

WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI  
POLITECHNIKI WROCŁAWSKIEJ

WYBRANE PROBLEMY  
ODPORNEJ OPTYMALIZACJI DYSKRETNEJ  
Z MOŻLIWOŚCIĄ MODYFIKACJI

TOMASZ STRZAŁKA

Praca magisterska napisana  
pod kierunkiem  
dr hab. Pawła Zielińskiego, prof. PWr



Politechnika  
Wrocławska

WROCŁAW 2016



Podziękowania chciałbym skierować do mojego promotora,  
dr hab. Pawła Zielińskiego, prof. nadzw. Politechniki Wrocławskiej,  
za nakład włożonej pracy w korektę tekstu,  
poświęcony na konsultacjach czas oraz zaangażowanie,  
pływające z chęci pomocy w przygotowaniu jak najlepszej pracy dyplomowej.

X,  
X  
X  
X.





# Spis treści

<b>1 Wstęp</b>	<b>1</b>
<b>2 Podstawy</b>	<b>3</b>
2.1 Grafy a drzewa rozpinające . . . . .	3
2.1.1 Drzewo rozpinające . . . . .	4
2.2 Problem minimalnego drzewa rozpinającego . . . . .	5
2.3 Znane algorytmy rozwiązujące problem MST . . . . .	7
2.3.1 Algorytm Kruskala . . . . .	7
2.3.2 Algorytm Prima . . . . .	9
2.4 Podsumowanie rozdziału . . . . .	9
<b>3 Odporna optymalizacja</b>	<b>11</b>
3.1 Scenariusze dyskretne a ciągłe . . . . .	11
3.2 Problemy minimaksowe . . . . .	12
3.2.1 Przypadek ciągły . . . . .	12
3.2.2 Przypadek dyskretny . . . . .	13
3.2.3 Sposoby aproksymacji problemu . . . . .	15
3.3 Problemy minimaksowe ze względnią funkcją optymalności . . . . .	17
3.3.1 Przypadek dyskretny . . . . .	17
3.3.2 Przypadek ciągły . . . . .	20
3.4 Optymalizacja z możliwością poprawy . . . . .	22
3.4.1 Problem przyrostowy . . . . .	22
3.4.2 Zagadnienia oparte na problemie INCREMENTAL . . . . .	25
3.5 Podsumowanie rozdziału . . . . .	28
<b>4 Problemy programowania liniowego</b>	<b>29</b>
4.1 Programowanie liniowe . . . . .	29
4.1.1 Problem minimalnego drzewa rozpinającego . . . . .	30
4.1.2 Naiwne podejście . . . . .	31
4.1.3 Przepływy . . . . .	31
4.1.4 Przepływy z wymuszoną unimodularnością . . . . .	34
4.2 Odporne minimalne drzewo rozpinające . . . . .	35
4.2.1 Relaksacja Lagrange'a . . . . .	38
4.3 Podsumowanie rozdziału . . . . .	42
<b>5 Algorytm odpornej optymalizacji regeneracyjnej</b>	<b>45</b>
5.1 Konstrukcja algorytmu i warunki optymalności . . . . .	45
5.2 Struktura drzewa i koszty krawędzi . . . . .	47
5.3 Binarne poszukiwanie rozwiązania . . . . .	51
5.4 Pełny algorytm przeszukiwania binarnego . . . . .	53
5.4.1 Lepiej, szybciej . . . . .	55
5.5 Analiza poprawności . . . . .	58
5.6 Podsumowanie rozdziału . . . . .	60

<b>6 Odporna optymalizacja regeneracyjna w sąsiedztwie</b>	<b>63</b>
6.1 Algorytm zachłanny . . . . .	63
6.1.1 Sąsiedztwo . . . . .	63
6.1.2 Wady algorytmu zachłannego . . . . .	65
6.2 Symulowane wyżarzanie . . . . .	66
6.3 Przeszukiwanie z listą Tabu . . . . .	66
6.3.1 Sąsiedztwo . . . . .	67
6.3.2 Losowe drzewo rozpinające i strategia dywersyfikacji . . . . .	68
6.3.3 Lista ruchów zakazanych i kryterium końca . . . . .	69
6.3.4 Podsumowanie rozdziału . . . . .	69
<b>7 Wyniki eksperymentalne</b>	<b>71</b>
7.0.1 Środowisko testowe . . . . .	71
7.1 Gęstość grafu . . . . .	71
7.1.1 Metodyka i dane . . . . .	71
7.1.2 Wyniki i wykresy zależności . . . . .	71
<b>8 Zakończenie</b>	<b>73</b>
<b>A Biblioteka: Incremental MST</b>	<b>79</b>





# Wstęp

---

xxx

Część implementacyjna pracy magisterskiej została napisana w języku C++ zgodnie ze standardem ISO-IEC 14882:2014 z wykorzystaniem środowiska programistycznego ECLIPSE w wersji 4.5.0 (MARS), debuggerów GDB oraz VALGRIND (w wersji 3.10.1). Tekst poniższej pracy został złożony w systemie L<sup>A</sup>T<sub>E</sub>X , między innymi z wykorzystaniem podstawowych pakietów do reprezentacji kodu programistycznego: LISTINGS oraz ALGORITHM2E. Do składu wszelkich ilustracji użyto aplikacji internetowej DRAW.IO<sup>1</sup> oraz programu INKSCAPE w wersji 0.91. Do wygenerowania przedstawionych w pracy wykresów dwu- i trójwymiarowych zastosowano programy: OCTAVE (w wersji 3.8.2) oraz CROPPDF (do korekty otrzymanych wykresów). Całość została skompilowana pod kontrolą systemu LINUX UBUNTU 15.04 (VIVID VERVET).

- Trzeba zrobić wstęp o programowaniu liniowym.
  - Klasa problemów 0-1 ( $\mathcal{C}$ ): kombinatroyczne (ściezka, drzewo), plecak, pokrycie zbioru
  - opowiedzieć o unimodularności, relaksacji Lagrangiana, warunkach komplementarności
- Wstęp o odpornej optymalizacji.

---

<sup>1</sup> Adres strony internetowej: <https://www.draw.io/>.



# Podstawy

---

Aby w pełni móc zrozumieć przekrój problemów jaki poruszamy, niezbędne jest uprzednie zapoznanie się z podstawami, na których są one oparte. Jako, że będziemy rozważali problemy odpornej optymalizacji na przykładzie minimalnych drzew rozpinających, w tym rozdziale zajmiemy się przedstawieniem podstawowych definicji dotyczących zagadnień bezpośrednio nimi związanych. Przytoczymy definicje samego **minimalnego drzewa rozpinającego** (ang. *minimum spanning tree*), **grafów**, terminów im pochodnych oraz sposobów ich reprezentacji. Na samym końcu dokonamy przeglądu algorytmów do rozwiązywania problemu minimalnego drzewa rozpinającego (dalej będziemy często wykorzystywać skrót od ich angielskiej nazwy — **MST**) oraz przyjrzymy się właściwościom tych specyficznych struktur grafowych, na podstawie których wspomniane algorytmy funkcjonują i zwracają poprawne rozwiązania.

