PSZT

Projekt 1

SK 1.3 Walizka

Michał Sobieraj TU WSTAWIĆ INDEKS i Michał Łątkowski 300208

Imiona i nazwiska autorów.

Interpretacja treści i dodatkowe założenia

Treść:

"Do walizki o ograniczonej pojemności C chcemy załadować przedmioty o jak największej wartości, mając jednak na uwadze, że każdy z nich zajmuje pewna objętość. Mając n przedmiotów wraz z ich wartościami oraz objętościami, znajdź przy użyciu Algorytmu Ewolucyjnego zestaw rzeczy mieszczących się w walizce o największej sumarycznej wartości."

Jest to problem zwany w literaturze problemem plecakowym [knapsack problem].
Do przyjętych przez nas założeń należą rozmiar problemu (tj. ilość przedmiotów) od 50 do 150.

Dla innych ilości przedmiotów nasz program oczywiście działa, niemniej jednak wszystkie parametry były strojone właśnie dla problemów znajdującym się w tym przedziale.

Dodatkowymi celami, jakie postawiliśmy sobie w naszym projekcie jest znalezienie rozwiązania korzystając z klasycznych wersji operatorów - to znaczy nie używając heurystyk, funkcji naprawczych, algorytmów zachłannych etc. Niezaprzeczalnie użycie takowych zapewniłoby lepsze parametry (i są one używane w wielu badaniach naukowych z dobrymi rezultatami), ale osiągnięte przez nas wyniki wydają się być wystarczające, a maksymalne uproszczenie zapewnia łatwiejszą analizę różnych czynników wpływających na działanie algorytmu.

Nie chcieliśmy też korzystać z poznanych na wykładzie strategii ewolucyjnych z uwagi na niskie możliwości ich modyfikacji.

Dla potrzeb testowania generujemy z jednakowym prawdopodobieństwem wagi przedmiotów od 1 do 15 oraz wartości przedmiotów od 10 do 750. Wartości te są czysto arbitralne i ustalone i w żaden sposób ze sobą nieskorelowane (w późniejszej części dokumentacji pokażemy analizę dla danych skorelowanych).

Wypunktowanie wkładu poszczególnych autorów (kto co zrobił)

Każdy po części wpływał na każdy aspekt rozwiązania, ale ogólny podział przedstawiałby się następująco:

- decyzje projektowe podejmowane wspólnie
- kod algorytmu i interakcji z użytkownikiem Łątkowski
- kod testów Sobieraj
- testy i analiza Sobieraj
- dokumentacja Łątkowski

Zestawienie ważnych decyzji projektowych.

Część decyzji podjęliśmy na starcie, część po analizie testowej.

1) Selekcja

Uznaliśmy że najlepiej będzie wybrać selekcję turniejową. Zapewnia ona możliwość przetrwania osobnikom, które przy zwykłej selekcji proporcjonalnej by nie przeżyły (a w przypadku naszego problemu powinny, bo niekiedy rozwiązanie niepoprawne po odjęciu zaledwie jednego przedmiotu staje się rozwiązaniem wręcz najlepszym). Znalezione przez nas badania potwierdzają tezę o skuteczności tej selekcji [1] [2] . Inną możliwością byłby liniowy ranking (ale, jak można dowieść matematycznie, jest to ekwiwalent selekcji turniejowej o rozmiarze turnieju 2) albo selekcja eksponencjalna, której nie implementowaliśmy z uwagi na fakt, że turniejowa działała dobrze, a prędkości obu były wg badań porównywalne.

Oczywiście warto zaznaczyć, że nie jest to idealne rozwiązanie-selekcja turniejowa cierpi na problem właściwy selekcji ruletkowej (która jest gorsza wydajnościowo), czyli brak gwarancji, że jakikolwiek, nawet najlepszy możliwy osobnik będzie wybrany w populacji, co może zakłócać poprawność wyniku [3].

