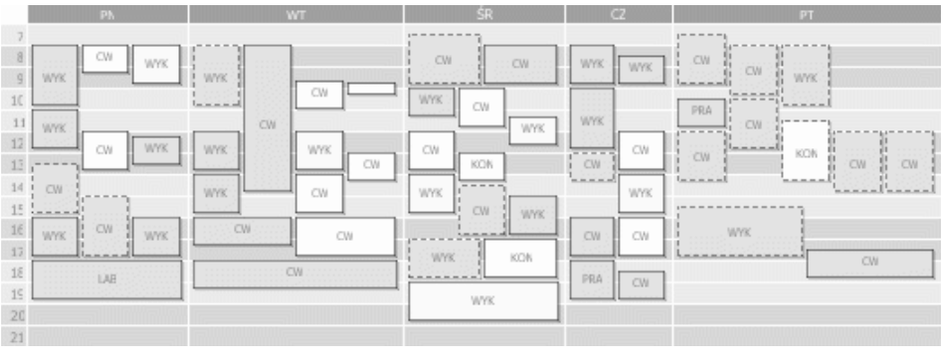
Ze świata USOS. Część 3 – O układaniu optymalnych planów zajęć w systemie USOS

Krzysztof CIEBIERA[*], *Marcin MUCHA*[*]

[*]Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

W tak dużej organizacji, jaką jest uczelnia wyższa, przy tak dużej liczbie uczestniczących w zajęciach studentów i wykładowców oraz tak bogatej ofercie dydaktycznej, przygotowanie dla wszystkich studentów indywidualnych planów zajęć, zgodnie z ich zainteresowaniami i preferencjami, to duże wyzwanie logistyczne. Wiele uczelni wymaga od studentów samodzielnego wyboru przedmiotów (takich jak *Wstęp do analizy* czy *Algorytmika*) oraz grup zajęciowych (takich jak grupa ćwiczeniowa numer 3, odbywająca się w środy między 8 a 10). Niektóre przedmioty są prowadzone poza macierzystym wydziałem studenta, np. lektoraty oferowane przez *Szkołę Języków Obcych*, czy zajęcia z wychowania fizycznego prowadzone przez *Studium WF*. **Rejestracja do grup** polega na tym, że student zapisany na **przedmiot** wybiera jedną z wielu **grup zajęć** (takich jak wykład, ćwiczenia, czy laboratorium) tego przedmiotu. Niektóre uczelnie układają plany tak, aby wszystkie zajęcia z jednego przedmiotu odbywały się w tym samym terminie, ale najczęściej grupy ćwiczeniowe mają różne terminy, różnych prowadzących, a nawet różne limity liczby osób. Studenci starają się tak dobierać grupy, aby trafić do lubianych profesorów, uniknąć kolizji i niepotrzebnych okienek między zajęciami. Nie zawsze jest to możliwe. Oto autentyczny (choć dość ekstremalny) plan jednego ze studentów studiującego równolegle dwa kierunki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego (w skrócie MIM).



Wydawać by się mogło, że najprostszą zasadą zapisów jest „kto pierwszy ten lepszy". Budzi ona jednak wiele kontrowersji, gdyż preferuje studentów mających sprawniejsze łącza internetowe czy komputery, niepotrzebnie stwarza też zagrożenie dla stabilności uczelnianych serwerów. Rozwiązanie zastosowane na Wydziale MIM polega na tym, że w pierwszym kroku zbiera się od studentów ich preferencje, a potem na ich podstawie przydziela studentów do grup. Język do opisu preferencji pozwala na definiowanie podzbiorów grup w kolejności od najbardziej do najmniej pożądanych, a także wskazanie, na które zajęcia student nie zamierza chodzić. Na przykład:

- najbardziej chcę chodzić na {wykład z przedmiotu A do grupy 1, ćwiczenia z przedmiotu A do grupy 1, laboratorium z przedmiotu B do grupy 3},
- w drugiej kolejności mój pożądany grafik to {wykład z przedmiotu A w grupie 3, ćwiczenia z przedmiotu A w grupie 3, laboratorium z przedmiotu B w grupie 2},
- nie będę uczęszczać na wykład z przedmiotu B (powtarzam przedmiot i chcę chodzić tylko na laboratorium).

Zadaniem programu jest znalezienie takiego przydziału studentów do grup, w którym studenci będą mieli **jak najmniej konfliktów** (zajęć odbywających się w tym samym czasie) i będzie spełnionych **najwięcej preferencji** (wszystkie grupy, na które zapisany jest student, należą do podzbioru odpowiadającego preferencji).

