Wszystkie problemy są albo trudne obliczeniowo, albo otwarte z punktu widzenia złożoności obliczeniowej, zatem zastosowanie metaheurystyki do ich rozwiązania jest uzasadnione. Wszystkie problemy mają tę zaletę, że bardzo łatwo można dla nich wygenerować jakiekolwiek rozwiązanie dopuszczalne (choć bardzo słabe), zatem nie zajdzie przypadek, że metaheurystyka nie zwróci żadnego rozwiązania. Chodzi jednak o to, żeby metaheurystyka w trakcie swojej pracy poprawiała rozwiązanie, tzn. zastępowała już posiadane rozwiązanie dopuszczalne lepszym rozwiązaniem dopuszczalnym, i żeby w efekcie wygenerowała rozwiązanie końcowe jak najbliższe rozwiązaniu optymalnemu. Wartość funkcji celu rozwiązania optymalnego można w przybliżeniu oszacować, jeśli stosuje się pewien schemat generowania instancji, o tym niżej. Znając przybliżoną wartość funkcji celu mogą Państwo ocenić, jak dobre są rozwiązania generowane przez metaheurystykę (im bliższa optymalnej wartości jest wartość funkcji celu wygenerowanego rozwiązania, tym lepiej) i jeśli niezbyt dobre, należy metaheurystykę poprawiać.

Instancje, które posłużą Państwu do testowania metaheurystyk, będą (w większości) odpowiadały sensownemu rozwiązaniu problemu; przykładowo, instancje problemów odwołujących się do PDP pozwolą na skonstruowanie mapy o liczbie cięć znacznie przekraczającej 1 czy 2. Proszę jednak zwrócić uwagę na to, że w sformułowaniach problemów dopuszczone są w miarę dowolne instancje na wejściu, które mogą być nawet całkowicie losowo dobranymi wartościami i dla których nawet najlepszy algorytm nie zdołałby stworzyć rozwiązania o solidnych rozmiarach. Metaheurystyka ma akceptować na wejściu wszystko, co pasuje do opisu instancji problemu i nawet dla absurdalnych wartości ma próbować wygenerować rozwiązanie (nie może się zawiesić).

W problemach I, II i III będziemy się spodziewać zazwyczaj instancji odpowiadającej rzeczywistej mapie, zawierającej trochę błędów. (Podejście mapowania z użyciem enzymów restrykcyjnych oraz problem PDP omówione są w ramach wykładu o mapowaniu z przedmiotu „Algorytmy kombinatoryczne w bioinformatyce”, slajdy wraz z materiałami uzupełniającymi na stronie https://www.cs.put.poznan.pl/mkasprzak/akb/akb.html). Przykład instancji, która odpowiada pewnej mapie i nie zawiera żadnych błędów:

D = {3, 4, 5, 8, 9, 13, 14, 17, 18, 21}

dla której rozwiązaniem jest (we wszystkich trzech problemach):

P = {0, 3, 8, 17, 21},

czyli po zwizualizowaniu na osi liczbowej mamy 0 i 21 jako końce cząsteczki, a cięcia zlokalizowane w punktach 3, 8, 17, co daje odstępy pomiędzy tymi punktami na osi, po kolei, 3, 5, 9, 4. Jednak w Państwa testach będą przeważały instancje z błędami (błędy rozumiane jako odchylenie od idealnej instancji odpowiadającej jakiemuś rozwiązaniu). Weźmy np. powyższy multizbiór D, w którym zastąpimy wartość 8 wartością 9. W takiej sytuacji (dla problemów I i II) nie da się odtworzyć rozwiązania jak wyżej, ale jakieś się da. W przypadku problemu I multizbiór D ma zawierać się w całości w, nazwijmy to, „produkcie” P, czyli dla P przyjętego jako rozwiązanie obliczamy odstępy pomiędzy każdą parą liczb w P i w tak uzyskanym multizbiorze ma zawierać się D (czyli skoro D ma dwa wystąpienia liczby 9, to co najmniej dwa wystąpienia 9 muszą być w produkcie P). Wygenerowanie rozwiązania dopuszczalnego, lecz bardzo słabego, jest zawsze możliwe i bardzo proste. Wystarczy np. odłożyć na osi liczbowej po kolei wszystkie elementy D i zapisać to jako P, wtedy P = {0, 3, 7, 12, 21, 30, 43, 57, 74, 92, 113}, gdzie m=11. Jednak jest to duża wartość m, a mamy ją w problemie minimalizować, jednym z optymalnych rozwiązań dla tak zmienionego D jest P = {0, 3, 8, 9, 17, 21} z m=6. Jeden błąd w D przekłada się tu na zwiększenie wartości funkcji celu rozwiązania optymalnego o 1 i ta zasada w przybliżeniu sprawdza się dla bardzo małej liczby błędów, także w innych problemach. Błędem może być nie tylko zamiana wartości w D na inną, w problemie I też dodanie elementu do D. Każdy dodany element to (dla małej liczby błędów) zwiększenie m o 1. Natomiast usunięcie elementu z D nie wpływa dla małej liczby takich błędów na m, trudno tu przeliczyć liczbę wprowadzonych błędów na oczekiwaną wartość funkcji celu, dlatego lepiej takiego błędu nie generować. W problemie II błędem może być zamiana wartości w D na inną albo usunięcie