2) Elitaryzm

Po stworzeniu algorytmu uznaliśmy, że sama selekcja turniejowa daje zbyt małą szansę na przeżycie jednostkom wybitnym i wprowadziliśmy elitaryzm, który zapewnia przeżycie najlepszym 15% populacji. Liczba 15% wynika z krótkich testów dla różnych wartości wszystkich innych parametrów (10% dawało zwykle lekko gorsze wyniki, 20% było z kolei przesadne).

3) Typ krzyżowania i mutacji

Początkowe założenie opierało się na twierdzeniu z badania [4], że w przypadku znanych problemów kombinatorycznych, mutacja jest wystarczająca i choć krzyżowanie zapewnia szybszą konwergencję, to pogarsza czas wykonania i nie zwiększa jakości rozwiązania. Opieraliśmy się więc na X szansy, że osobnik dostąpi mutacji, czyli że każdy bit zostanie odwrócony z prawdopodobieństwem Y (nazwaliśmy to roboczo "dużą mutacją").

Ostatecznie wyniki okazały się nie być satysfakcjonujące i zdecydowaliśmy się na wprowadzenie krzyżowania, co poprawiło zarówno konwergencję, jak i wyniki. Oczywiście wówczas wprowadziliśmy znacznie mniejszą mutację (nazwaną przez nas "małą mutacją"), czyli X prawdopodobieństwa na odwrócenie losowego bitu. To pozwoliło osiągnąć dobry balans miedzy eksploracją a eksploatacją.

Wybraliśmy krzyżowanie jednopunktowe, bo (znowu patrząc na [4]), krzyżowanie dwupunktowe nie zmienia znacząco wyniku, a krzyżowanie jednorodne [uniform] pogarszało znacznie czas wykonania i jakość rozwiązania. Punkt przecięcia jest wybierany losowo dla każdego aktu krzyżowania.

4) Funkcja dopasowania

Bazujemy na prostej funkcji dopasowania, przyjmującej następujące wartości:

- 0 (jeśli waga przedmiotów danego osobnika przekracza rozmiar plecaka)
- suma iloczynów poszczególnych bitów i przypisanych im wartości (jeśli rozmiar plecaka nie jest przekroczony)

Rozważaliśmy również funkcję dopasowania karającą mniej te geny, które tylko nieznacznie przekraczają wagę plecaka, ale niełatwo bez zachłannych algorytmów dobrać taki wzór, żeby nie otrzymać na sam koniec populacji składającej się w dużej mierze z osobników nielegalnych. Może gdybyśmy nie korzystali z naszej opisanej w 6) metody inicjalizacji, byłoby to konieczne - w naszym rozwiązaniu nie było.

5) Prawdopodobieństwo krzyżowania

Ustaliliśmy prawdopodobieństwo krzyżowania na 0.8. Jest to wartość często przewijająca się w literaturze. Krótkie, losowe testy dla różnych parametrów potwierdzały tezę, że wartość ta powinna oscylować wokół 0.8.

6) Inicjalizacja

Zaimplementowaliśmy inicjalizację zależną od stosunku rozmiaru plecaka do sumarycznej wagi przedmiotów. Dla przykładu, dla przedmiotów o średniej sumarycznej wadze 1200, a plecaka o rozmiarze 60, wygenerowanie z równym prawdopodobieństwem zer i jedynek jest bez sensu, jedynek musi być zaledwie kilka. Z kolei dla bardzo pojemnego plecaka inicjalizacja samymi zerami (czyli szykowanie się na najgorszy scenariusz) byłaby również nieoptymalna. Dlatego generujemy tyle jedynek, żeby mniej więcej połowa plecaka była na początku zapełniona (oczywiście średnio połowa).

Wspomniane niżej testy potwierdziły, że inicjalizacja samymi zerowymi osobnikami (tj. osobnikami o genotypie z samych zer) jest gorsza od naszej metody (choć trzeba przyznać, że nieznacznie).

7) Rozmiar populacji, prawdopodobieństwo mutacji, liczba generacji

Te wartości dobraliśmy już po stworzeniu algorytmu właściwego, niżej przedstawione testy prowadzące do otrzymanych wartości.

