|  |  |
| --- | --- |
| **Problem** | **Metaheurystyka** |
| Problem nr 2 | Algorytm Genetyczny |

Naszym zadaniem było zaimplementowanie wybranej metaheurystyki do wybranego przez nas problemu szeregowania zadań. Naszym celem było zminimalizowanie całkowitego czasu wykonania wszystkich operacji. Systemem obsługi zadań naszych maszyn był system Jobshop. Nasze operacje mogą być wznawiane z karą +25% czasu.

1. Jako metaheurystykę wybraliśmy *Algorytm Genetyczny.* Jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. Działanie algorytmu przebiega następująco:

* Losowana jest pewna **populacja początkowa**
* Populacja poddawana jest **selekcji**, w celu przeniesienia najlepszych rozwiązań do następnego pokolenia. My wykorzystaliśmy metodę **selekcji turniejowej**. Metoda ta polega na wybraniu do turnieju losowych osobników populacji. Z turnieju wychodzi zwycięsko osobnik z najlepszą funkcją celu. Liczba turniejów jest parametrem, który mogliśmy sobie dowolnie zmieniać. Wygrane jednostki przechodzą do następnego kroku.
* Następuje teraz etap **krzyżowania.** Jednostki, które wyszły zwycięsko z turnieju są ze sobą kojarzone. Z nich rodzą się nowe, teoretycznie lepsze osobniki, które przechodzą do kolejnego pokolenia. Nasze **krzyżowanie** polegało na „odcinaniu” zadań osobników w pewnym momencie jego listy zadań a następnie krzyżowo dołączało się brakujące zadania w takiej kolejności w jakiej występowały u drugiego osobnika. **Krzyżowaniem** mogliśmy manipulować wybierając inny punkt przecięcia.
* Po stworzeniu nowych osobników, mogły one podlegać **mutacji**, czyli drobnej losowej zmiany. U nas polegało to na zamienieniu ze sobą miejscami dwóch zadań na liście. Szansa na zajście **mutacji** była określana procentowo i jest ona kolejnym parametrem, którym mogliśmy sterować.
* **Selekcje**, **krzyżowanie** oraz **mutacje** powtarzamy aż uzyskamy taką samą liczbę nowych osobników jak liczba pierwszej populacji. Wtedy rodzi się nowe pokolenie.

1. Naszym pierwotnym założeniem generowania rozwiązań było wkładanie najpierw wszystkich operacji pierwszych na maszyny, a następnie od momentu gdzie się zakończyła najpóźniejsza operacja pierwsza, wrzucaliśmy operacje drugie. Niestety później na etapie ewolucji wyniki niewiele różniły się od wartości początkowych, więc uznaliśmy, że warto by zmienić konwencję. Ostatecznie zadania wrzucamy po kolei, tak jak są umiejscowione na liście. Najpierw pierwsza operacja zadania, później druga a potem następne zadanie.
2. Strojenie metaheurystyki

Za parametry podstawowe metaheurystyki przyjęliśmy:

* Wielkość populacji = 200
* Współczynnik mutacji = 10%
* Liczba kandydatów do turnieju = 20
* Liczba zwycięzców turniejów (liczba turniejów) = 20
* Współczynnik podziału = 50%

Dla tych parametrów przeprowadziliśmy testy na 40 instancjach zawierających 50 zadań, po 5 przerw konserwujących na maszynie, oraz czas trwania operacji i przerw w przedziale 1-20, dla 100 generacji algorytmu genetycznego.

Wynikiem jest porównanie najlepszego osobnika z ostatniej generacji względem średniej funkcji celu dla pierwszej populacji.

Nasze testy wykazały, że algorytm poprawił funkcję celu o średnio **24,74%.** Teraz zmieniając parametry algorytmu będziemy porównywali nowe wyniki względem tej wartości. Następne testy będą wykonywane na 20 instancjach dla każdego punktu pomiarowego, dla 100 generacji algorytmu genetycznego.

* Współczynnik mutacji

Z wykresu wynika, że zwiększając prawdopodobieństwo mutacji, metaheurystyka coraz bardziej poprawia funkcje celu. Naszym zdaniem, dzieje się tak dlatego, ponieważ, pomimo tego, że mutacja wprowadza losowe zmiany (lepsze lub gorsze), to w etapie następnej generacji selekcja najprawdopodobniej wybierze dobre osobniki (te, w których zaszła dobra zmiana).

* Liczba kandydatów do turnieju

Z wykresu nie wynika jednoznaczna zależność poprawy wyniku od liczby kandydatów do turnieju przy niezmienionych pozostałych parametrach metaheurystyki. Jednakże można zauważyć bardzo delikatny wzrost optymalizacji przy większej liczbie kandydatów. Wynika to z tego, że przy dużej liczbie kandydatów bardzo często będą wybierane dobre osobniki, lecz będą się one później powtarzać i na etapie krzyżowania nie nastąpi żadna poprawa. Nawet jeśli na etapie mutacji zajdzie jakaś zmiana w jednym to w krzyżowaniu przy następnym obiegu algorytmu genetycznego nadal nie będzie widać znaczącej różnicy.