wartości z D. Rozwiązaniem jest tu takie P, że jego produkt zawiera się w całości w D i P jest jak największe. Dla D ze zmienioną 8 na 9 rozwiązaniem dopuszczalnym, choć bardzo słabym, może być np. P = {0, 3} (albo jakakolwiek inna para liczb, których różnica daje liczbę obecną w D), rozwiązaniem optymalnym jest natomiast P = {0, 3, 17, 21} z m=4. Tu też jeden błąd przełożył się na różnicę 1 do wartości optymalnej sprzed wprowadzenia tego błędu. Usunięcie wartości z D (np. liczby 8) w problemie II daje podobny efekt. W problemie III błędem może być zamiana wartości w D na inną. Rozwiązaniem optymalnym dla D zmodyfikowanego jak wcześniej jest zbiór P = {0, 3, 8, 17, 21}, gdzie różnica do produktu P (czyli do D') wynosi 1 element, czyli wartość funkcji celu wynosi 9. I tutaj jeden błąd przekłada się na różnicę 1 do optymalnej wartości funkcji celu sprzed wprowadzenia błędu (którą jest |D|=10). Rozwiązaniem dopuszczalnym w problemie III jest jakikolwiek zbiór P mający m elementów.

Problemy IV i V mają na wejściu macierz binarną, w ogólności dowolną, jednak w testach najczęściej będą Państwo podawać macierz odpowiadającą jakiejś w miarę rzeczywistej instancji. Przykład bez błędów (taki sam jak na slajdach z wykładu o mapowaniu z przedmiotu „Algorytmy kombinatoryczne w bioinformatyce"; tam też omówione zostało podejście mapowania przez hybrydyzację):

0 1 0 1
1 1 1 1
1 0 0 1
0 0 1 0

i rozwiązanie idealnie pasujące do tej instancji (w obu problemach), czyli uporządkowanie kolumn, tu podane w postaci indeksów kolumn macierzy wejściowej: (2, 4, 1, 3). W tym przypadku nic z macierzą nie trzeba robić, żeby własność consecutive 1s w wierszach była osiągnięta, czyli że po przeszeregowaniu kolumn jedynki we wszystkich wierszach macierzy będą pogrupowane w nieprzerwane bloki. Jednak w ogólności macierz na wejściu będzie zawierała błędy, błędem w obu problemach może być zamiana 0 na 1 lub 1 na 0, ale proszę zwrócić uwagę na to, że nie w każdym polu macierzy zamiana wprowadzi rzeczywiste błędy. Jeśli zmienione zostanie pole na granicy ciągu zer i jedynek w danym wierszu (po optymalnym przeszeregowaniu kolumn), to nie wpłynie to na własność consecutive 1s (np. zmiana drugiej jedynki w drugim wierszu na zero albo dowolnej jedynki w pozostałych wierszach na zero nadal podtrzymuje własność consecutive 1s dla tej macierzy, uszeregowanie optymalne kolumn jak wyżej). Zmiana w innym miejscu też nie musi wpłynąć na tę własność (jeszcze inne uszeregowanie kolumn może pokazać, że nadal w każdym wierszu są jedynki pod rząd), ale w przybliżeniu przyjmijmy, że wpływa i w przybliżeniu (dla małej liczby błędów) można uznać, że jedna zmiana to zmiana szacowanej optymalnej wartości funkcji celu o 1. Przykładowo, weźmy macierz z jednym błędem:

0 1 0 1
1 1 1 0
1 0 0 1
0 0 1 0

gdzie rozwiązaniem optymalnym (dla obu problemów) może być uszeregowanie np. (2, 4, 1, 3) albo (4, 1, 2, 3). W problemie IV minimalizowaną funkcją celu jest liczba pól, których wartość należy zmienić celem doprowadzenia do własności consecutive 1s (wszystko jedno, czy 0 na 1 czy 1 na 0), dla obu tych rozwiązań jest to jedno pole (czyli optymalna wartość funkcji celu tutaj wynosi 1). W problemie V minimalizowaną funkcją celu jest liczba kolumn, które należy usunąć w celu osiągnięcia tej własności (nie musi to oczywiście być kolumna z błędem wprowadzonym na etapie generowania instancji, gdyż tego miejsca algorytm nie zna), tutaj w obu rozwiązaniach trzeba usunąć jedną kolumnę (czyli optymalna wartość funkcji celu to 1). W przypadku problemu V, jeśli chcielibyśmy, żeby jeden błąd wprowadzony do instancji przekładał się w przybliżeniu na wzrost optymalnej wartości funkcji celu o 1, powinniśmy plasować błędy w różnych kolumnach. Ale i wtedy optymalne

rozwiązanie może nas zaskoczyć, bo np. wprowadziliśmy błędy w dwóch polach w dwóch różnych kolumnach, a lepszym wyjściem niż usuwanie tych dwóch kolumn będzie usunięcie jednej innej.

W problemach IV i V także da się wygenerować w bardzo prosty sposób słabe rozwiązanie dopuszczalne dla dowolnej instancji. W problemie IV można to zrobić na przykład w ten sposób, że pomijamy sprawdzanie różnych uszeregowań kolumn i zliczamy po wszystkich wierszach macierzy liczbę przeszkadzających zer lub jedynek. Dla macierzy z błędem powyżej dałoby to liczbę 2 (usunięcie ostatnich jedynek w wierszach pierwszym i trzecim daje ciągi jedynek pod rząd). W problemie V możemy zliczać w ten sposób liczbę przeszkadzających kolumn (jedna kolumna dla powyższego przykładu).

W problemach VI i VII na wejściu w ogólności dopuszczamy dowolną macierz m×n wypełnioną znakami ze zbioru {A, C, G, T}. (Więcej o problemie dopasowania wielu sekwencji na slajdach i w materiałach uzupełniających do wykładu o dopasowaniu sekwencji z przedmiotu „Algorytmy kombinatoryczne w bioinformatyce”.) Jednak im bardziej podobne do siebie te sekwencje, tym sensowniejsze (krótsze) rozwiązanie. Najkrótsze rozwiązanie osiągniemy dla zbioru identycznych sekwencji, nie o to nam chodzi, chcemy różne sekwencje, które sprawią pewną trudność metaheurystyce, lecz dość do siebie podobne. Przykładowo, dla sekwencji na wejściu AATCCG, ATACCG, AAACGG dostaniemy rozwiązanie optymalne dla problemu VII np. takie:

```
AAT_CC_G
A_TACC_G
AA_AC_GG
```

z liczbą kolumn dopasowania równą 8 (to wartość funkcji celu). Tutaj można zastosować następujący schemat generowania instancji, co do której możemy oszacować (w przybliżeniu), jakie rozwiązanie optymalne posiada. Generujemy losowo zerojedynkową macierz m×d, gdzie d to liczba kolumn dopasowania w taki sposób, żeby w każdym wierszu było dokładnie n jedynek. Macierz ta odpowiada rozwiązaniu (dopasowaniu globalnemu sekwencji), gdzie 1 to pozycja znaku, a 0 to pozycja spacji. Dodatkowo generujemy losową sekwencję o długości d nad alfabetem {A, C, G, T}. Sekwencje wejściowe o długości n uzyskujemy przez wzięcie z tej sekwencji o długości d tych znaków, dla których w macierzy zerojedynkowej wystąpiła 1. Przykładowo, instancję powyższą uzyskalibyśmy z macierzy zerojedynkowej o postaci

```
1 1 1 0 1 1 0 1
1 0 1 1 1 1 0 1
1 1 0 1 1 0 1 1
```

i sekwencji znaków AATACCGG. Wiemy wtedy, że na pewno istnieje rozwiązanie o długości d, choć może istnieć rozwiązanie lepsze. Błędem może być tu zmiana znaku na inny (wtedy zazwyczaj dopasowanie wydłuży się o 1 kolumnę), dodanie znaku nie wchodzi w grę, bo zmienia długość sekwencji względem innych (wszystkie sekwencje muszą być tej samej długości). Bardzo słabe rozwiązanie dopuszczalne można osiągnąć np. tak:

```
AATCCG______________
_______ATACCG_______
______________AAACGG
```

lub trochę sprytniej np. tak:

```
AATCCG_____
A_T___ACCG_
AA____AC_GG
```

Dla tych samych przykładowych sekwencji i problemu VI rozwiązanie optymalne mogłoby wyglądać tak samo jak dla problemu VII, gdyż długość dopasowania nie przekracza 12 i w żadnej kolumnie nie

ma niezgodnych znaków. Żeby zaobserwować niezerową wartość funkcji celu, trzeba dać większą liczbę sekwencji i/lub bardziej je zróżnicować, np. jak poniżej:

```
A____ACT
A_TGG___
___GGAC_
_TT___CC
```