Lista wykorzystanych narzędzi i bibliotek

Program pisaliśmy w całości w C++, wykresy i analiza danych w Excelu. Do generacji losowych wartości stworzyliśmy klasę wykorzystującą bibliotekę <random> (Mersenne Twister, seedowanie za pomocą aktualnego czasu). Do pomiaru czasu użyliśmy biblioteki <chrono>.

Instrukcję pozwalającą na odtworzenie uzyskanych wyników

Program ma na celu demonstrację naszego algorytmu. Stąd ograniczyliśmy jego funkcjonalność do konsoli. Można wybrać dwa tryby-pierwszy pozwala na uruchomienie jednej instancji dla ustalonej przez użytkownika ilości przedmiotów i rozmiaru plecaka. Drugi zaś daje możliwość replikacji poniższych testów. Warto zwrócić uwagę, że z racji na uruchamianie wielu (czasem 20, czasem 50) instancji danego problemu dla różnych parametrów, wykonywanie się testów trwa względnie długo.

Cele i tezy przeprowadzonych badań

1. Pierwszym przeprowadzonym przez nas testem była wielowymiarowa analiza problemu w celu znalezienia najbardziej optymalnego (pod względem czasu i jakości rozwiązania) algorytmu. W tym celu modyfikowaliśmy:

-ilość generacji (od 200 do 1000, krok 200)

-rozmiar populacji (od 50 do 450, krok 50)

-prawdopodobieństwo mutacji (od 0.1 do 0.9, krok 0.1)

Testowaliśmy dla różnych trudności problemu (trudność oznaczamy jako **S**, zależy od ilości plecaków i stosunku rozmiaru plecaka do łącznej wagi przedmiotów, co udowodnimy niżej).

S jest równe ilorazowi rozmiaru plecaka i średniej wagi przedmiotów.

Jakość [quality] oznacza jakim procentem idealnego rozwiązania jest otrzymane przez nas rozwiązanie.

Rozważaliśmy ten problem w 4 wariantach zależnych od:

1) tego, czy inicjalizujemy "naszym", przedstawionym w 6) sposobem, czy inicjalizujemy samymi zerami
2) tego, którą wersję krzyżowania wybierzemy. Pierwsze krzyżowanie zakłada krzyżowanie tylko najlepszej połowy wyselekcjonowanej populacji (każda para ma dwójkę dzieci), drugie zwyczajne krzyżowanie w losowej kolejności całej wyselekcjonowanej populacji (każda para ma jedno dziecko)

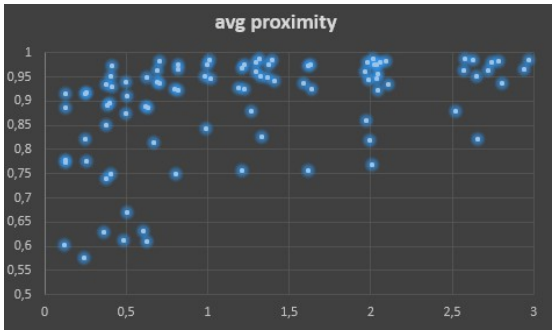
Jak można zobaczyć niżej, żadna z 4 kombinacji nie różniła się znacząco od drugiej, w końcu wybraliśmy rozwiązanie z inicjalizacją naszym sposobem i zwyczajnym algorytmem krzyżowania, jako że jest to rozwiązanie bardziej czytelne, uniwersalne dla problemów o innym rozmiarze niż 50-150, a także daje nieco lepsze wyniki.

Wyniki testów były zaskakujące, naszą tezą na wstępie było bowiem *"Populacja musi być około 100, generacji musi być wiele, rozmiar mutacji powinien oscylować wokół 0.7".*

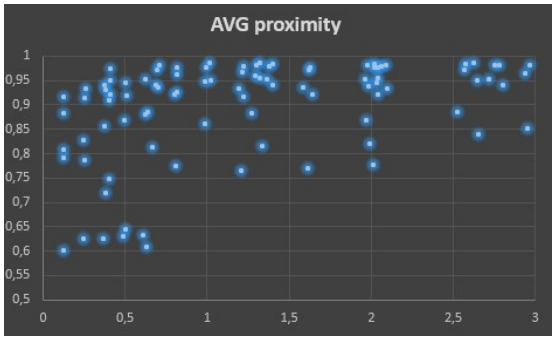
Okazuje się, że dla naszego problemu populacja musi być duża, liczba generacji mała, zaś rozmiar mutacji 0.5 lub 0.7.

