Maciej Oziębły 184147
Tomasz Wojciechowski 184132

Sterowanie procesami dyskretnymi
Sprawozdanie 4

Zakres sprawozdania:

- **Problem przepływowy z kryterium C średnie i opóźnieniami transportowymi**
- **Stworzenie rozwiązania przybliżonego za pomocą algotymy NEH**
- **Próba ulepszenia otrzymanych wyników za pomocą algorytmu symulowanego wyżarzania**

Spis treści

Środowisko

System operacyjny: Windows 8 Professional
Platforma programistyczna: .NET 4.5
Język programowania: **C#**
IDE: Visual Studio 2012 Ultimate (wersja z MSDNAA)
Komputer wypozażony w 2 rdzeniowy procesor korzystający z technologii Hyper-threading (więc mogący wykonywać równolegle 4 operacje)

Opis problemu

Rozwiązywany problem jest bardzo podobny do tego, który rozwiązywaliśmy na 2 laboratorium. Jest to klasyczny problem przepływowy, lecz tym razem rozwiązywaliśmy go względem kryterium średniego czasu zakończenia zadań oraz dołożyliśmy do niego opóźnienia transportowe. Opóźnienie trasportowe oznaczają, że pomiędzy wykonywaniem pracy nad tym samym zadaniem na dwóch różnych maszynach musi upłynąć pewien z góry ustalony okres czasu.

Aby stworzyć rozwiązanie problemu posłużylismy się algorytmem NEH, a następnie zastosowaliśmy algorytm symulowanego wyżarzania, aby poprawić otrzymane rezultaty.

Problem przepływowy z transportem z kryterium średniego czasu wykonania

Zmierzyliśmy się z problemem uszeregowania n zadań, które muszą zostać wykonane na m maszynach w zadanej kolejności, kiedy czas wykonania każdego z podzadań oraz transportu pomiędzy maszynami jest zdefiniowany z góry. Naszym kryterium jest uszeregowanie zadań tak, aby średni czas zakończenia wykonywania zadań był możliwie najmniejszy. Matematycznie rzecz biorąc dążymy do stworzenia takiego uszeregowania μ, że funkcja $CAvg(\mu)$ będzie przyjmowała jak najmniejszą wartość. Matematycznie rzecz biorąc:

$$CAvg(\mu) = \frac{1}{n}\sum_{i=0}^{n} c_{\mu_i}, gdzie\ c_{\mu_i}\ to\ czas\ zakończenia\ i-tego\ zadania\ w\ uszeregowaniu\ \mu$$

Opóźnienie transportowe oznacza konieczność czekania na przeniesienie wykonywania zadnaia pomiędzy maszynami. Czas rozpoczęcia kolejnego podzadania jest więc równy:

$$a_{\mu_{(i,j)}} = \max\left(c_{\mu_{(i,j-1)}} + transport_{\mu_{(i,j-1)}}, c_{\mu_{(i-1,j)}}\right), gdzie$$

$$a_{\mu_{(i,j)}}\ to\ czas\ rozpoczęcia\ wykonywania\ podzadania\ i\ zadania\ j\ uszeregowania\ \mu$$

$$c_{\mu_{(i,j)}}\ to\ czas\ zakończenia\ wykonywania\ podzadania\ i\ zadania\ j\ w\ uszeregowaniu\ \mu$$

$$transport_{\mu_{(i,j)}}\ to\ czas\ transportu\ dla\ zadania\ i\ pomiędzy\ maszynami\ \text{"j"}\ i\ \text{"}j+1\text{"}$$

Algorytm NEH

Ideą algorytmu NEH jest uszeregowanie zadań poprzez stworzenie pustego zbioru uszeregowanych zadań, a następnie wkładanie do niego kolejnych zadań (posortowanych po czasie przygotowania) sprawdzając wszystkie dostępne pozycje, tak aby po włożeniu nowego zadania zminimalizować funkcję celu.

