|  |  |  |  |
| --- | --- | --- | --- |
| Imię | Nazwisko | Nr Indeksu | E-mail |
| Krzesimir | Samborski | 109795 | krzesimir.samborski@student.put.poznan.pl |
| Maciej | Kozłowski | 109782 | maciej.r.kozlowski@student.put.poznan.pl |

Job shop z wywłaszczaniem na algorytmach - losowym i SJF

# Opis rozwiązania

## Generator

Rozwiązanie zadania uszeregowania typu job shop rozpoczynamy od skonstruowania generatora tworzącego odpowiednie dane, pozwalające spełnić warunki polecenia. W tym celu zaczęliśmy od napisania klasy Generator, zawierającej liczbę zadań (wielkość instancji problemu), wektor z maszynami, zadaniami oraz szacunkową długość rozwiązania optymalnego. Ta ilość danych daje nam liniową złożoność pamięciową (dodatkowo zależną od ilości przestojów) i jest w zupełności wystarczająca do podstawowej funkcjonalności i nie tylko.

Przeanalizujmy teraz jednak specyfikację i złożoność metod zawartych w klasie:

- metoda generujZadanie, o złożoności wywoływana razy daje nam czas na wygenerowanie wszystkich potrzebnych zadań wraz z ich operacjami.

- metoda generujMaszyny, o złożoności liniowej, niezależnej od ilości zadań, tylko od liczby przestojów. Tworzy wektor maszyn wraz z przestojami. Przestoje generowane są z uwzględnieniem szacowanej długości instancji oraz dodatkowo tak, aby się wzajemnie nie nakładały.

- metoda sprawdzająca, czy dana operacja nie koliduje z przestojem (złożoność liniowa względem ilości przestojów)

- metoda obliczająca długość optymalnego uszeregowania ()

## Pozostałe klasy

Oczywiście korzystanie wyłącznie ze zmiennych całkowitych w wektorach byłoby wielce niedogodne, gdyż należałoby wykorzystać dużo czasu procesora na zdobywanie dodatkowych danych o procesach i maszynach. Dlatego w celu rozwiązania tego problemu zostały napisane dwie dodatkowe klasy: Operacja oraz Maszyna.

Te typy obiektów nie zawierają żadnych metod, jedynie pola, ułatwiające dostęp do powiązanych z nimi informacji.

Klasa Maszyna składa się z numeru maszyny, liczby przestojów, które na niej się znajdują, wektora czasów rozpoczęć tych przestojów, ich długości oraz wektor uszeregowania operacji.

### Klasa Operacja zawiera czas trwania operacji i jej rozpoczęcia w danym uszeregowaniu, numer jej i Zadania, do którego należy, flagę ukończenia operacji oraz wskaźnik na Zadanie i Maszynę, na której się znajduje.

## Algorytm losowy

Podejście do algorytmu losowego, jako najbardziej swobodnego w implementacji dawało największe pole manewru w wyszukiwaniu rozwiązania.

Korzystamy z informacji o instancji problemu pobranych z Generatora oraz z dwóch trzyelementowych tablic - zawierającej obecny czas i zawierającej numery pierwszego niewykorzystanego przestoju na każdej z maszyn.

Zadaniem algorytmu jest zapełnienie wektora uszeregowania znajdującego się na poszczególnych maszynach. Zatem pamięciowo zajmujemy liniową ilość przestrzeni. Przebieg algorytmu rysuje się następująco:

W głównej pętli wykonujemy instrukcje aż nie zostaną wykorzystane wszystkie zadania z tablicy. Najpierw wybieramy najmniej zajętą maszynę i następnie próbujemy umieścić na niej kolejną operację zadania z wektora zadań w czasie bieżącym, określonym osobno dla każdej z maszyn (jest to druga pętla algorytmu). Jeżeli nie jest to możliwe to przeglądamy wektor pozostałych jeszcze zadań tak długo, aż w końcu uda się coś umieścić (i wtedy szukamy ponownie najmniej zajętej maszyny) bądź też aż nie osiągniemy końca wektora. Jeżeli nastąpi druga ewentualność to umieszczamy w bieżącej maszynie operację bezczynności zajmującą okres o długości 1 (lub jeśli poprzednia operacja też jest “zapychaczem” to zwiększamy czas jej trwania o 1), a następnie losujemy inną maszynę. W ten sposób rozwiązujemy potencjalną sytuację, w której nie można umieścić, bez żadnego oczekiwania, operacji na żadnej z maszyn.