Ostateczny wybór punktu jest niełatwy, zależy oczywiście od tego, na ile zależy nam na czasie, a na ile na dokładności. Postawiliśmy sobie za cel maksymalnie zredukować czas przy jednoczesnym sztywnym ograniczeniu, że w najtrudniejszej wersji problemu (**S**=0.05), średnia jakość rozwiązania (przy uruchomionych parudziesięciu instancjach testowych) nie może być gorsza niż 97% najlepszego rozwiązania. Tak właśnie otrzymaliśmy nasze parametry algorytmu, przedstawione już wyżej.

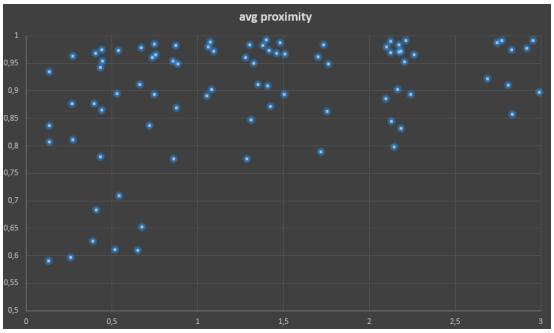
Oczywiście wyniki mogłyby się trochę różnić, gdybyśmy testowali raz jeszcze z racji na losowość algorytmu (którą staraliśmy się ograniczyć uruchamiając każdą kombinację 20 razy i uśredniając wyniki).



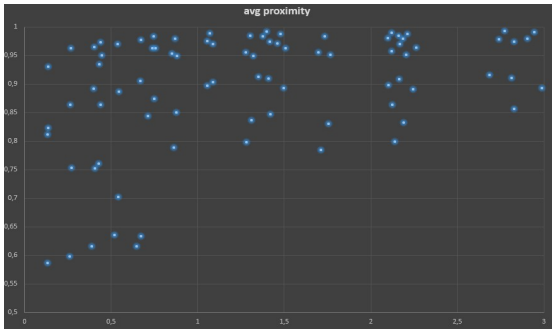
Dla inicjalizacji naszym sposobem i elitarnego krzyżowania



Dla inicjalizacji zerami i elitarnego krzyżowania



Dla inicjalizacji naszym sposobem i normalnego krzyżowania (stąd wybraliśmy punkt)



Dla inicjalizacji zerami i nieelitarnego krzyżowania

Parametry ostatecznie wybranego punktu:

Prawdopodobieństwo mutacji	Rozmiar populacji	Ilość generacji	Ilość przedmiotów	Jakość	Czas	Rozmiar plecaka	S (trudność)
0,5	250	200	50	0,992917	0,602951	60	0.05
0,5	250	200	150	0,983489	0,745425	60	0.075
0,5	250	200	150	0,978244	0,893204	60	0.15

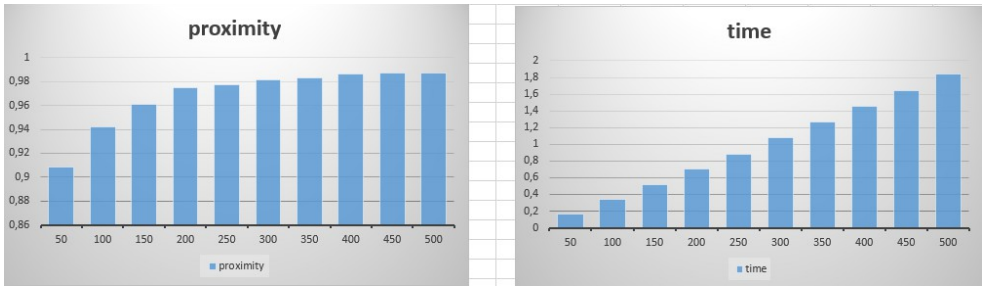
Jak widać, te wykresy nie różnią się znacznie i w każdym z nich moglibyśmy wybrać dobre punkty.

