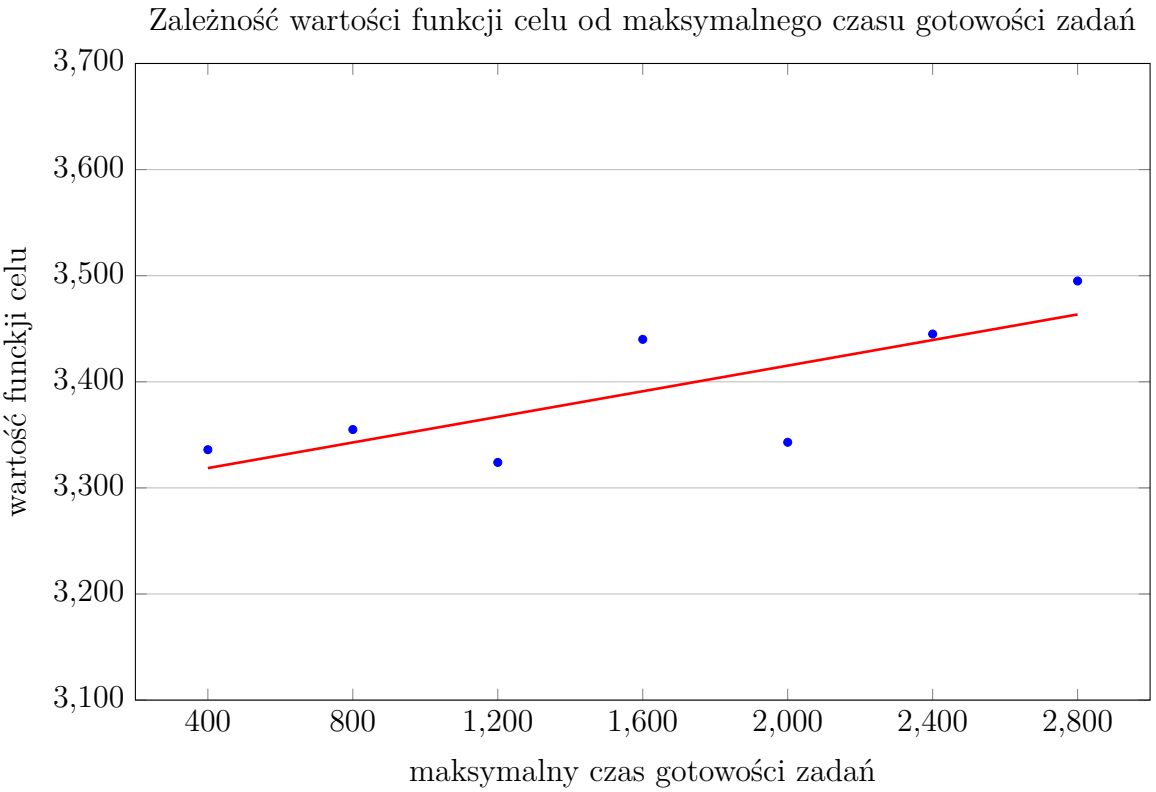
3 Testy parametrów instancji

Parametry domyślne użyte podczas testowania instancji:

- liczba zadań, minimalny i maksymalny czas trwania:
  [[20, 5, 20], [15, 40, 80], [15, 80, 120]]
- czas gotowości operacji nr 1 każdego zadania: $\frac{1}{20}$ sumy czasów wszystkich operacji
- wielkość populacji: 100
- prawdopodobieństwo mutacji: 0,1
- czas pracy nad 1 instancją: 10 ms
- siła mutacji: 2
- liczba punktów krzyżowania: 1
- liczba turniejów: 10

3.1 Zmiana maksymalnego czasu gotowości zadań

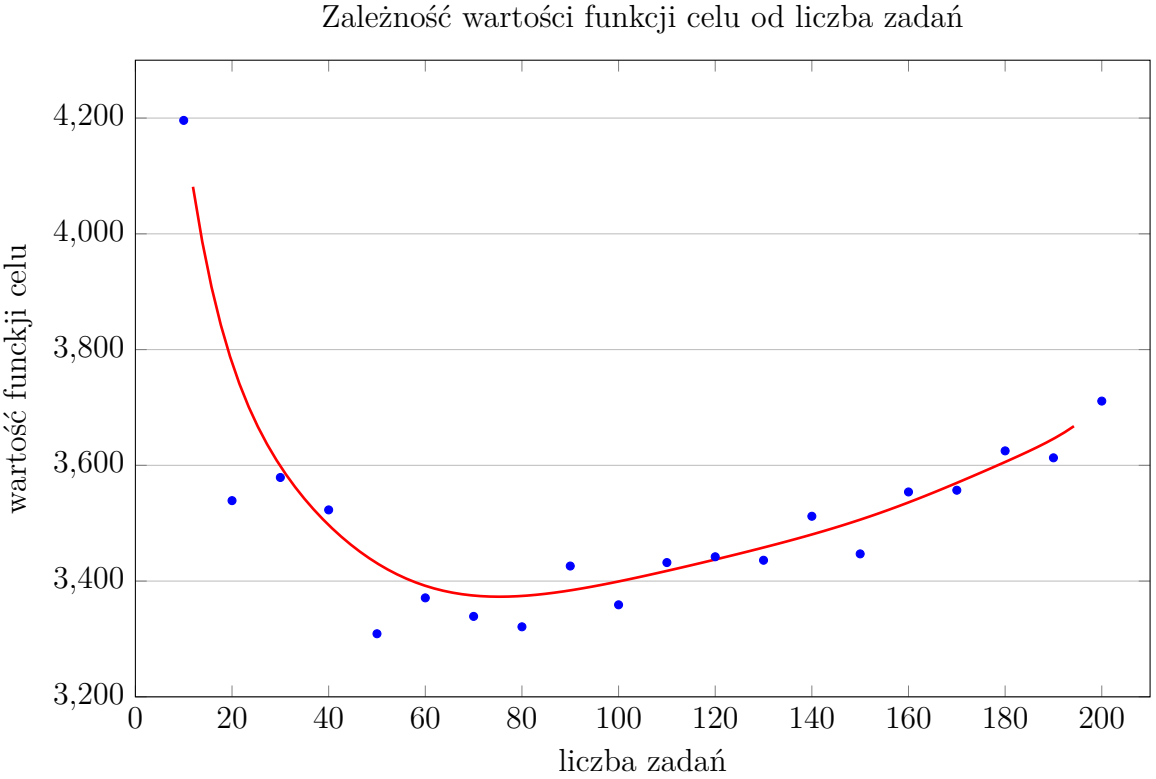
Ostatni przedstawiony test polega na zmianie maksymalnego czasu gotowości zadań. Wartości parametru dla pojedynczego zadania losowane są z zakresu od zera do zadanej wartości.