* Liczba zwycięzców turniejów

Powyższy wykres obrazuje podobną sytuację co poprzedni. Generalnie oba parametry łączą się ze sobą i powodują niewielką zmianę w optymalizacji. Wnioski do tych testów są analogiczne do poprzednich.

* Liczba kandydatów do turniejów i liczba zwycięzców turniejów

Po raz kolejny postanowiliśmy podjąć próbę sprawdzenia czy oba te parametry wpływają znacząco na wynik i zwiększaliśmy je razem. Kolejny raz testy nie wykazały znaczącej poprawy funkcji celu, a jedynie to co poprzednie wykresy.

Teraz zaczęliśmy od dużej liczby kandydatów, którą zmniejszaliśmy, zaś liczba turniejów zwiększaliśmy. Nadal nie widać jakiegoś konkretnego zachowania algorytmu. Jedyną odskocznią jest liczba kandydatow = 30 i liczba turniejów = 75 i te wartości przyjmiemy w głownych testach.

* Współczynnik podziału

Z wykresu wynika, że gdy współczynnik jest mały i dochodzi do 0.5 (współczynnik 0.5 mówi o tym, że przecinamy osobniki w połowie), optymalizacja niewiele różni się od bazowej. Współczynnik zbliżając się do jedności zmniejsza optymalizacje funkcji celu(poza jednym skokiem, który, wydaję nam się, jest po prostu losowy i zależy od wygenerowanych przez nas instancji). Optymalizacja się zmniejsza, gdyż w im późniejszym miejscu przetniemy nasze osobniki, tym mniej będą się różniły osobniki potomne od swoich rodziców, co spowoduje przenoszenie do następnych pokoleń osobników praktycznie takich samych, więc funkcja zamiast się optymalizować może słabnąć.

1. Testy główne

Do testów głównych przeszliśmy wykorzystując następujące parametry algorytmu genetycznego:

* Wielkość populacji = 200
* Współczynnik mutacji = 100%
* Liczba kandydatów do turnieju = 30
* Liczba zwycięzców turniejów (liczba turniejów) = 75
* Współczynnik podziału = 50%

Postanowiliśmy przeprowadzić testy tych parametrów na tych samych instancjach co wcześniej. Średnio algorytm poprawiał funkcję celu o 33,38%, co okazuje się sukcesem w doborze parametrów. Jeszcze bardziej utwierdziło nas to w przekonaniu by używać ich w dalszych testach. W dalszej części jeśli nie jest nic wspomniane o wartości danego parametru, to znaczy że ma on wartość "bazową":

**liczba zadań = 50**, **liczba przerw = liczba zadań/10, czas trwania operacji i przerw = 1-20.**

* Procent polepszenia funkcji celu względem rosnącej liczby generacji algorytmu genetycznego

Jak widać na wykresie, im więcej razy potraktujemy populację algorytmem genetycznym, tym lepszy wynik końcowy uzyskamy. Można zauważyć, że z początku jest duży skok wartości funkcji celu, lecz od pewnego momentu rośnie już bardzo wolno, wręcz czasami niezauważalnie.

Można wysnuć wnioski, że im dłużej działa algorytm genetyczny tym lepszy wynik uzyskujemy, dlatego też nie trzeba było robić oddzielnych testów dla parametrów czasu, gdyż liczba generacji jest jakby takim „czasem”.

* Polepszenie funkcji celu przy zmiennej liczbie zadań

Patrząc na wykres widzimy niewielką poprawę funkcji celu, gdy zwiększamy liczbę zadań. Po pewnym szczytowym momencie, który jest bliski 50 zadaniom (naszemu standardowemu parametrowi) optymalizacja funkcji celu zaczyna maleć. Niestety nasze pomiary kończą się przy 200 zadaniach, gdyż nawet już dla tej ilości testy trwały bardzo długo, a co więcej dla 250 zadań program wyrzucił błąd o braku pamięci. Podsumowując, patrząc na trend na naszym wykresie możemy dojść do wniosku, że wraz ze wzrostem liczby zadań optymalizacja funkcji celu będzie maleć. Może to być spowodowane tym, że im większa liczba zadań tym większe możliwości krzyżowań i mutacji, a przez to, że przyjęliśmy tylko 100 generacji algorytmu genetycznego to nie jesteśmy w stanie uzyskać jeszcze lepszych wyników, które moglibyśmy dostać po większej ilości ewolucji.

* Polepszenie przy zwiększaniu maksymalnego czasu i operacji i przerwy konserwacyjnej

Z wykresu wynika, że jednoczesne zwiększanie maksymalnego czasu operacji i maksymalnego czasu trwania przerwy konserwacyjnej nie wpływa dużo na optymalizacje funkcji celu, wartości są generalnie zbliżone do siebie. Po tym teście uznaliśmy, że warto by jeden z tych parametrów zostawić stały a drugi zmieniać i zobaczyć co z tego wyniknie.