Tak wygląda wykres dopasowania dla przykładowej instancji problemu:

Można by uznać, że 200 generacji jest wartością zatem przesadną, skoro wiele problemów zatrzymuje się dla 120. Ale wybieramy tę wartość, gdyż jest ona bezpieczna.

Teraz pokażemy wpływ różnych czynników na zachowanie algorytmu. Wszystkie testy przeprowadzamy dla najtrudniejszego rozważanego przypadku, tzn. **S**=0.05 (w tym przypadku rozmiar plecaka 60) i największej rozważanej ilości (150 przedmiotów).

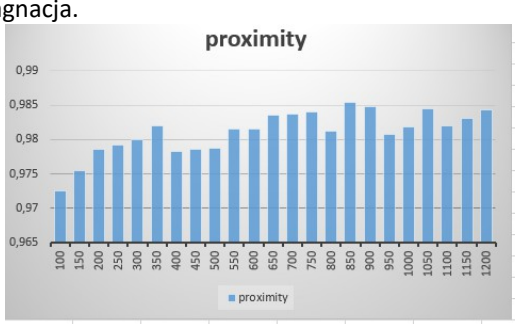
2. Wpływ rozmiaru populacji na czas wykonania i jakość algorytmu
Teza: Wraz ze wzrostem populacji wzrośnie czas wykonania, ale rośnie też jakość. Od pewnych wartości jakość może się zmniejszyć.

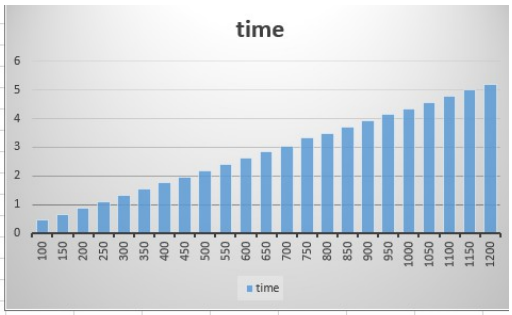


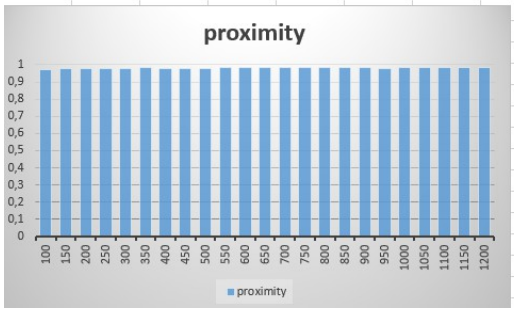
Wnioski: Istotnie czas rośnie w miarę rozmiaru populacji liniowo-jest to logiczne, bo ilość wykonanych operacji (krzyżowanie, mutacja) rośnie liniowo wraz ze wzrostem populacji. Jakość rośnie nieregularnie, od populacji 400 nastąpiła stagnacja jakościowa. Widać że między 250 a 500 jest pewna różnica w jakości, ale jednocześnie algorytm działa około 2 razy dłużej-stąd nie wybieraliśmy takich rozwiązań w rozważaniach dot. optymalnego rozwiązania. Nie nastąpiło pogorszenie jakości przy dużych rozmiarach populacji-najwyraźniej to zjawisko nie zachodzi w przypadku naszego problemu, lub zachodzi dla dużo większych, nietestowanych rozmiarów.

3. Wpływ ilości generacji na czas wykonania i jakość algorytmu

Teza: Im więcej generacji, tym dłużej wykona się algorytm. Do pewnego etapu jakość będzie się polepszała, potem nastąpi stagnacja.