---

## 2.1 Grafy a drzewa rozpinające

Chcąc mówić o problemie **minimalnego drzewa rozpinającego** musimy najpierw przypomnieć sobie definicję podstawowych zagadnień związanych z grafami. **Grafem**  $G = (V, E)$  będziemy zatem nazywać zbiór punktów  $v_i \in V$  opcjonalnie ze sobą połączonych krawędziami  $e_{ij} \in E$ , gdzie przyjmujemy następującą definicję krawędzi:  $e_{ij} \equiv v_i \xrightarrow{1} v_j$ , która wyraża fakt istnienia łuku (krawędzi) pomiędzy wierzchołkami (punktami) grafu  $v_i$  oraz  $v_j$ , które są nim bezpośrednio połączone<sup>1</sup>. Same zaś wierzchołki będziemy numerować za pomocą indeksów  $i \in \{1, \dots, |V|\}$ , gdzie  $|V| = n$  i wyraża **moc** (liczbę elementów) zbioru wierzchołków grafu. Analogicznie będziemy przyjmować, że  $|E| = m$ .

Przy omawianiu problemu **minimalnego drzewa rozpinającego** będziemy rozważać tylko specyficzną rodzinę grafów: grafów nieskierowanych (ang. *undirected graphs*), które charakteryzują się tym, że fakt istnienia krawędzi  $e_{ij}$  pomiędzy wierzchołkami grafu  $v_i$  oraz  $v_j$  nie przesąduje o **kierunku** danego łuku — w przypadku grafu nieskierowanego nasza definicja łuku tak naprawdę powinna wyglądać następująco:  $e_{ij} \equiv v_i \xrightarrow{1} v_j \wedge v_j \xrightarrow{1} v_i$  (zatem w grafie nieskierowanym  $e_{ij} \equiv e_{ji}$ , zaś stosowana w zapisie kolejność indeksów jest tylko umowna). Będziemy wymiennie, w zależności od sytuacji, stosować aż cztery rodzaje oznaczeń krawędzi  $e$ :

- $e_{ij}$ , gdy będziemy chcieli podkreślić fakt wystąpienia krawędzi pomiędzy wierzchołkami grafu o indeksach  $i$  oraz  $j$ ,
- $(i, j)$ , gdy będziemy chcieli położyć szczególny nacisk na połączone ze sobą wierzchołki grafu,
- $e_i$ , gdzie w tym przypadku  $i \in \{1, \dots, |E|\}$  oznacza indeks krawędzi w grafie, oraz  $v_i \xrightarrow{1} v_j$ , gdy będziemy chcieli albo podkreślić rozpatrywany przez nas „kierunek” krawędzi nieskierowanej.

Dodatkowo każda krawędź będzie posiadała dodatkowy atrybut, zwany przez nas dalej **kosztem** (lub **wagą**) krawędzi. Jako że bierzemy pod uwagę tylko grafy nieskierowane, koszty krawędzi  $e_{ij}$  oraz  $e_{ji}$  z definicji są takie same. Wagi krawędzi  $e_{ij}$  będziemy oznaczać jako  $c_{e_{ij}}$  (ang. *cost*) lub jako  $c_e$  (w przypadku, gdy nie

---

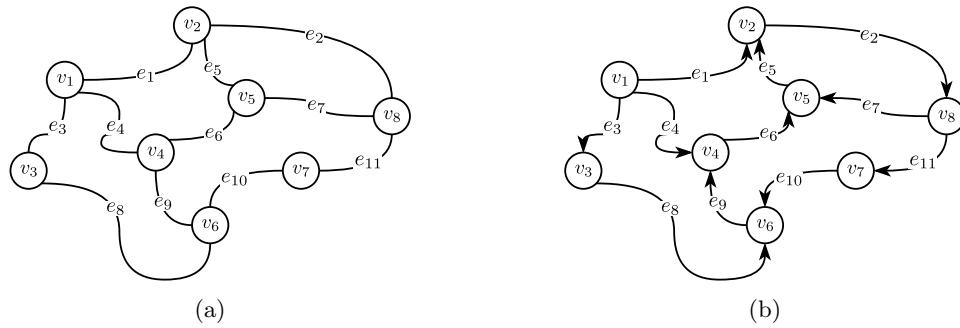
<sup>1</sup> Wyrażeniem  $v_i \xrightarrow{k} v_j$  często też będziemy chcieli zaznaczać fakt istnienia **ścieżki** pomiędzy wierzchołkami  $v_i$  a  $v_j$ , gdzie  $k$  oznaczać będzie dokładną liczbę krawędzi, która znajduje się na takiej ścieżce między wymienionymi wierzchołkami (jeśli  $k = *$ , wtedy ich liczba może być dowolna, dla  $k = 1$  będziemy tę liczbę pomijać). **Ścieżką** pomiędzy wierzchołkami  $v_i$  oraz  $v_j$  będziemy zaś nazywać następujący ciąg krawędzi:  $\left\{ v_i \xrightarrow{1} v_{a_1}, v_{a_1} \xrightarrow{1} v_{a_2}, \dots, v_{a_b} \xrightarrow{1} v_{a_{b+1}}, v_{a_{b+1}} \xrightarrow{1} v_j \right\}$ .



będą nas interesowały wierzchołki które dana krawędź łączy), lub jako  $c_k$  (gdy będziemy chcieli odwołać się do kosztu krawędzi  $e$  o indeksie  $k$ ). Pełną zatem definicją krawędzi w grafie nieskierowanym przedstawiła równość:

$$e_{ij} \equiv v_i \xrightarrow{1, c_{ij}} v_j \wedge; v_j \xrightarrow{1, c_{ij}} v_i \quad (2.1)$$

Różnicę między grafami skierowanymi a nieskierowanymi przedstawiają rysunki 2.1a oraz 2.1b — w drugim przypadku widzimy, że wiele ścieżek, możliwych do skonstruowania dla grafu nieskierowanego, jest nieosiągalnych w przypadku nadania krawędziom kierunku. Dodatkowo ważnym założeniem które przyjmiemy, będzie brak występowania w grafie wielu krawędzi o wspólnym wierzchołku początkowym oraz końcowym, czyli:  $(e_i \equiv v_{s_i} \rightsquigarrow v_{t_i} \neq e_j \equiv v_{s_j} \rightsquigarrow v_{t_j}) \rightarrow s_i \neq s_j \vee t_i \neq t_j$ , gdzie  $s_i, s_j$  oznaczają odpowiednio wierzchołki początkowe (źródła — ang. *sources*) krawędzi  $e_i$  oraz  $e_j$ , zaś  $t_i, t_j$  — ich węzły końcowe (cele — ang. *targets*). Grafy, które nie spełniają tej własności (posiadają więcej niż jedną krawędź prowadzącą bezpośrednio z wierzchołka początkowego  $v_s$  do węzła końcowego  $v_t$  nazywamy **multigrafami** i nie będziemy się nimi zajmować.)