Pierwszy opracowany algorytm ustawiał studentów w losowej kolejności, a następnie w tej kolejności przypisywał ich do grup. Dla każdego studenta sprawdzano wszystkie możliwe przypisania i wybierano najlepsze z nich. Takie rozwiązanie mogłoby wydawać się racjonalne, ale szybko okazało się, że część studentów (ta, która nie miała szczęścia w losowaniu) była z niego niezadowolona, gdyż ich plany miały wiele kolizji.

Żeby móc poszukiwać rozwiązania optymalnego, trzeba najpierw opisać problem matematycznie i podać **funkcję celu**, która będzie optymalizowana – dokładnie lub w sposób przybliżony. Wprowadziliśmy porządek na przypisaniach i mówimy, że jeśli w planie A jest mniej kolizji niż w planie B, to plan A **jest lepszy** od planu B. Jeśli plany A i B mają tę samą liczbę kolizji, to lepszy jest plan, który spełnia więcej preferencji. Wartość preferencji pojedynczego studenta definiujemy w taki sposób, że spełnienie najgorszej preferencji jest warte 0,1, spełnienie najlepszej warte jest 1,0, a pośrednie preferencje są rozłożone liniowo.

Każdemu przypisaniu odpowiada **kara** w postaci pojedynczej liczby wymiernej, której część całkowita to liczba konfliktów, a część ułamkowa to jeden minus suma preferencji przeskalowana tak, aby wynosiła jeden w przypadku spełnienia wszystkich najlepszych. Spośród wszystkich przypisań poszukujemy takiego, które będzie miało najniższą karę.

Idealnie byłoby, gdyby rozwiązanie miało karę 0, bo wtedy nie byłoby konfliktów, a wszyscy dostaliby to, co chcieli. Jak jednak znaleźć przypisanie o najniższej karze?

Nie można po prostu przeglądać wszystkich możliwych przypisań, gdyż ich liczba jest wykładnicza, trzeba wymyślić coś sprytniejszego. Można udowodnić, że problem jest **NP-trudny**, więc najprawdopodobniej nie istnieje jego wielomianowe rozwiązanie. Można by spróbować poszukiwać dobrych rozwiązań „chaotycznie", losując dużą ich liczbę i wybierając najlepsze z wylosowanych.

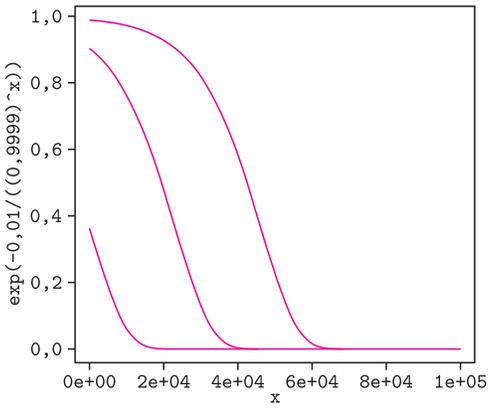
Takie rozwiązanie w przypadku niektórych problemów działa dobrze, w przypadku naszego, niestety, się nie sprawdza. Można spróbować innego algorytmu.

```
najlepsze_przypisanie = wylosuj_przypisanie_startowe();
while masz_jeszcze_czas() do
    potencjalne_przypisanie = zmodyfikuj_losowo(najlepsze_przypisanie);
    if kara(potencjalne_przypisanie) < kara(najlepsze_przypisanie) then
        najlepsze_przypisanie = potencjalne_przypisanie;
return najlepsze_przypisanie;
```

Program ten implementuje odmianę algorytmu **wspinaczki na szczyt** (ang. *hill climbing*), w którym poprzez niewielkie zmiany rozwiązania znajdujemy dobre rozwiązanie. Oczywiście, to rozwiązanie wcale nie musi być najlepsze. Tak napisany program może działać wolno, gdyż dla każdego z potencjalnych przypisań musi policzyć karę, a naturalna implementacja funkcji kara wymaga przejrzenia zapisów wszystkich studentów. Można go usprawnić w następujący sposób:

```
najlepsze_przypisanie = wylosuj_przypisanie_startowe();
while masz_jeszcze_czas() do
    mała_modyfikacja = wylosuj_małą_modyfikację(najlepsze_przypisanie);
    if różnica_kar(najlepsze_przypisanie, mała_modyfikacja) < 0 then
        najlepsze_przypisanie.zaaplikuj_zmianę(mała_modyfikacja);
return najlepsze_przypisanie;
```