* Optymalizacja funkcji przy niezmienionym maksymalnym czasie przerw konserwacyjnych i zmiennym maksymalnym czasie operacji

W tej konfiguracji parametrów ponownie nie widać żadnych wielkich zmian. Spowodowane jest to tym, że zwiększając maksymalny czas trwania operacji a przerwy zostawiając w niezmienionej konwencji (czas trwania 1-20) teoretycznie nie szkodzimy algorytmowi ani mu nie pomagamy. Dzieję się tak, ponieważ algorytm działa jedynie na zadaniach i operacjach, przerw w ogóle nie rusza, czyli jeśli przerwy nie są coraz dłuższe to nam nie przeszkadzają. Powinno to być udowodnione poprzez kontrargument na następnym wykresie.

* Optymalizacja funkcji przy zmienionym maksymalnym czasie przerw konserwacyjnych i stałym maksymalnym czasie operacji

Jak widać, przypuszczenia się sprawdziły i zwiększone czasy trwania przerw konserwacyjnych zmniejszą procent optymalizacji funkcji celu. Niestety algorytm nie może tych przerw ani przesunąć, ani skrócić, co utrudnia mu lepsze uszeregowania.

* optymalizacja funkcji celu przy zmianie liczby przerw

(n - liczba zadań)

Jak wyraźnie widać zwiększanie liczby przerw powoduje proporcjonalny spadek procentu optymalizacji funkcji celu. Występuje tu zależność liniowa. Dzieje się tak, gdyż przerwy są niezmienne, niepodzielne, nie można ich przesunąć ani w żaden sposób skrócić, czyli krócej mówiąc nie da się ich w żaden sposób optymalizować i zawsze "psują" wynik tak samo, w tym samym stopniu.

**Wnioski**

Przy strojeniu algorytmu genetycznego praktycznie niemożliwe jest znalezienie najlepszego zestawu parametrów, gdyż ich kombinacje są nieskończone, jak również polepszenie optymalizacji zależy od instancji, na której wykonujemy szeregowanie. Przeprowadziliśmy serię testów i wybraliśmy najlepszą kombinację parametrów, jaką mogliśmy wywnioskować z owych testów. Nasze strojenie zadziałało, algorytm dawał lepsze wyniki, lecz należy zaznaczyć, że zawsze będą istnieć jakieś (chociaż minimalnie) lepsze parametry.

Fakt, że najlepsze wyniki otrzymywaliśmy przy szansie na mutację równej 100% jest dziwny, gdyż standardowo zalecana jest szansa około 1%. Może to wynikać z naszego sposobu implementacji mutacji, gdzie zamieniane miejscami są tylko 2 zadania. Gdyby mutacja wprowadzała większe zmiany (zamienia więcej zadań i np. jeszcze dodatkowo liczba zamienionych zadań jest losowa), prawdopodobnie rezultaty strojenia byłby inne. Warto zaznaczyć, że przy szansie na mutację równej 100% mutowaniu ulega każdy osobnik, także ten najlepszy. Im lepszy jest osobnik, tym większa szansa, że mutowania go pogorszy. Uważamy, że w ten sposób algorytm, gdy wyniki będą bliskie optimum, będzie je optymalizował coraz gorzej, a osobnik optymalny prawdopodobnie w ogóle nie powstanie, bo mutacja zawsze będzie go "psuła". Mimo wszystko musimy być konsekwentni z własnymi rezultatami i w testach przyjmujemy szansę na mutację 100%.

Zmiany współczynnika podziału nie przynosiły jednoznacznych rezultatów. Wiedzieliśmy na pewno, że współczynnik nie może być bardzo duży, gdyż wtedy osobniki-dzieci byłby prawie identycznymi kopiami rodziców, ani nie może być bardzo mały, gdyż wtedy osobniki-dzieci nie miałyby prawie nic wspólnego z rodzicami. Oba te przypadki zaprzeczają idei algorytmu genetycznego.

Tak samo zmiany liczby turniejów i kandydatów do turniejów nie dawały zbyt sensownych rezultatów. Zauważyliśmy delikatne polepszenie funkcji celu przy jednej z kombinacji tych parametrów oraz w jej okolicach (na wykresie) i wybraliśmy tę kombinację. Kolejne testy potwierdziły, że faktycznie jest ona korzystna, chociaż ciężko powiedzieć dlaczego akurat taka przynosiła dobre rezultaty.

Po właściwych testach doszliśmy do dosyć oczywistego wniosku, że im więcej kolejnych populacji tworzy algorytm, tym bardziej optymalizuje się wynik. Po prostu każda nowa populacja jest lepsza (w różnym stopniu) od poprzedniej.

Kolejnym wnioskiem jest to, iż zwiększenie liczby zadań sprawia, że optymalizacja staje się mniej skuteczna, potrzeba więcej osobników i nowych generacji, aby zachować poziom optymalizacji z mniejszej liczby zadań.

Zmiana przedziału czasu trwania operacji nie ma większego wpływu na przebieg optymalizacji, za to im dłuższe są przerwy, tym mniejsza jest optymalizacja, gdyż ich w żaden sposób nie da się "poprawić", są "stałe, nieruchome i niepodzielne" na maszynach.

Podobnie wzrost liczby przerw powoduje pogorszenie optymalizacji.