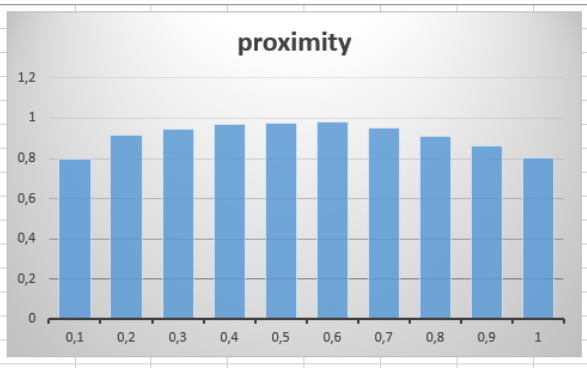
Wnioski: Nie jest zaskakujący wzrost liniowy, analizując algorytm łatwo wywnioskować, iż przy A generacjach wszystkie operacje wykonają się AX razy, przy 2A generacjach 2AX razy.

Dwa wykresy jakości rozwiązania mają na celu ilustrację, że nasza teza była błędna-ilość generacji generalnie nie wpływa w żaden wyraźny sposób na jakość znalezionego rozwiązania. Powodem tego faktu może być to, że nasz algorytm ma stosunkowo dużą populację i konwergencja w optimum lokalnym/globalnym następuje szybko. Warto dodać, że dla innej skali problemu np. 1000 przedmiotów, ilość koniecznych generacji byłaby zupełnie inna. W celu zapewnienia lepszych wyników dla nierozważanych ilości, można by uzależnić ilość generacji od tego, czy w ostatnich generacjach nastąpiła poprawa.

4. Wpływ prawdopodobieństwa mutacji na czas wykonania i jakość algorytmu

Teza: Mutacja będzie miała niewielki bądź pomijalny wpływ na czas (im większe prawdopodobieństwo, tym większy czas), jakość będzie najlepsza dla 0.5.

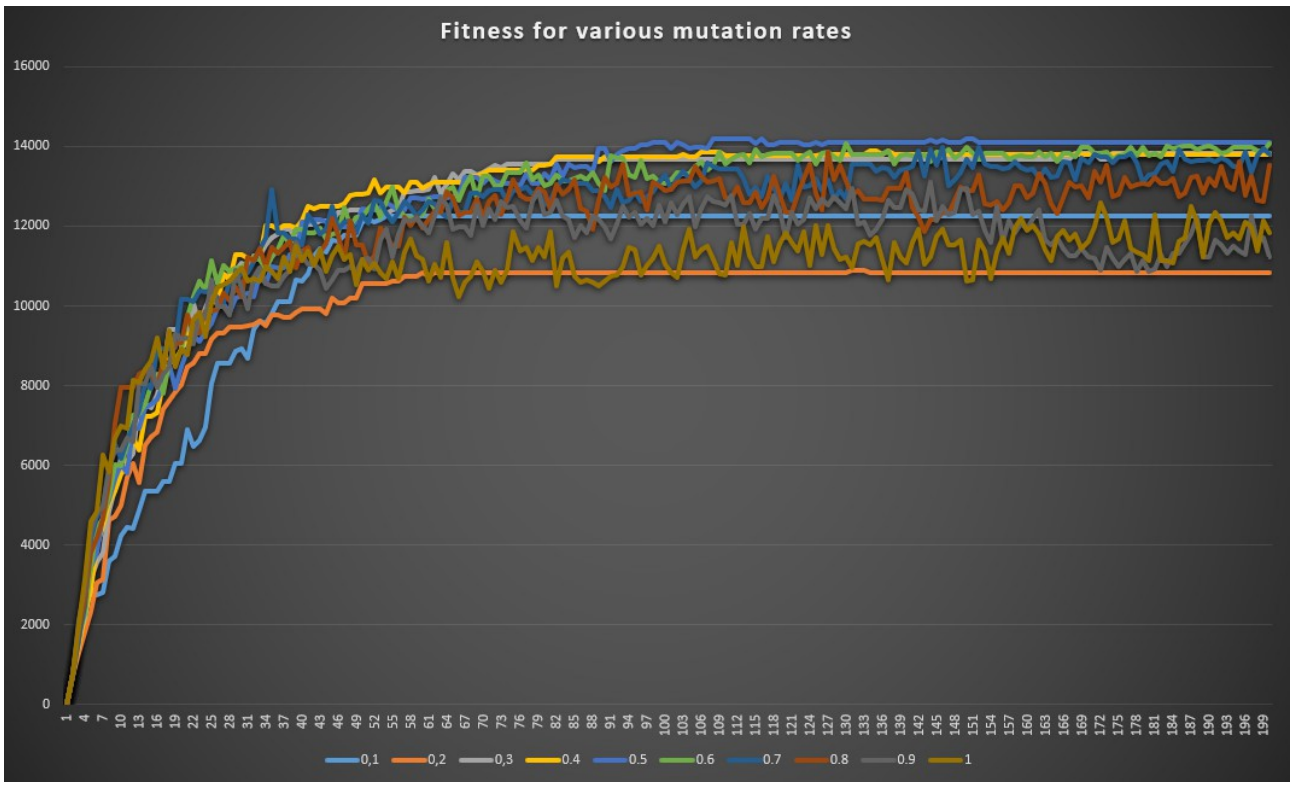
time
0,856911
0,865206
0,873696
0,876698
0,882728
0,888904
0,887615
0,883786
0,87921
0,874733