Symulowane wyżarzanie

Symulowane wyżarzanie to algorytm heurystyczny oparty na zjawisku wyżarzania w metalurgii. W początkowej fazie agorytmu, gdy temperatura jest wysoka algorytm bardzo często mutując uszeregowanie wprowadza do niego zmiany, które nie są korzystne z punktu widzenia funkcji celu. W czasie działania algorytmu temperatura zmniejsza się, a szansa na wprowadzenie do uszeregowania zmiany, która pogorsza uszeregowanie maleje niemal do zera.

W naszym przypadku w każdej iteracji algorytm symulowanego wyżarzania wprowadza do szeregowania jedną z 3 mutacji:

- Swap – zamiana miejscami 2 zadań
- Insert – wstawienie 1 z zadań w inne miejsce
- Reverse order in block – odwrócenie kolejności zadań w bloku

Schemty stygnięcia:

- Geometryczny - (co cykl: $T := T * const;$)
- Wykładniczy – (co cykl: $T := Pow(T + 1, const) - 1$), przesunięte o 1, aby zbiegało do 0

Implementacja

Implementacja algorytmów, oprogramowania pomocniczego i oprogramowania testującego poprawność algorytmów oraz oprogramowania testującego szybkość działania algorytmów jest dostępna w plikach:

- AutoOrderingOptimization.cs – generyczna klasa z metodami umożliwiającymi przeprowadzenie symulowanego wyżarzania po podaniu handlera do funkcji celu, kontenera z obiektami oraz handlear do funkcji mutującej
- FileOperations.cs – klasa pomocnicza do ułatwienia przeprowadzania operacji na plikach
- Program.cs – główna klasa programu
- Task.cs – klasa przechowująca zadanie
- NEHOrdering.cs – klasa pozwalająca wykonać szeregowanie algorytmem NEH
- UnitTest1.cs – klasa przechowująca testy jednostkowe

Implementacja algortymów została przeprowadzona zgodnie z obeznymi trędami w programowaniu metodą **TDD** (Test Driven Developement) – przed implementacją każdej funkcjonalności tworzyliśmy test, który będzie testował poprawność naszej implementacji.

NEH

Nasza implementacja algorytmu NEH zawiera się w funkcji (a dokładniej rzecz biorąc metodzie statyczniej):

```
public static LinkedList<Task> NEHOrderingCAvg(LinkedList<Task> inputOrdering)
```

Po dostarczeniu tej funkcji uszeregowania używa ona algorytmu NEH do stworzenia nowego uszeregowania, które jest zwracane z tej funkcji. Działanie funkcji opiera się na wstawianiu kolejnych zadań do nowego uszeregowania za pomocą funkcji pomocniczej:

```
private static void InsertTaskToGetLowestCAvg(LinkedList<Task> currList, Task taskToInsert)
```

Funkcja ta sprawdza wszystkie możliwe miejsca, w które można włożyć nowe zadanie i wybiera najlepsze korzystając z pomocniczej funcji wyliczającej funkcję celu z nowo powstałych uszeregowań:

```
public static double CalcCAvgforPermutation(LinkedList<Task> currList)
```

Funkcja ta oblicza CAvg z podanej permutacji.

Symulowane wyżarzanie

Do wywołania symulowanego wyżarzania trzeba wywołać funkcję o nagłówku:

```
public static LinkedList<T> SimulatedAnealing<T>
            (LinkedList<T> inputList,
            Func<LinkedList<T>, double> targetFoo,
            Action<LinkedList<T>> mutatorFoo,
            double tempDecreasingSpeed, double startingTemperature,
            int iterationsWOChangeToStop)
```

Opis parametrów funkcji:

- inputList – uszeregowanie, na którym odbędzie się próba poprawy przez algorytm heurystyczny
- targetFoo – handler na funkcję celu
- mutatorFoo – handler na funkcję mutującą uszeregowanie
- tempDecreasingSpeed – prędkość zmniejszania się temperatury
- startingTemperature – temperatura początkowa
- iterationsWOChangeToStop – liczba iteracji bez wykonania żadnej mutacji pod rząd potrzebna do zakończenia algorytmu

Funkcja ta w każdej swojej iteracji mutuje uszeregowanie funkcją mutującą, a następnie oblicza zmianę długości uszeregowania za pomocą dostarczonej jej funkcji celu. W zależności od wyniki obliczeń i obecnej temperatury mutacja zostanie wprowadzona do aktualnego uszeregowania – im większa temperatura i mniej szkodząca uszeregowaniu zmiana tym większa szansa na pozostawienie mutacji. Funkcja zapamiętuje najlepsze kiedykolwiek znalezione przez nią uszeregowanie i zwraca je, gdy przez określoną liczbę iteracji nie nastąpiła mutacja aktualnego uszeregowania.