Rysunek 2.1: (a) Nieskierowany graf  $G = (V, E)$ , gdzie  $V = \{v_1, v_2, \dots, v_8\}$  i  $E = \{e_1, e_2, \dots, e_{11}\}$ . (b) Skierowana wersja tego samego grafu.

### 2.1.1 Drzewo rozpinające

Rozpoczniemy od definicji drzewa, które jest specyficznym rodzajem grafu:

**Definicja 2.1.1** Drzewo jest spójnym grafem nie zawierającym żadnych cykli.

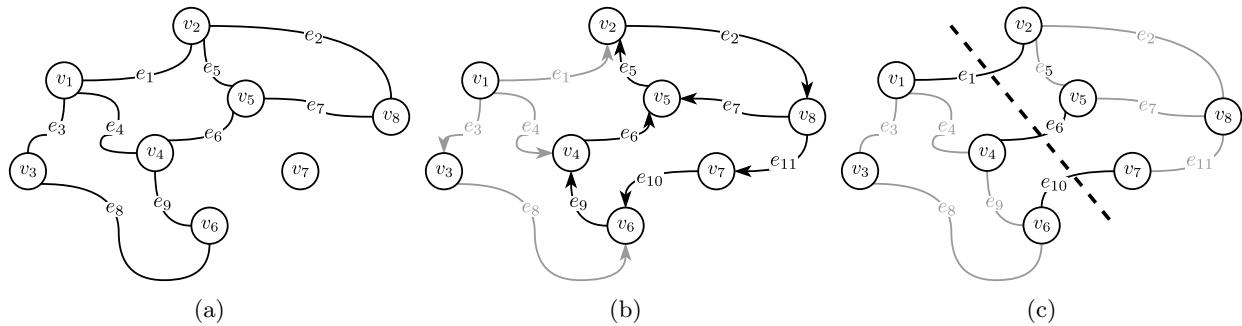
**Grafem spójnym** z kolei nazywamy graf, którego wszystkie wierzchołki są ze sobą w dowolny sposób połączone — do wszystkich z nich jesteśmy w stanie dojść z wykorzystaniem pewnej liczby krawędzi grafu. **Cyklem** zaś nazywamy taką ścieżkę w grafie, której wierzchołek początkowy jest jednocześnie węzłem na końcu tej ścieżki.

**Drzewem rozpinającym** dany graf  $G = (V, E)$  będziemy nazywać taki najmniej liczny zbiór krawędzi  $T$ , który łączy ze sobą wszystkie wierzchołki w grafie. Formalnie:

$$T = \left\{ e \in E : |T| = |V| - 1 \wedge (\forall v, v' \in V : v \neq v') !\exists v \xrightarrow{*^T} v' \right\}, \quad (2.2)$$

gdzie  $|T|$  symbolizuje liczbę krawędzi w zbiorze  $T$ ,  $v \xrightarrow{*^E} v'$  wyraża dowolnej długości ścieżkę składającą się tylko z krawędzi generowanego zbioru  $T$ . Zbiór ten, jak widać z powyższej definicji, powinien mieć tę własność, że dla dowolnych dwóch różnych wierzchołków należących do grafu, istnieje dokładnie jedna ścieżka pomiędzy tymi wierzchołkami. Aby przekonać się, że liczba krawędzi należących do zbioru  $T$  rzeczywiście powinna wynosić  $|V| - 1$ , możemy posłużyć się następującą konstrukcją:

- poprowadźmy w grafie ścieżkę przechodzącą przez wszystkie wierzchołki dokładnie raz i kończącą się w wierzchołku początkowym (zbudujmy **cykl Hamiltona**) — nie trudno zauważyc, że aby połączyć ze sobą wszystkie wierzchołki, potrzebujemy z każdego kolejnego wierzchołka poprowadzić nową krawędź. Otrzymujemy zatem cykl złożony z dokładnie  $|V|$  krawędzi.



Rysunek 2.2: (a) Graf nieskierowany  $G' = (V, E)$ , gdzie  $V = \{v_1, v_2, \dots, v_8\}$  i  $E = \{e_1, e_2, \dots, e_{11}\}$  zawierający 5 cykli:  $\{e_1, e_2, e_7, e_6, e_9, e_8, e_3\}$ ,  $\{e_2, e_7, e_5\}$ ,  $\{e_2, e_7, e_6, e_4, e_1\}$ ,  $\{e_1, e_5, e_6\}$ ,  $\{e_1, e_5, e_6, e_9, e_8, e_3\}$ . Graf jest niespójny — wierzchołek  $v_7$  nie jest w żaden sposób połączony z pozostałymi wierzchołkami grafu. (b) Graf skierowany posiadający tylko dwa cykle (zaznaczone czarnym kolorem):  $\{e_2, e_7, e_5\}$  oraz  $\{e_2, e_{11}, e_{10}, e_9, e_6, e_5\}$ . (c) Przykład cięcia w grafie.

- Usuimy teraz dowolną krawędź z cyklu. Ta operacja powoduje oczywiście jego przerwanie, zaś ze sposobu jego konstrukcji wynika, że pozostała ścieżka przechodzi kolejno przez wszystkie węzły w grafie — tworzy drzewo rozpinające o liczbie krawędzi równej  $|V| - 1$ .

## 2.2 Problem minimalnego drzewa rozpinającego

Niech  $\mathcal{T}_G$  oznacza zbiór wszystkich drzew rozpinających dla grafu  $G$ . Problem **minimalnego drzewa rozpinającego** polega na znalezieniu takiego zbioru krawędzi  $T^* \in \mathcal{T}_G$ , że ich całkowity koszt jest najmniejszy spośród wszystkich pozostałych możliwych rozwiązań. Zanim jednak bezpośrednio przejdziemy do omawiania algorytmów, które służą do odnajdywania konstrukcji o takich właściwościach, przedstawimy warunki jakie musi spełniać drzewo, abyśmy mieli pewność że suma kosztów jego krawędzi rzeczywiście jest najmniejsza.