Gdy uda się umieścić ostatnią operację danego zadania, usuwamy je, przybliżając się do opróżnienia wektora zadań i powracamy do początku głównej pętli (czyli wyboru najmniej zajętej maszyny).

Jeżeli obecny czas dla danej maszyny koliduje z przestojem, dodajemy do tego czasu długość trwania przestoju oraz zwiększamy okres poprzedniej operacji (tej, która nachodzi na przestój) o karę (30%). Widzimy zatem, że przestoje nie są przedstawione jawnie w wektorze uszeregowania, lecz raczej programowo uwzględniane jest ich istnienie. Złożonościowo analizując algorytm mamy dwie pętle (główną oraz umieszczającą operacje na maszynach) wykonujące się w pesymistycznym przypadku odpowiednio oraz razy (jeżeli dwukrotnie nie udało się trafić na odpowiednią maszynę z każdą operacją). Daje nam to efektywność obliczeniową rzędu

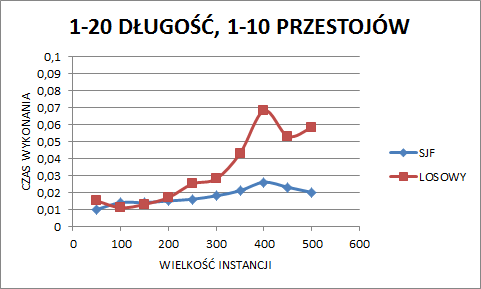
## Algorytm Shortest Job First

Do implementacji algorytmu SJF (najkrótsze zadanie jako pierwsze) oczywiście wykorzystujemy dane pobrane Generatora oraz podobnie jak w algorytmie losowym – z trzyelementowej tablicy, w której dla każdej maszyny osobno przechowujemy czas zakończenia ostatniej operacji. Również w algorytmie SJF zapełniamy operacjami wektor uszeregowania dla każdej z maszyn oraz liniowo przestrzeń pamięci. Kolejne kroki algorytmu:

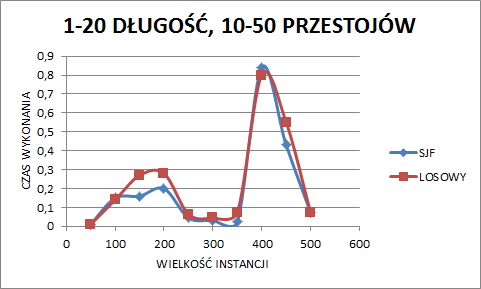
Na samym początku sortujemy wszystkie zadania rosnąco według zsumowanej długości trwania wszystkich 3 operacji. Następnie będziemy wykonywali pętle “while”, tak długo, aż nie usuniemy wszystkich zadań, a usuwamy je w momencie, gdy wykonają się wszystkie 3 operacje. Kolejnym krokiem jest wyszukanie najmniej zapełnionej maszyny, poprzez sprawdzenie aktualnych ich czasów. Rozpoczynamy od zadania numer 0 i wchodzimy do kolejnej pętli, w której po kolei sprawdzamy dla danego zadania, która operacja nie została wykonana. Dla pierwszej operacji sprawdzamy czy jest już gotowa, a dla kolejnych czy poprzednia/poprzednie się skończyły oraz czy wykonywały się na tej samej maszynie. W takim przypadku wybieramy niewykorzystaną maszynę. Kolejnym warunkiem jaki sprawdzamy to czy w danym momencie zaczyna się przestój, jeśli tak to go dodajemy do uszeregowania, sprawdzamy również czy podczas wykonywania danej operacji napotkamy przestój, w takiej sytuacji dodajemy do uszeregowania operację, następnie dodajemy przestój i dodajmy dodatkowe 30% trwania operacji. Oczywiście w każdym przypadku dodajemy do kwantu czasu maszyny odpowiednią liczbę, dla danej operacji przypisujemy daną maszynę, a także polu begin - wbrew nazwie - czas z zakończenia operacji i zmieniamy flagę wykonania operacji na “true”. Jeśli dla danego zadania nie ma operacji do wykonania w danym momencie, to zwiększamy według posortowania nr zadania. W przypadku kiedy sprawdziliśmy wszystkie zadania, wracamy do zerowego, sprawdzamy czy jest przestój, jeśli nie to dodajemy na danej maszynie operację zapychacz, o czasie trwania 1 i znów wybieramy najmniej wykorzystaną maszynę. Na końcu sprawdzamy czy nie doszliśmy do jakiegoś przestoju i usuwamy go z idealnego rozwiązania.