W tym optymalnym dopasowaniu o długości ≤ 2n=8 jest jedna kolumna z niezgodnymi znakami (wartość funkcji celu wynosi 1). Instancję o znanej optymalnej wartości funkcji celu można wygenerować np. w podobny sposób jak powyżej, gdzie macierz zerojedynkowa przyjmie długość 2n. Jeśli odwzorujemy losową sekwencję o długości 2n na pozycje w macierzy zerojedynkowej, otrzymamy zbiór sekwencji o optymalnym dopasowaniu bez błędów. Wtedy każdy wprowadzony do instancji błąd, czyli zmiana jednego znaku na inny, może spowodować zwiększenie optymalnej wartości funkcji celu o 1. Rozwiązaniem dopuszczalnym jest każde dopasowanie nie dłuższe niż 2n, do tych gorszych jakościowo można zaliczyć np. takie:

```
AACT____
____ATGG
GGAC____
____TTCC
```

gdzie wartość funkcji celu wynosi 7.

Generator instancji, niezależnie od problemu, powinien działać w ten sposób, żeby użytkownik mógł oszacować (w przybliżeniu) optymalną wartość funkcji celu dla danej generowanej instancji, żeby potem porównać z nią jakość rozwiązania osiągniętego przez metaheurystykę. Należy to osiągnąć w ten sposób, że generator zaczyna od wygenerowania rozwiązania problemu, a potem przekształca je w instancję wejściową problemu. Wtedy wiadomo, że (o ile nie wprowadzono błędów) dla takiej instancji istnieje rozwiązanie, od którego zaczęliśmy. Dla problemów I, II i III należy więc wygenerować zbiór P, potem uzyskać jego produkt i będzie to D w wersji bez błędów. Dla problemów IV i V należy wygenerować macierz zerojedynkową, w której jedynki będą występować w blokach pod rząd w wierszach, a instancję wejściową bez błędów otrzymujemy przez przesortowanie kolumn tej macierzy. Dla problemów VI i VII generujemy instancję jak wyżej. Dodanie błędów (więcej o nich było przy poszczególnych problemach) zmienia wartość funkcji celu, ale relację 1-1 (jeden błąd – zmiana o jeden optymalnej wartości funkcji celu) można w przybliżeniu założyć tylko dla małej liczby błędów. Jednak błędów należy też wprowadzać do instancji więcej, tylko wtedy trzeba być świadomym, że rozwiązanie optymalne może być lepsze od założonego. Rozwiązanie optymalne może być lepsze od założonego także dla małej liczby błędów, a nawet przy braku błędów. Metaheurystyka nie może nic wiedzieć o rozwiązaniu początkowym ani o oszacowanej wartości funkcji celu, żadna informacja nie może przepływać z generatora do metaheurystyki poza samą instancją, ale obowiązkowo przesortowaną względem oryginalnego rozwiązania (problemy VI i VII nie wymagają sortowania). Oszacowana wartość funkcji celu ma posłużyć do oceny jakości działania metaheurystyki w sprawozdaniu.

Generator powinien umożliwiać wprowadzanie parametrów określających stopień trudności instancji. Podstawowymi takimi parametrami są rozmiar instancji (k dla problemów I, II i III, m i n dla pozostałych) i liczba wprowadzonych błędów. Ponadto:
— dla problemów I, II i III na stopień trudności mogą wpłynąć wartości liczbowe w multizbiorze D, bardziej lub mniej unikalne;
— dla problemów IV i V może to być stopień wypełnienia wierszy jedynkami, najtrudniejsze instancje to zwykle te o wypełnieniu zbliżonym do połowy szerokości wiersza; proszę pamiętać, żeby nie generować wierszy wypełnionych w całości jedynkami albo takich z jedną lub wcale, gdyż taki wiersz nie stanowi żadnego wyzwania w uszeregowaniu (każde jest równie dobre) i w istocie zmniejsza instancję (o 1 na każdy taki wiersz); należy też rozkładać ciągi jedynek w miarę równomiernie w

macierzy, żeby raczej wszystkie kolumny były pokryte jakimiś ciągami i nie okazało się, że np. prawa część macierzy zawiera same zera, to też praktycznie zmniejsza instancję (o 1 na każdą taką pustą kolumnę);
— dla problemów VI i VII może to być rozmiar alfabetu, który w celu ułatwienia instancji można ograniczyć do np. dwóch tylko znaków {A, C}; także długość dopasowania d w schemacie generowania jak powyżej.

Zachęcam do zastanowienia się, czy są inne możliwości wpłynięcia na stopień trudności instancji.