Pierwszym takim podejściem do zdefiniowania warunków optymalności rozwiązania  $T$  jest warunek **optymalnych cięć**. Zaprezentowany na rysunku 2.2c przykład prezentuje cięcie grafu  $G$  — w przypadku cięcia drzew rozpinających takie posunięcie spowoduje jego podział na dwie części, tak jak to pokazano na rysunkach 2.3a–2.3c, gdyż w wyniku zastosowania cięcia przez wybraną krawędź jest ona usuwana z ciętego zbioru. Optymalnym cięciem natomiast będziemy nazywali takie cięcie, w wyniku którego z grafu usuwana jest krawędź o jak najmniejszym (w przypadku problemów minimalizacyjnych) koszcie ze wszystkich znajdujących się na drodze takiego cięcia (np. na rysunku 2.2c cięcie przechodzące przez krawędzie  $\{e_1, e_6, e_{10}\}$ ). W przypadku cięcia drzewa rozpinającego  $T$  (patrz rysunek 2.4) przez krawędź  $e_6$ , zbiór taki będziemy oznaczać przez  $\mathcal{Q}(T, e_6)$  i będą do niego należeć wszystkie krawędzie  $e' \in E$  łączące ze sobą dwa, powstałe w wyniku cięcia, zbiory wierzchołków, zgodnie z poniższą definicją.

$$\mathcal{Q}(T, e) = \{(i, j) : v_i \in V_1 \wedge v_j \in V_2\} \quad (2.3)$$

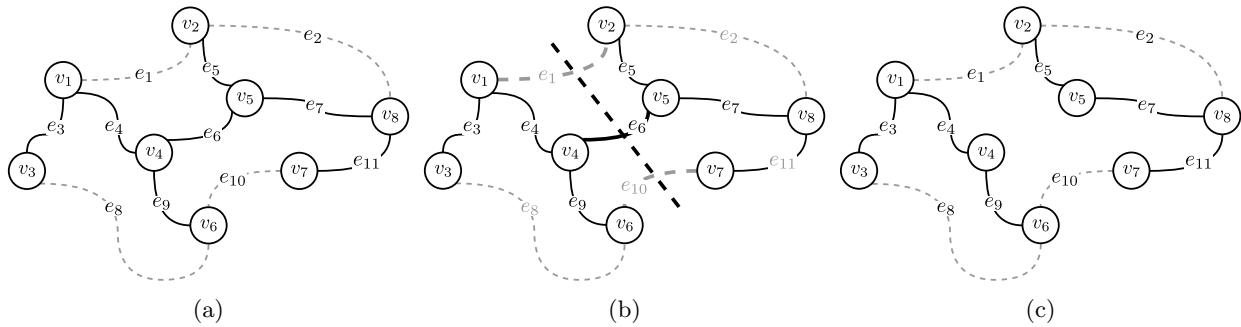
**Twierdzenie 2.2.1 (Kryterium optymalnych cięć (ang. Cut Optimality Conditions))** Dla grafu  $G = (V, E)$ , drzewo rozpinające  $T^*$  jest minimalnym drzewem rozpinającym dany graf wtedy i tylko wtedy, gdy dla każdej krawędzi  $e_{ij} \in T^*$  jej koszt  $c_{ij}$  jest najmniejszy spośród wszystkich krawędzi zbioru  $\mathcal{Q}(T^*, e_{ij})$ , powstającego w wyniku cięcia drzewa  $T^*$  przez krawędź  $e_{ij}$ .

Innymi słowy, przyglądając się rysunkowi 2.3b, koszt krawędzi przez którą dokonujemy cięcia ( $e_6$ ) musi być mniejszy bądź równy wagom krawędzi  $e_1$  oraz  $e_{10}$  — własność ta powinna zachodzić dla każdego innego możliwego cięcia w drzewie rozpinającym (dla przykładu z omawianego rysunku tych cięć jest jeszcze 6 — każde takie cięcie usuwa z drzewa rozpinającego inną jego krawędź). W ogólnym przypadku liczba cięć równa się liczbie krawędzi należących do drzewa rozpinającego (jedno cięcie nie może przebiegać przez wiele krawędzi  $e \in T$  naraz). Argument przemawiający za takim kryterium optymalności rozwiązania jest łatwy do zauważenia — jeżeli krawędź  $e_6$  (patrz 2.3b) nie miałaby najmniejszego kosztu spośród wszystkich krawędzi



należących do  $\mathcal{Q}(T, e_6)$  (to jest albo  $c_1 < c_6$  albo  $c_{10} < c_6$ ), wtedy usuwając z drzewa rozpinającego krawędź  $e_6$  (zrywając połączenie między wierzchołkami grafu) a dodając do niego krawędź  $e_1$  lub  $e_{10}$  (zależnie od przypadku) stworzylibyśmy nowe drzewo rozpinające  $T'$  o koszcie oczywiście mniejszym niż koszt drzewa  $T$ . Zatem drzewo  $T$  w takiej sytuacji z pewnością nie jest szukanym rozwiązańem optymalnym.

**Dowód.** Dowód w pierwszą stronę, pokazujący że jeżeli drzewo  $T$  jest minimalnym drzewem rozpinającym to musi spełniać podane kryterium, jest bardzo prosty, jego główną ideę zdążyliśmy już przedstawić, zatem skupimy się na pokazaniu odwrotnej zależności — jeżeli drzewo  $T^*$  spełnia warunki optymalnych cięć, musi być minimalnym drzewem rozpinającym. Założymy zatem, że drzewo  $T$  jest minimalnym drzewem rozpinającym i jest różne od  $T^*$ . Z połączenia faktów, że  $|T| = n - 1 = |T^*|$  i  $T \neq T^*$  otrzymujemy wniosek, że do drzewa  $T^*$  musi należeć choć jedna krawędź (niech będzie to łuk  $e_{ij} \in T^*$ ), która nie należy do drugiego z drzew ( $e_{ij} \notin T$ ). Usuńmy tą krawędź z  $T^*$ . Tym samym stworzymy podział drzewa  $T^*$  na dwa poddrzewa —  $T_1$  oraz  $T_2$  — których wierzchołki łączone przez ich krawędzie podzielimy na zbiory  $V_1$  i  $V_2$ . Spójrzmy teraz na drzewo  $T$ . Jego krawędzie oczywiście łączą ze sobą te same zbiory wierzchołków (składa się tylko z innych krawędzi). Z definicji zaś drzewa rozpinającego wiemy, że dodając do niego jeszcze jedną krawędź, stworzymy nim cykl. Dodajmy do niego zatem łuk, o którym wiemy, że nie należy do tego drzewa —  $e_{ij}$ . Dodając do tego drzewa dodatkową krawędź stworzyliśmy cykl, którego elementy przynajmniej dwukrotnie przechodzą pomiędzy wierzchołkami należącymi do  $V_1$  oraz  $V_2$  (możemy tu odwołać się do rysunku 2.3b, gdzie przedstawione na nim drzewo to  $T^*$ , zaś usuwana z niego krawędź  $e_{ij}$  to łuk  $e_6$  — z założenia, że  $e_6 \notin T$  wiemy, że do  $T$  musi należeć przynajmniej jeden z wierzchołków  $\{e_1, e_{10}\}$ , tak aby zachować połączenie między wierzchołkami, więc dodanie do niego krawędzi  $e_6$  owocuje obecnością dwóch takich wierzchołków, które łączą ze sobą węzły zbioru  $V_1$  z tymi należącymi do  $V_2$ ). Niech ta drugą krawędzią będzie krawędź  $e_{kl}$ . Drzewo  $T^{ast}$  z założenia spełnia warunek optymalnych cięć tak więc  $c_{ij} \leq c_{kl}$  (cieliśmy je wzduż krawędzi  $c_{ij}$ , więc z założenia wszystkie krawędzie należące do zbioru  $\mathcal{Q}(T^*, e_{ij})$  — w tym  $e_{kl}$  mają koszty nie mniejsze od wagi  $e_{ij}$ ). Dodatkowo, na początku dowodu założyliśmy, że drzewo  $T$  jest minimalnym drzewem rozpinającym graf (jest rozwiązańiem optymalnym) — jego optymalność wymusza aby koszt krawędzi  $e_{kl} \in T$  spełniał warunek  $c_{kl} \leq c_{ij}$  (inaczej z pierwszej części dowodu natychmiast otrzymalibyśmy wynik, że  $T$  nie jest optymalne). Z tych dwóch nierówności otrzymujemy, że  $c_{ij} = c_{kl}$ . Możemy zatem bezkarnie wymienić krawędź  $e_{ij} \in T^*$  na łuk  $e_{kl} \notin T^*$  — otrzymane drzewo nadal będzie mieć takie same koszty (pozostanie rozwiązań optymalnych) a przy okazji liczba krawędzi różniczących go od drzewa  $T$  (optymalnego z założenia) ulegnie zmniejszeniu. Kontynuując powyższe czynności (wymieniając krawędzie drzewa  $T^*$ , które nie należą do  $T$  na te, które są jego częścią), na pewnym etapie konstrukcji takiego drzewa okaże się, że skonstruowane drzewo jest drzewem zawierającym te same krawędzie co  $T$  — jako że po drodze ani razu nie zmienialiśmy kosztów konstruowanego drzewa możemy wyciągnąć wniosek, że drzewo  $T^*$  (od którego wyszliśmy) od początku było optymalne, co mieliśmy udowodnić. ◆