# Wyniki testów

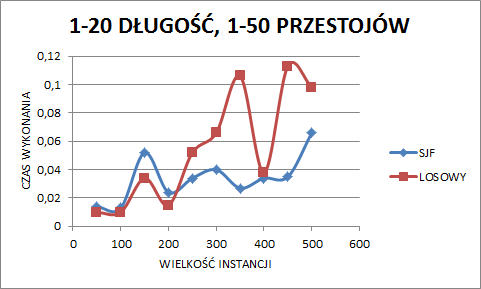
Serię wykresów rozpoczynamy od analizy i porównania czasu potrzebnego do wykonania poszczególnych algorytmów. Podane zostały w tytułach wykresu właściwości serii danych - długość operacji i ilość przestojów.



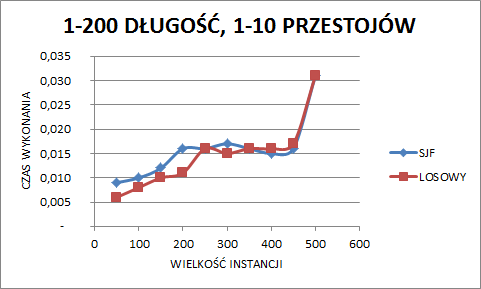
Dla niewielkiej instancji czasy są bliskie zeru



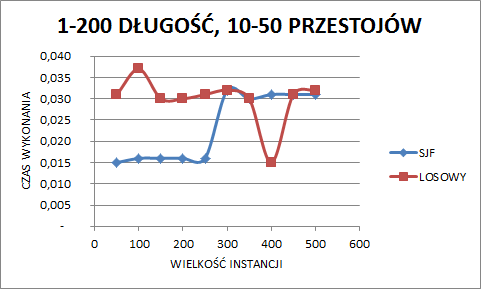
Jak widać często czas wykonania zależy od używanych danych wylosowanych przez generator



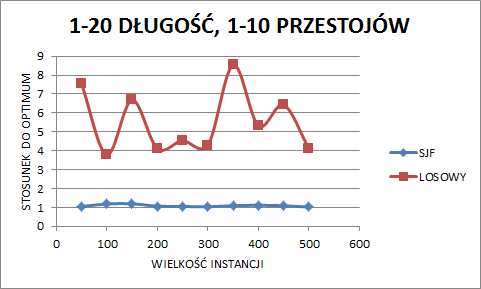
Średnio czasy uzyskiwane przez algorytm losowy były odrobinę gorsze