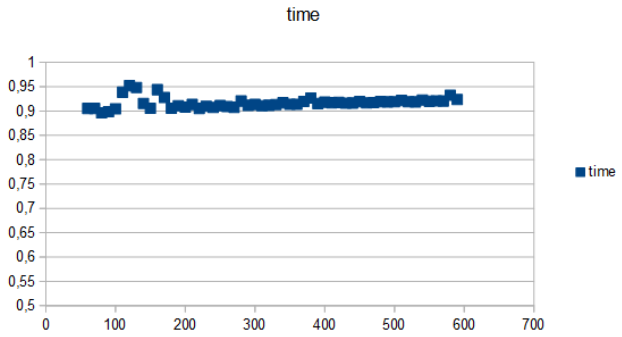
Wnioski:

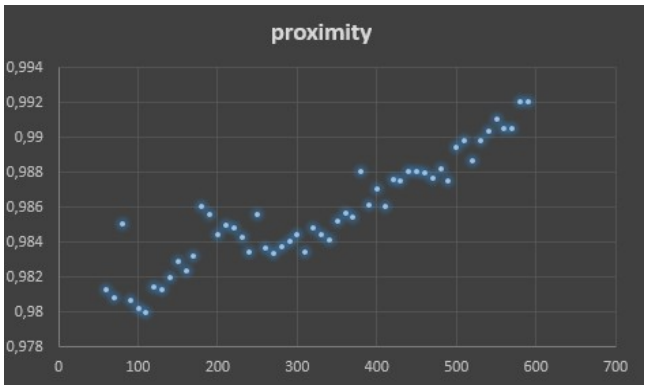
Dla zbyt małych wartości mutacji nie mamy wystarczającej eksploracji i eksploatacji, dla za dużych pogarszamy zbytnio otrzymane dobre wyniki i nie dajemy dobrym osobnikom rozsądnych szans na przetrwanie w niezmienionej formie. Także dla tych konkretnych parametrów okazuje się, że 0,5 i 0,6 to są zdecydowanie najlepsze parametry. Potwierdza to nasz wybór parametru 0,5 (zapewne 0,6 miał gorsze wyniki czasowe). Czasy jak widać na tabelce powyżej nie ulegają zmianie.

Poniżej zamieszczamy jeszcze przykład jak zmienia się w czasie nasza funkcja "fitness" w zależności od współczynnika mutacji



5. Wpływ czynnika S na czas wykonania i jakość algorytmu
Teza: Im mniejszy ten czynnik, tym trudniej znaleźć rozwiązanie, więc jakość będzie gorsza. Czas się nie zmieni, bo ten czynnik nie wpływa na ilość wykonywanych operacji