Rysunek 2.3: (a) Drzewo rozpinające dla grafu  $G = (V, E)$ , gdzie  $V = \{v_1, v_2, \dots, v_8\}$  i  $E = \{e_1, e_2, \dots, e_{11}\}$ . (b) Cięcie przez krawędź drzewa rozpinającego  $e_6$  w grafie. Krawędzie leżące na cięciu zostały pogrubione. (c) Zbiory wierzchołków  $V_1 = \{v_1, v_3, v_4, v_6\}$  oraz  $V_2 = \{v_2, v_5, v_7, v_8\}$  powstałe w wyniku podziału drzewa rozpinającego  $T$  na mniejsze poddrzewa. Zbiór krawędzi  $\mathcal{Q}(T, e_6)$  definiowany przez to cięcie zawiera elementy:  $\{e_1, e_{10}\}$ .

Alternatywnym warunkiem określającym optymalność rozwiązania problemu minimalnego drzewa rozpinającego jest kryterium optymalnych ścieżek (ang. *Path Optimality Conditions*), które wygląda następująco:



**Twierdzenie 2.2.2** Kryterium optymalny ścieżek Drzewo rozpinające  $T^*$  jest minimalnym drzewem rozpinającym wtedy i tylko wtedy, gdy dla każdej krawędzi spoza tego drzewa  $e_{kl} \in E \setminus T^*$ , dla każdej krawędzi  $e_{ij} \in T^*$  należącej do ścieżki  $v_k \xrightarrow{*} v_l$  zachodzi  $c_{ij} \leq c_{kl}$ .

**Dowód.** Pokażmy, że jeśli drzewo  $T^*$  jest minimalnym drzewem rozpinającym, to spełnia warunek optymalnych ścieżek. Dowód ten częściowo wynika z poprzedniego — jeżeli drzewo  $T^*$  jest optymalne i założymy, że istnieje taka krawędź  $e_{ij}$  na ścieżce pomiędzy wierzchołkami  $v_k$  a  $v_l$  taka, że  $c_{ij} > c_{kl}$ , to dodając do drzewa krawędź  $e_{ij}$  otrzymamy cykl, którego nie wszystkie koszty krawędzi są mniejsze od wagi nowo dodanego łuku. Aby na powrót otrzymać drzewo rozpinające musimy przerwać cykl, wyrzucając z rozwiązania jedną krawędź cyklu — jako że własność optymalnej ścieżki nie była spełniona, wśród krawędzi cyklu na pewno jest łuk  $e$ , którego koszt jest większy od kosztu nowej krawędzi, zatem naturalnym posunięciem będzie usunąć właśnie ten łuk, by otrzymać jak najlepsze rozwiązanie. Usuwając go jednak otrzymujemy nowe rozwiązanie, którego koszt jest mniejszy od kosztu pierwotnego drzewa  $T$ , które było optymalne. Otrzymaliśmy zatem sprzeczność. Możemy także pokazać równoważność powyższych dwóch twierdzeń — zauważmy, że jeśli dane drzewo  $T^*$  podzielimy poprzez wykonanie cięcia wzdłuż krawędzi  $e_{ij}$  (tak jak w poprzednim dowodzie tworząc zbiory  $T_1, T_2, V_1, V_2$ ), wybierzymy dowolną krawędź  $e_{kl}$  taką, że  $v_k \in V_1$  i  $v_l \in V_2$ , to z warunku optymalności ścieżki otrzymujemy natychmiast, że  $c_{ij} \leq c_{kl}$ , gdzie krawędź  $e_{kl}$  jest dowolną krawędzią należącą do  $\mathcal{Q}(T^*, e_{ij})$  — przedstawiona własność jest własnością optymalnego cięcia, którą drzewo  $T^*$  spełnia zatem na mocy poprzedniego twierdzenia jest optymalne. ♦

## 2.3 Znane algorytmy rozwiązuające problem MST

Problem minimalnego drzewa rozpinającego jest bardzo dobrze znany toteż istnieje wiele algorytmów (ich wariacji) radzących sobie z danym problemem. W tej części skupimy się na dwóch podstawowych: algorytmie Josepha Kruskala [1, 520–522], Vojtěcha Jarníka (Prima) [1, 523–525]. Inne sposoby podejścia do problemu o jakich wspomnijmy w następnych rozdziałach to: algorytm Chazelle'iego i również sobie z nim radzące modele programowania liniowego oraz całkowitoliczbowego (o tych ostatnich więcej opowiem w rozdziale 4). Algorytmami, którymi się nie będziemy zajmować są natomiast: algorytm Tarjana oraz Borůvka.