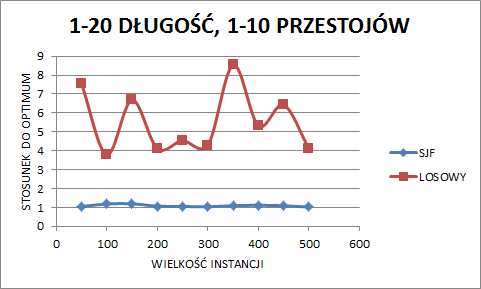
Dla niewielkiej ilości przestojów czasy są znacząco mniejsze (ponad dwukrotnie)



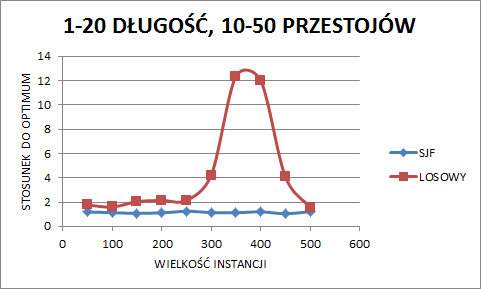
Następnie przejdźmy do omówienia długości generowanych uszeregowań (wyrażonych w stosunku do optimum):



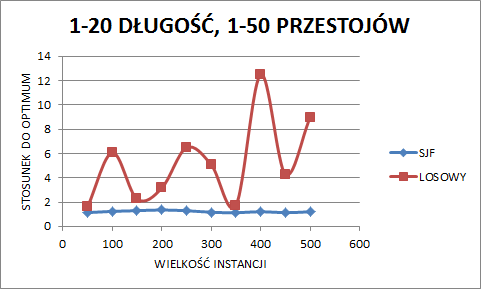
Algorytm losowy wykazuje duże rozproszenie wyników

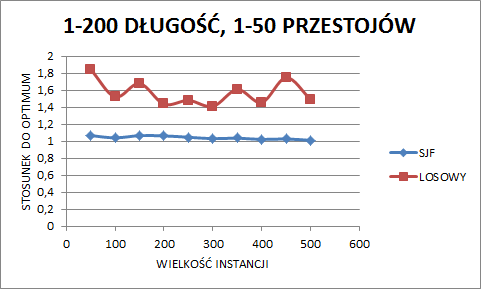


W przeciwieństwie do SJF, który oscyluje w okolicach 120% optimum

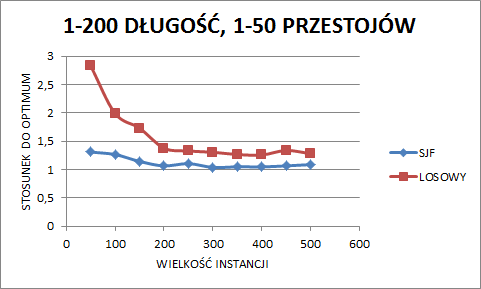


W niemalże każdym przypadku SJF okazuje się lepszy lub dużo lepszy od algorytmu losowego

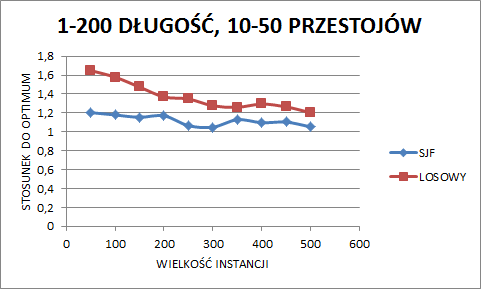




Dłuższe operacje przynoszą stabilizację wyników osiąganych przez algorytm losowy



Większa ilość przestojów też wpływa na wariancję



Długie operacje i dużo przestojów gwarantuje ustandaryzowane wyniki

# Wnioski

Pomimo że czasy wykonywania algorytmów są zbliżone do siebie z raczej niewielkimi odchyleniami to generowane przez nie uszeregowania znacząco się od siebie różnią. Im mniejsza ilość zadań/przestojów tym bardziej uwidacznia się losowa natura pierwszego algorytmu. Wraz z wzrostem skomplikowania problemu różnice niwelują się osiągając dla większych instancji podobne rezultaty.