Wyniki

Zaimplementowany przez nas algorytm NEH dla zmodyfikowanego problemu daje takie same wyniki jak ten z programu z Pańskiej strony internetowej. Pozwala to domniemywać, że implementacja daje prawidłowe rezultaty.

Aby poprawić wyniki algorytmu NEH wykorzystaliśmy algorytm symulowanego wyżarzania, a dokładną zmianę wyników można zobaczyć w załączonym pliku „wyniki.xlsx”. Wnioski z jego działania opisane są w rozdziale „wnioski”.

Wszystkie otrzymane przeze mnie wyniki są dostępne w pliku wyniki.xslx. Opis poszczególnych zakładek:

- Mixed – wymieszanie wszystkich metod mutacji , stygnięcie geometryczne
- Swap – metoda mutacji swap, stygnięcie geometryczne,
- Insert – metoda mutacji insert, stygnięcie geometryczne
- ReverseOrderInBlock – metoda mutacji reverse order in block, stygnięcie geometryczne
- Exp – wymieszanie wszystkich metod mutacji, stygnięcie wykładniczne

Tabele wyników

Średnie przyspieszenie uszeregowania przez algorytm symulowanego wyżarzania dla różnych wielkości instancji

Ilość zadań	Ilość maszyn	Średnie przyspieszenie	Średnie maksymalne przyspieszenie (z 5 prób)
20	5	3,08%	3,77%
20	10	2,68%	3,35%
20	20	2,53%	3,05%
50	5	5,13%	5,96%
50	10	2,91%	3,72%
50	20	2,44%	3,06%
100	5	5,89%	5,89%
100	10	3,15%	3,76%
100	20	1,58%	2,20%
200	10	3,35%	3,35%
200	20	0,80%	1,16%
500	20	0,51%	0,82%

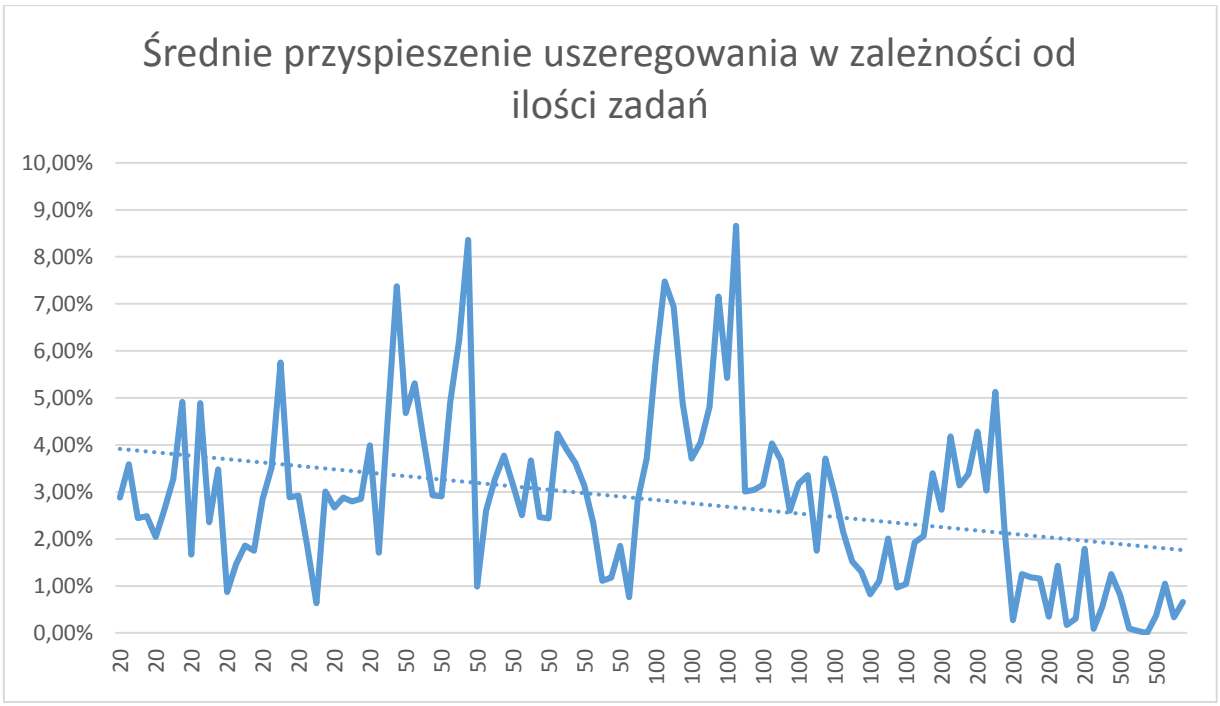
Poprawa uszeregowania dla różnych schematach stygnięcia