### 2.3.1 Algorytm Kruskala

Pierwszym algorytmem, którego schemat działania omówimy, będzie algorytm Kruskala, który w bardzo dużym stopniu polega na udowodnionym przez nas kryterium optymalności drzewa rozpinającego — optymalnych ścieżek (2.2.2). Jak pamiętamy, zgodnie z podanym kryterium, drzewo rozpinające  $T$  jest minimalnym drzewem rozpinającym grafu  $G$  tylko wtedy, gdy wszystkie krawędzie nienależące do tego drzewa, zaś należące do ścieżki między dwoma wierzchołkami, które łączy krawędź należąca do  $T$ , mają koszt nie większy niż waga tej ostatniej. Definicja ta bezpośrednio przekłada się na ideę algorytmu: będziemy chcieli kolejno dodawać do naszego rozwiązania krawędzie w kolejności od ich najmniejszego kosztu do największej wagi, konstruując przy tym coraz to dłuższe ścieżki, aż do momentu, w którym na ścieżkach nie zaczną pojawiać się cykle. Dzięki uprzedniemu posortowaniu krawędzi względem ich kosztów mamy w tym przypadku pewność, że na takiej ścieżce znajdują się tylko krawędzie o najniższych kosztach, zaś wszystkie pozostałe łuki, które leżały na tej ścieżce (a których nie możemy już dodać ze względu na pojawienie się cyklu), mają większy koszt krawędzi niż ostatni łuk dodany do ścieżki. Naszym głównym celem zatem jest konstruowanie ścieżek — w linii 8 pseudokodu 1 sprawdzamy, czy oba wierzchołki, które łączy analizowana przez nas krawędź nie należą do tego samego zbioru. Jeśli tak jest — krawędź którą chcemy dodać utworzyłaby cykl na ścieżce, do której należą oba te wierzchołki, zatem nie chcemy dodawać takiej krawędzi. Oczywiście aby algorytm działał poprawnie, zakładamy że zbiór krawędzi, po którym iterujemy (7–12) jest odpowiednio posortowany. W przypadku chęci dodania kolejnej krawędzi do ścieżki (gdy krawędź łączy różne zbiory —  $T_0 \neq T_1$ ) musimy zaś połączyć ze sobą zbiory  $T_0$  i  $T_1$ . Całość prezentuje się w formie pseudokodu zamieszczonego w 1.

Czas działania takiego algorytmu oczywiście zależy od sposobu zaimplementowania linii 5 oraz 8–12, lecz górną jego granicą to  $O(m + n \cdot \log(n))$  [1, 522].

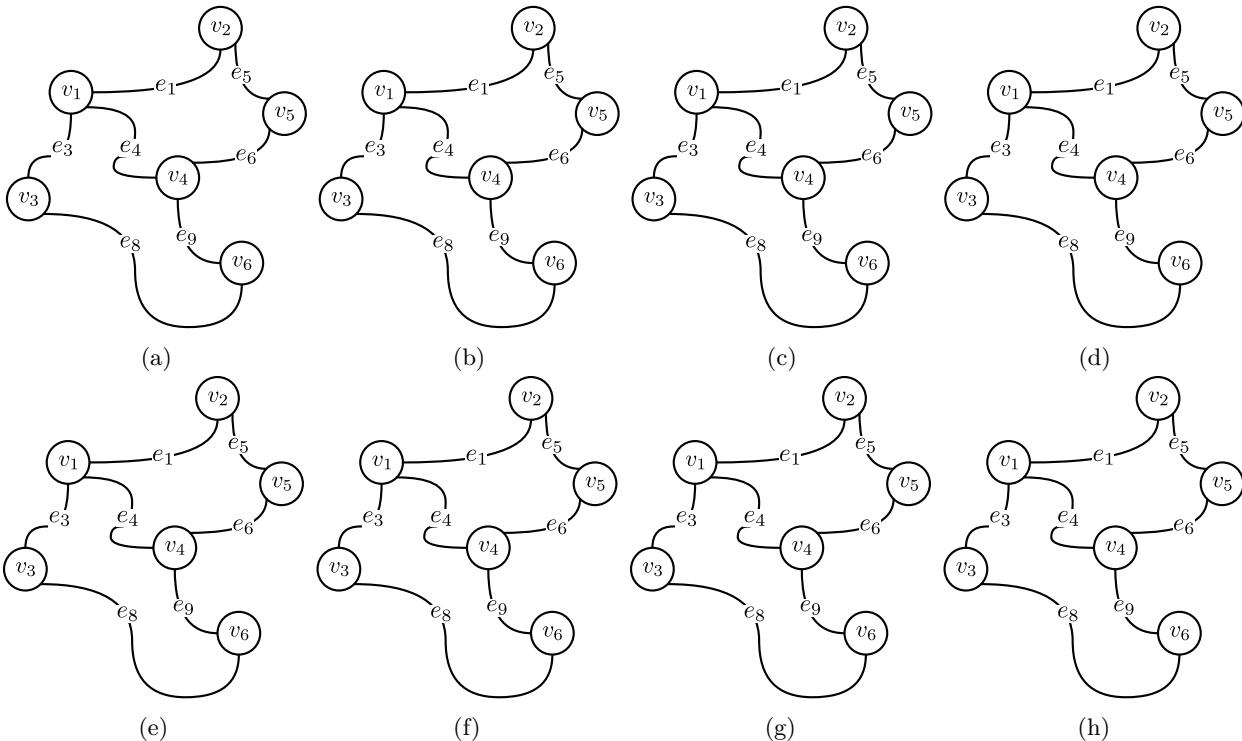
**Pseudokod 1: KRUSKAL-MST ( $G$ )**

**Wejście:**  $G = (V, E)$  — graf wejściowy,  
**Wyjście:**  $T^*$  — minimalne drzewo rozpinające.

```

1 begin
2    $T^* \leftarrow \emptyset$ 
3    $T \leftarrow \emptyset$ 
4   foreach  $v \in V$  do
5      $T \leftarrow T \cup \{v\}$ 
6   INC-ORDER ( $E$ )
7   foreach  $e_{ij} \in E$  do
8     if  $(T_0 : v_i \in T_0) \neq (T_1 : v_i \in T_1)$  then
9        $T^* \leftarrow T^* \cup e_{ij}$ 
10       $T \leftarrow T \setminus T_0$ 
11       $T \leftarrow T \setminus T_1$ 
12     $T \leftarrow T \cup \{v : v \in T_0 \cup T_1\}$ 
13
return  $T^*$ 

```



Rysunek 2.4: (a) Drzewo rozpinające dla grafu  $G = (V, E)$ , gdzie  $V = \{v_1, v_2, \dots, v_8\}$  i  $E = \{e_1, e_2, \dots, e_{11}\}$ .  
(b) Cięcie przez krawędź drzewa rozpinającego  $e_6$  w grafie. Krawędzie leżące na cięciu zostały pogrubione.  
(c) Zbiory wierzchołków  $V_1 = \{v_1, v_3, v_4, v_6\}$  oraz  $V_2 = \{v_2, v_5, v_7, v_8\}$  powstałe w wyniku podziału drzewa rozpinającego  $T$  na mniejsze poddrzewa. Zbiór krawędzi  $\mathcal{Q}(T, e_6)$  definiowany przez to cięcie zawiera elementy:  $\{e_1, e_{10}\}$ .