Z powyższego wykresu wynika prosty wniosek, że wraz ze wzrostem maksymalnego czasu gotowości zadań liniowo rośnie wartość funkcji celu.

3.2 Zmiana liczby zadań przy stałej sumie czasów trwania

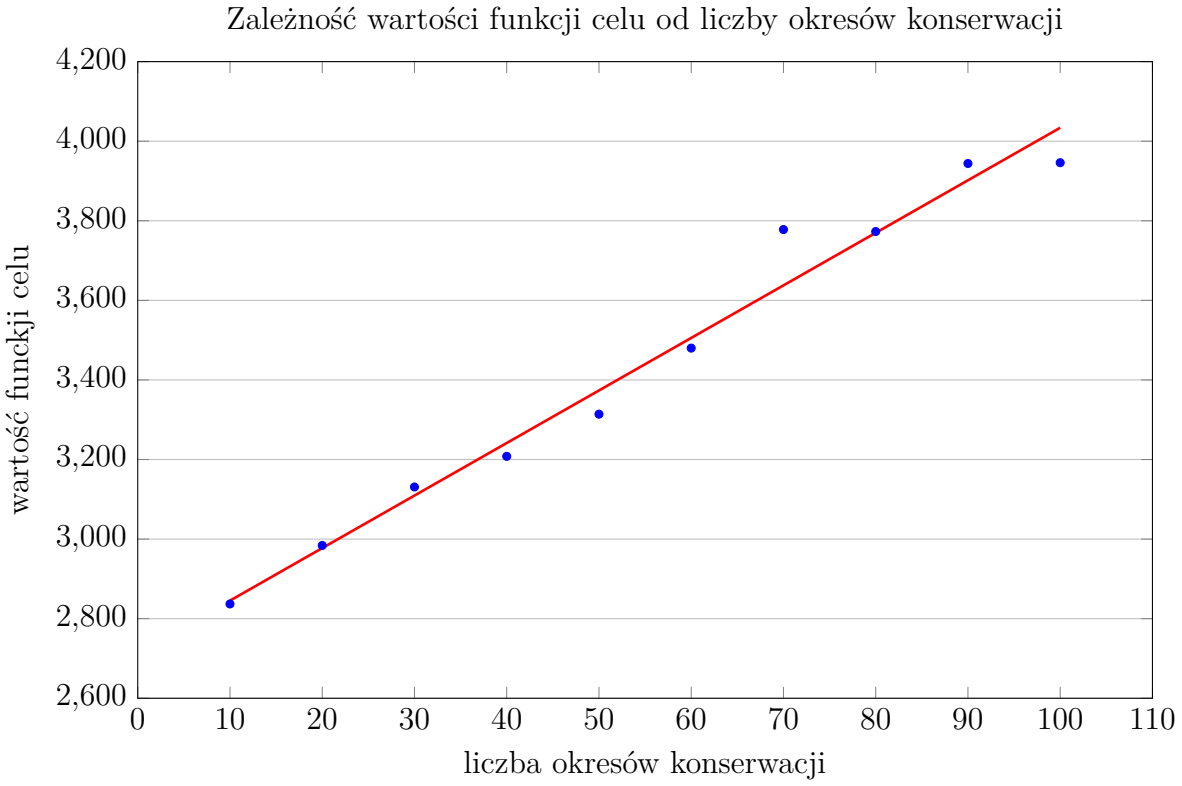
Test badający zależność wartości funkcji celu od liczby zadań algorytmu został przeprowadzony dla stałej sumy czasów trwania wszystkich operacji równej 6000.



Dla mniejszej ilości zadań algorytm może działać szybciej przez mniej wymaganą ilość operacji, ale wynik jest gorszy przez długie operacje (trudniej jest dopasować w dostępne miejsca między innymi operacjami lub okresami konserwacji). Przeciwnym przypadkiem jest duża liczba zadań z krótkimi operacjami. Algorytm ma więcej operacji do wykonania i dlatego od pewnego momentu na wykresie zależność jest rosnąca. Optimum znajduje się w granicach $70 - 90$ zadań.

3.3 Zmiana liczby okresów konserwacji przy stałej sumie czasów trwania

Poniższy wykres prezentuje zmianę liczby okresów konserwacji przy stałej sumie czasów trwania w zakresie od 10 do 100 z krokiem co 10.



Algorytm zwraca znacznie lepszy wynik w przypadku małej liczby długich okresów konserwacji w porównaniu. Dużo krótkich okresów konserwacji w instancji może być tworzyć obszar, o znacznie dłuższym okresie trwania, gdzie nie zmieści się żadna operacja lub niewielka ich liczba.