Wnioski:
W pewnym miejscu wykresu widać niewielką nieregularność, ale można wciąż wyciągnąć następujące wnioski-czas się znacznie nie zmienia. Z kolei, choć różnice w jakości rozwiązania wydają się nieznaczne, to można wciąż zaobserwować pewien wzrost. Te różnice są nieznaczne, bo wybieraliśmy punkt, który będzie bardzo dobrze działał dla takich niskich rozmiarów plecaka. Jeśli byśmy w analizie przedstawionej w 1 punkcie rozważali np. dużo łatwiejsze rozwiązania, zaobserwowalibyśmy, że algorytm gorzej radziłby sobie dla tak restrykcyjnego plecaka. Niemniej jednak wciąż możemy wyciągnąć wniosek, że nasze rozwiązanie spełnia przyjęte przez nas kryteria - nawet dla przypadku o S=0.05 nie osiąga gorszych wyników niż 0.97 jakości, a nawet spokojnie osiąga lepsze wyniki. I że faktycznie im większy rozmiar plecaka, tym łatwiej dopasować dobre rozwiązanie.

6. Analiza danych skorelowanych i nieskorelowanych
Teza: Jako że algorytm był testowany i dostrajany do danych nieskorelowanych, dla danych skorelowanych będzie on miał gorsze wyniki jakościowe, czas nie ulegnie zmianie

Dane wzięliśmy ze zbiorów Davida Pisingera [5]. 100 przedmiotów, S=0.05, dla każdej ze 100 instancji uruchomiliśmy algorytm 50 razy i uśredniliśmy wyniki.

Dane nieskorelowane:

Średnia jakość: 0,981435
Średni czas: 0,753279

Dane słabo skorelowane

Średnia jakość: 0,959921
Średni czas: 0,747024

Dane mocno skorelowane:

Średnia jakość: 0,97666
Średni czas: 0,753151

Dla S=0.05

Sprawdziliśmy też dane słabo skorelowane i mocno skorelowane dla S=0.5
Dla słabo skorelowanych średnia jakość wynosiła 0.986785, dla mocno skorelowanych średnia jakość wynosiła 0.982002

Wnioski: Nasza początkowa teza okazała się być fałszywa. Jakość dla danych nieskorelowanych zgadza się ze średnią jakością dla problemu o takiej trudności. Czas faktycznie nie ulega zmianie, bo ilość przedmiotów jest taka sama.

Niemniej jednak, wbrew naszym oczekiwaniom, program działa relatywnie dobrze dla danych słabo skorelowanych i mocno skorelowanych. Co ciekawe, jakość rozwiązania dla danych mocno skorelowanych jest wręcz lepsza niż dla danych słabo skorelowanych dla S=0,05. Zbadaliśmy zatem jak się te dane zachowają dla S=0,5, tutaj już wyniki były porównywalne (widać dla danych słabo skorelowanych tak restrykcyjne plecaki są problemem trudniejszym).

Jest to wynik przeczący naszej intuicji-w teorii i w praktyce dane skorelowane powinny być trudniejsze [6], a tutaj tak naprawdę nie różnią się zbytnio od nieskorelowanych. Może wyjaśnieniem byłby fakt, że naszych przedmiotów jest stosunkowo mało. W podanym badaniu [6] przedmiotów było o parę rzędów wielkości więcej-wtedy różnice między danymi skorelowanym a nieskorelowanymi mogłyby się okazać bardziej widoczne. Niemniej jednak przeprowadzenie takiego badania wymagałoby dużo więcej testów, bo trzeba by wziąć pod uwagę inne czynniki (np. niedostosowanie algorytmu do problemów rzędu n=100000)

BIBLIOGRAFIA:

[1] https://tik-old.ee.ethz.ch/file//6c0e384dceb283cd4301339a895b72b8/TIK-Report11.pdf
[2] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.552&rep=rep1&type=pdf
[3] https://www.researchgate.net/publication/2853422_An_Empirical_Comparison_of_Selection_Methods_in_Evolutionary_Algorithms
[4] https://www.hindawi.com/journals/tswj/2014/154676/
[5] http://hjemmesider.diku.dk/~pisinger/codes.html
[6] https://www.researchgate.net/publication/2548374_Core_Problems_in_Knapsack_Algorithms