### 2.3.2 Algorytm Prima

---

**Pseudokod 2:** PRIME-MST ( $G, v_1$ )

---

**Wejście:**  $G = (V, E)$  — graf wejściowy,  
 $v_1$  — węzeł początkowy, od którego rozpocznie się konstrukcja rozwiązania.  
**Wyjście:**  $T^*$  — minimalne drzewo rozpinające.

```
1 begin
2   foreach  $v \in V \setminus (v_1 \cup \{v' : v_1 \sim v'\})$  do
3      $v.d \leftarrow \infty$ 
4    $v_1.d \leftarrow 0$ 
5   foreach  $v_i : v_1 \sim v_i$  do
6      $v_i.d \leftarrow c_{1i}$ 
7      $v_i.p \leftarrow v_1$ 
8    $H \leftarrow \text{CREATE-HEAP}(G)$ 
9   foreach  $v \in V$  do
10     $\text{INSERT}(v, H)$ 
11    $T^* \leftarrow \emptyset$ 
12   while  $|T^*| < |V| - 1$  do
13      $v_i \leftarrow \text{FIND-MIN}(H)$ 
14      $\text{DELETE-MIN}(H)$ 
15      $T^* \leftarrow T^* \cup (v_i.p \sim v_i)$ 
16     foreach  $j : v_i \sim v_j \wedge v_j \in H$  do
17       if  $v_j.d > c_{ij}$  then
18          $v_j.c \leftarrow c_{ij}$ 
19          $v_j.p \leftarrow v_i$ 
20          $\text{DEC-KEY}(j, c_{ij}, H)$ 
21   return  $T^*$ 
```

---

## 2.4 Podsumowanie rozdziału

Problem minimalnego drzewa rozpinającego jest jednym z fundamentalnych problemów optymalizacyjnych pojawiających się w wielu dziedzinach codziennego życia. Wszędzie tam, gdzie może nam zależeć na np. jak największym uproszczeniu badanej struktury grafowej, pozbycia się redundantnych rozwiązań, bardzo prawdopodobnym jest, że napotkamy właśnie problem minimalnego drzewa rozpinającego. Uogólniając, zależnie od sposobu interpretacji elementów grafu możemy znaleźć wiele zastosowań dla wspomnianego problemu. Warto też zauważać, że nie musimy ograniczać się tylko do problemu znalezienia rozwiązania o najmniejszej sumie kosztów — jeżeli nasze koszty będą skonstruowane w ten sposób, że każdy z nich przybierać będzie postać logarytmu o ustalonej podstawie, ich suma tak naprawdę będzie wyrażać iloczyn rzeczywistych kosztów jakie chcemy w problemie zatrzymać. Tworzy nam to jeszcze więcej możliwości zastosowania tak postawionego problemu [1, 512–516].

Zapoznawszy się z podstawowymi pojęciami dotyczącymi problemów grafowych, naszą uwagę w następnych rozdziałach poświęcimy problemom bardziej złożonym, u podstaw których znajdziemy właśnie problem minimalnego drzewa rozpinającego (widzimy zatem, że jego zastosowania nie kończą się tylko na bezpośrednim zidentyfikowaniu problemu jako minimalnego drzewa rozpinającego, lecz jest on również podstawą do rozwiązywania wielu innych, bardziej złożonych problemów). W tym zaś przedstawiliśmy podstawowe narzędzia pozwalające nam na uporanie się z nim, przedstawiliśmy warunki optymalności takiej konstrukcji oraz przytoczyliśmy liczne definicje, z których będziemy korzystać we wszystkich następnych rozdziałach.



# Odporna optymalizacja

---

W poprzednim rozdziale zapoznaliśmy się z podstawowymi informacjami na temat problemu odnajdywania minimalnych drzew rozpinających w grafach oraz poznaliśmy sposoby radzenia sobie z nim. Do tej pory jednak nie poruszyliśmy najistotniejszego dla nas tematu: **optymalizacji odpornej** na czynniki zewnętrzne. Co przez to należy rozumieć? Do tej pory poznaliśmy algorytmy, które potrafią rozwiązywać problem w środowisku zamkniętym — idealnym, w którym nic nie ma prawa się zmieniać przez cały czas pracy algorytmu. W tym rozdziale omówimy modele problemów nie posiadających takiego ograniczenia — w modelach tych dane nie tylko będą mogły się zmieniać, ale będziemy także dopuszczać możliwość bycia nieświadomym tych zmian. W charakterze podsumowania formalnie zapiszemy model jednego z wariantów przedstawionych problemów, wskazując tym samym ogólnie podejście do jego rozwiązania.

---

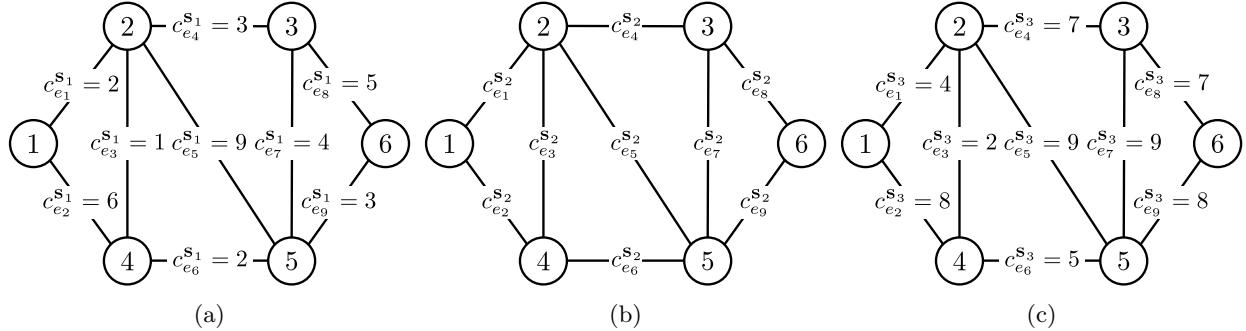
## 3.1 Scenariusze dyskretnie a ciągłe

W poprzednim rozdziale skupialiśmy swoją uwagę na problemie wyszukiwania minimalnego drzewa rozpinającego w grafie  $G = (V, E)$ , czyli takiego zbioru krawędzi  $E^*$ , który łączył ze sobą wszystkie wierzchołki w grafie, a jednocześnie którego suma kosztów krawędzi  $e \in E^*$  była możliwie najmniejsza. Formalnie:

$$E_{G=(V,E)}^* = \left\{ e \in E : E_{G=(V,E)}^* \in \mathcal{T}_G \wedge (\forall E' \in \mathcal{T}_G) \sum_{e \in E'} c_e \geq c_{E_{G=(V,E)}^*} \right\}, \quad (3.1)$$

gdzie  $\mathcal{T}_G$  oznaczał zbiór wszystkich drzew możliwych do skonstruowania w grafie  $G$ ,  $c_e$  — koszt krawędzi  $e \in E$ , zaś  $\sum_{e \in E'} c_e = c_{E'}$  — sumę kosztów wszystkich krawędzi wynikającą ze **scenariusza**. Scenariuszem  $\mathbf{s}$  zatem nazwiemy zbiór definiujący koszt dla każdej krawędzi w grafie:  $\mathbf{s} = \{c_{e_i} : i = \{1, \dots, |E|\}\}$ . Na rysunku 3.1 odpowiednio widzimy przykłady trzech różnych scenariuszy zastosowanych do tego samego drzewa. Rzecz jasna, zależnie od kosztów jakie zostaną narzucone krawędziom przez dany scenariusz, zwracane przez dotychczas poznane algorytmy rozwiązania będzie się różnić. Zwróćmy uwagę na rysunek 3.1b, gdzie celowo nie zostały ujawnione koszty scenariusza — jedyne co o nim wiemy to to, że żaden koszt krawędzi w tym scenariuszu nie jest mniejszy od kosztu odpowiadającej mu krawędzi w  $\mathbf{s}_1$  ani większy od kosztu tej samej krawędzi w scenariuszu  $\mathbf{s}_3$ . Taki rodzaj scenariusza nazywamy **scenariuszem ciągłym**, zaś scenariusze, których poszczególne koszty krawędzi są najmniejsze (bądź największe) spośród wszystkich scenariuszy, nazywamy **scenariuszami granicznymi/krytycznymi** (odpowiednio **scenariuszem najlepszego/najgorszego przypadku**).