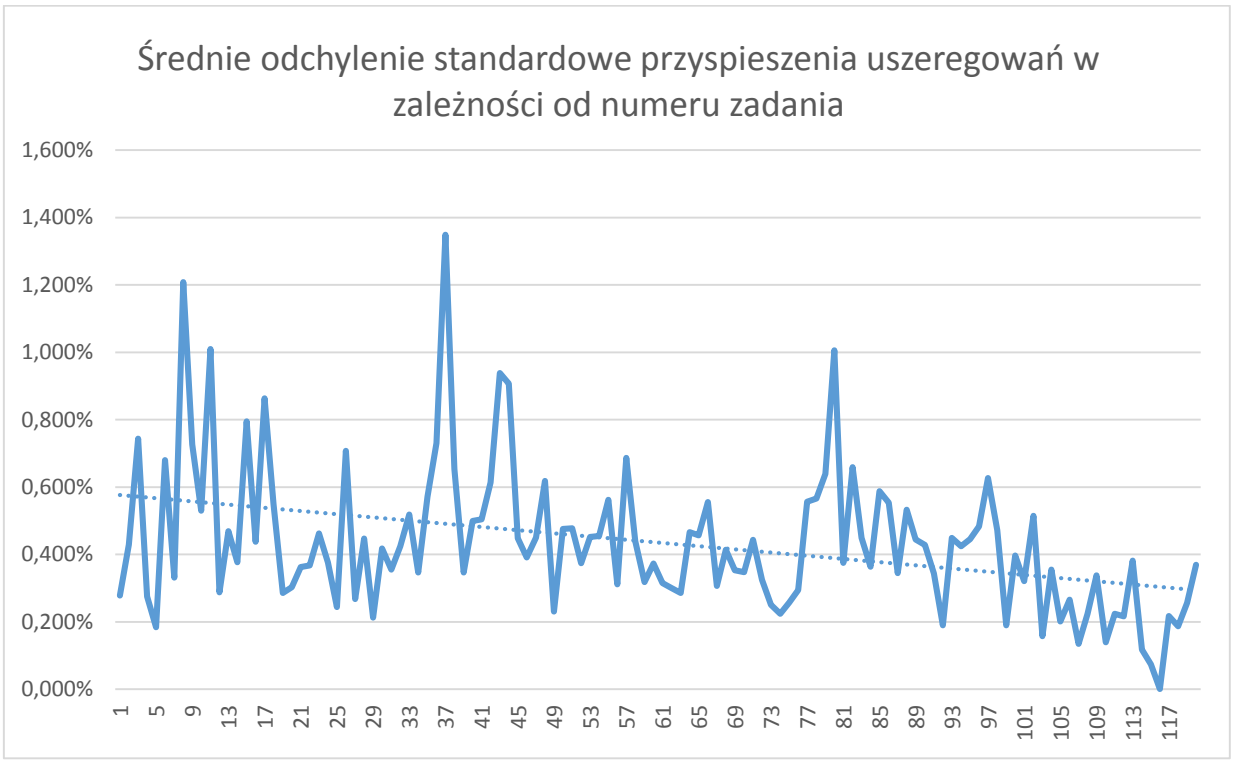
	Geometryczne	Wykładnicze
Globalne średnie przyspieszenie	2,838%	2,508%
Globalne średnie maksymalne przyspieszenie (dla 5 wywołań)	3,422%	3,171%