Teraz program nie musi liczyć kary w każdym przebiegu pętli, wystarczy, że potrafi policzyć, czy mała modyfikacja poprawia rozwiązanie. Ale jak mogłaby wyglądać mała modyfikacja przypisania? Naturalnym pomysłem byłoby uznanie, że polega ona na przepisaniu studenta z jednej grupy do drugiej, pod warunkiem wszakże, iż nie powoduje to przekroczenia limitu miejsc. W przypadku takiego rozwiązania funkcja różnica_kar() jest prosta i szybka (wymaga sprawdzenia jedynie przypisania pojedynczego studenta), ale w praktyce rozwiązanie to się nie sprawdzi. Dlaczego? Bardzo wiele przedmiotów ma ścisłe limity (np. 90 studentów i 6 grup po 15 osób) i żadne pojedyncze przepisanie bez przekroczenia limitu nie jest możliwe. Można rozbudować małe modyfikacje tak, aby dopuszczały wymiany dwóch osób, albo dopuścić możliwość chwilowego przekroczenia limitów. Zdecydowaliśmy się na to drugie rozwiązanie.



Zwiększyliśmy karę o składnik równy liczbie przekroczonych miejsc pomnożonej przez niedużą liczbę naturalną, aby przekroczenie miejsca było ważniejsze od konfliktu studenta, który ma wagę jeden. Zmodyfikowaliśmy też program, aby dopuszczał czasem gorsze rozwiązania.

```
najlepsze_przypisanie = wylosuj_przypisanie_startowe();
while masz_jeszcze_czas() do
    mala_modyfikacja = wylosuj_małą_modyfikację(najlepsze_przypisanie);
    delta_kary = różnica_kar(najlepsze_przypisanie, mała_modyfikacja);
    if delta_kary < 0 or dopuszczamy_gorsze(delta_kary, czas_do_końca) then
        najlepsze_przypisanie.zaaplikuj_zmianę(mała_modyfikacja);
return najlepsze_przypisanie;
```

Decyzję o tym, czy dopuścić gorsze rozwiązanie, program podejmuje w oparciu o schemat **symulowanego wyżarzania** (ang. *simulated annealing*), którego sposób działania przypomina zjawisko wyżarzania w metalurgii. Załóżmy, że mamy temperaturę, która spada od temperatury początkowej T_0 do zera. W i-tym kroku temperatura T wynosi $(1-\varepsilon)^i \cdot T_0$. Prawdopodobieństwo zaakceptowania gorszego rozwiązania jest zależne od temperatury i od tego o ile rozwiązanie jest gorsze od poprzedniego i wynosi $e^{(-\text{delta\_kary}/T)}$. Widać zatem, że szansa na zaakceptowanie gorszego rozwiązania maleje wraz ze spadkiem temperatury, jak również wraz ze wzrostem delta_kary. Wykres pokazuje, jak wraz z czasem maleje prawdopodobieństwo zaakceptowania gorszych rozwiązań, dla różnych delt (1; 0,1; 0,01).

Mogłoby się wydawać, iż jeśli pozwoli się programowi na przekraczanie limitów, to będzie on to robił nagminnie. Okazuje się, że praktycznie się to nie zdarza. Natomiast z naszego punktu widzenia w łatwy sposób pozwoliło to na rozszerzenie programu o dodatkowe warunki (np. wszyscy chodzący do grupy A1 jednych zajęć muszą również chodzić do grupy B1 innych zajęć). Program zyskał dodatkową swobodę, ponieważ może przeglądać większą przestrzeń rozwiązań, a to ma korzystny wpływ na ich jakość.

Pomijając dodatkowe szczegóły, tak właśnie działa w USOS **rejestracja do grup**. W praktyce, w stosunku do poprzedniego rozwiązania, liczba osób z konfliktami zmniejszyła się o połowę, a ręczne analizy osób z największą liczbą konfliktów wykazały, że nie istnieją dla nich rozwiązania lepsze, gdyż konflikty są wymuszone przez nakładające się zajęcia.

Warto także zwrócić uwagę na nieinformatyczne aspekty rozwiązania. Rejestracja do grup jest poprzedzona wyborem przedmiotów. Studenci wskazują pożądane przedmioty, a władze wydziału, widząc, jak rozkładają się preferencje, mogą jeszcze przed kolejnym etapem dostosować do nich liczbę grup. W drugim kroku studenci ponownie pytani są o preferencje, ale już tylko w odniesieniu do grup zajęć przedmiotów, na które zostali przyjęci. To zatem studenci decydują o kształcie swoich studiów. A dzięki zastosowaniu wyżarzania zamiast rejestracji „kto pierwszy ten lepszy" uczelniane serwery nie są narażone na gwałtowny wzrost obciążenia. Warto znać się na algorytmice.

A jeśli jakiś wydział mimo wszystko chce zastosować zasadę „kto pierwszy ten lepszy"? O tym, jak mimo wszystko zabezpieczyć serwery w sytuacji wzrostu obciążenia, będzie mowa w jednym z kolejnych odcinków serii.