Poprawa uszeregowania dla różnych funkcji mutujących

	Swap	Insert	Reverse Order In Block	Mixed
Globalne średnie przyspieszenie	2,862%	2,508%	0,221%	2,838%
Globalne średnie maksymalne przyspieszenie (dla 5 wywołań)	3,396%	3,171%	0,471%	3,422%

Wykresy


Średnie przyspieszenie uszeregowania w zależności od ilości zadań
10,00%
9,00%
8,00%
7,00%
6,00%
5,00%
4,00%
3,00%
2,00%
1,00%
0,00%
20 20 20 20 20 20 20 20 50 50 50 50 50 50 100 100 100 100 100 100 100 100 200 200 200 200 200 500 500


Średnie odchylenie standardowe przyspieszenia uszeregowań w zależności od numeru zadania
1,600%
1,400%
1,200%
1,000%
0,800%
0,600%
0,400%
0,200%
0,000%
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101 105 109 113 117

Wnioski

Wykorzystanie algorytmów heurystycznych takich jak algorytm symulowanego wyżarzania pozwala poprawić jakość rozwiązań problemów, dla których jesteśmy w stanie wyznaczyć tylko rozwiązania przybiliżone.

Co ciekawe algorytm symulowanego wyżarzania tak samo jak algorytm NEH potrzebuje coraz więcej czasu, aby móc znaleźć lepsze rozwiązanie dla większej instancji problemu.

Algorytm symulowanego Wyżarzania jest algorytmem polegającym na losowych zmianach, co oznacza, że wielokrotne wykorzystanie tego samego algorytmu nawet na tej samej instancji problemu może przynieść różne rezultaty.

W zależności od parametrów z jakim zostało wywołane symulowane wyżarzanie (temperatura, prędkość stygnięcia, ilość iteracji bez zmian potrzebna do zakończenia algorytmu), algorytm wykonuje się przez różną ilość czasu. Otrzymane wyniki są zdecydowanie lepsze, gdy wywołamy algorytm kilkukrotnie z parametrami zmniejszającymi długość pracy algorytmu i zwróceniu najlepszej znalezionej wartości, niż przy jednokrotnym „długim" wywołaniu.

Z regresji liniowej widocznej na wykresach widać, że średnie przyspieszenie uszeregowania zadań spada wraz ze wzrosem wielkości instancji problemu. Z tabelki widać, że tak naprawdę duzy spadek widowczny jest dopiero przy największych instancjach problemu, wcześniej wartości są bardzo mocno zmienne.

Średnia wartość przyspieszenia dla wszystkich instancji problemu wyniosła 2,883% przy odchyleniu standardowym 1.847%. Widać tutaj, że algorytm ten daje bardzo zróżnicowane wyniki (średnia jest równa 1,56 wartości sigmy).

Algorytm symulowanego wyżarzania poprawił wyniki w bardzo dużym stopniu i uważamy, że zdecydowanie nadaje się do roziwiązywania tego typu problemów tym bardziej, że nie jest konieczna skomplikowana analiza problemu, a tylko znajomość funkcji celu i stworzenie funkcji mutującej uszeregowanie.

Zmiana schematu stydnięcia nie ma znaczącego wpływu na wydajność algorytmu. Zmiana schamtu stygnięcia z geometrycznego na wykładniczy spowodowało nieznaczne pogorszenie działania algorytmu, lecz doprowadziło przyspieszenia działania programu.

Co ciekawe mutacja tylko metodą swap jest okazała się być niemal tak samo wydajna jak wykorzystanie wszystkich metod jednocześnie. Metoda insert okazała się być nieco gorsza od metody swap. Metoda odwracania kolejności zadań w bloku całkowicie nie nadaje się do samodzielnego wykorzystania.