Przedstawione rodzaje scenariuszy można rozważyć dla wszystkich problemów odpornej optymalizacji dyskretnej jakie tutaj wymienimy. W przypadku problemu minimalizacyjnego jakim jest omawiany przez nas problem znajdowania minimalnego drzewa rozpinającego, krytyczne scenariusze najlepszego i najgorszego przypadku będące oznaczać odpowiednio  $\underline{\mathbf{s}}$  oraz  $\bar{\mathbf{s}}$ , gdzie  $0 \leq \underline{\mathbf{s}} \leq \bar{\mathbf{s}}$ . Dla problemów maksymalizacyjnych znaczenie przedstawionych symboli byłoby zgoła odwrotne:  $\underline{\mathbf{s}}$  oznaczałby krytyczny scenariusz o największych współczynnikach (najlepszego przypadku), zaś  $\bar{\mathbf{s}}$  — najgorszego, gdzie każdy jego współczynnik nie byłby większy od odpowiadających mu współczynników w pozostałych scenariuszach. **Scenariuszem dyskretnym** (ang. *discrete scenario*) będące natomiast nazywać każdy scenariusz, który ma zdefiniowane koszty dla każdej krawędzi. Takimi scenariuszami są na przykład scenariusze krytyczne (ang. *extreme scenarios*).



Rysunek 3.1: Scenariusze  $s_1$ ,  $s_2$ ,  $s_3$  dla grafu nieskierowanego  $G = (V, E)$ ,  $V = \{1, 2, \dots, 6\}$ ,  $E = \{e_i : i = \{1, \dots, 9\}\}$ . (a) Scenariusz najlepszego przypadku  $s_1 = [2, 6, 1, 3, 9, 2, 4, 5, 3]$ . Dla scenariusza zachodzi:  $(\forall i \in \{1, \dots, 9\}) c_{e_i}^{s_1} \leq c_{e_i}^{s_2}$ . Scenariusz o takich właściwościach będziemy też oznaczać jako  $\underline{s}$ . (b) Scenariusz ciągły  $s_2$ . Koszty tego scenariusza zdefiniowane są następująco:  $(\forall i \in \{1, \dots, 9\}) c_{e_i}^{\underline{s}} \leq c_{e_i}^{s_2} \leq c_{e_i}^{\bar{s}}$ . (c) Krytyczny scenariusz najgorszego przypadku  $s_3 = [4, 8, 2, 7, 9, 5, 7, 9, 8]$ . Dla scenariusza zachodzi:  $(\forall i \in \{1, \dots, 9\}) c_{e_i}^{s_2} \leq c_{e_i}^{s_3}$ . Scenariusz o takich właściwościach będziemy też oznaczać jako  $\bar{s}$ .

## 3.2 Problemy minimaksowe

Problemami minimaksowymi nazywamy problemy, których celem jest odnalezienie najlepszego rozwiązania przy założeniu wystąpienia najgorszych z możliwych scenariuszy dla danych rozwiązań. Możemy rozróżnić tutaj problemy typu MIN–MAX 3.2 oraz MAX–MIN 3.3:

$$\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s}) \quad (3.2)$$

oraz

$$\max_{\mathbf{x} \in X} \min_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s}), \quad (3.3)$$

przy czym  $\mathbf{x}$  jest rozwiązaniem problemu, wybranym ze zbioru wszystkich możliwych rozwiązań  $X$ ,  $v(\mathbf{x}, \mathbf{s})$  reprezentuje koszt rozwiązania problemu dla scenariusza  $\mathbf{s}$  i wybranego zbioru krawędzi  $\{e \in E : x_e = 1\}$ <sup>1</sup>, zaś  $S$  reprezentuje zbiór dostępnych scenariuszy. Oczywiście w przypadku gdy  $|S| = 1$ , problem MIN–MAX SPANNING TREE sprowadza się do klasycznego problemu minimalnego drzewa rozpinającego.

### 3.2.1 Przypadek ciągły

Niech  $S = \{\mathbf{s} : (\forall i \in \{1, \dots, |E|\}) c_e^{\mathbf{s}} \in [c_e^{\underline{s}}, c_e^{\bar{s}}]\}$ , gdzie scenariusze  $\underline{s}$  i  $\bar{s}$  są odpowiednio scenariuszami przedstawionymi na rysunkach 3.1a i 3.1c zaś  $c_e^{\mathbf{s}}$  oznacza koszt krawędzi  $e$  dla scenariusza  $\mathbf{s}$ . Innymi słowy: zbiór scenariuszy  $S$  składa się z takich schematów kosztów  $\mathbf{s}$ , że każdy koszt krawędzi  $c_e^{\mathbf{s}}$  w tym scenariuszu może przyjmować wartość należącą do przedziału  $[c_e^{\underline{s}}, c_e^{\bar{s}}]$ . Naszym celem niech będzie rozwiązanie problemu MIN–MAX INTERVAL SPANNING TREE.

Jak łatwo zauważać, w przypadku ciągłym zachodzi następująca prawidłowość:

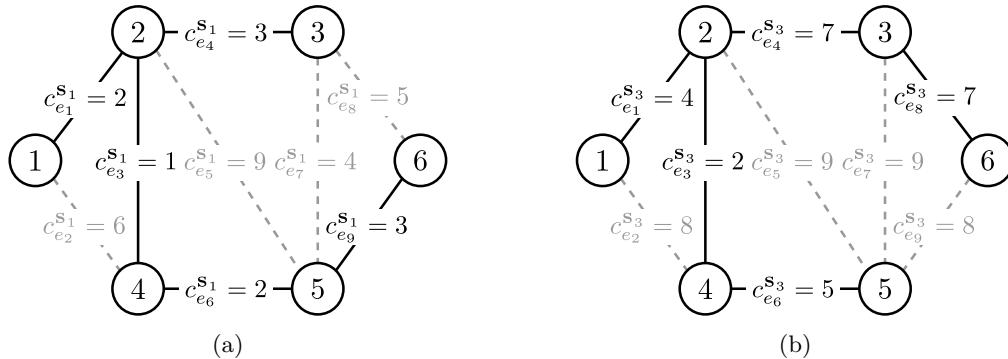
$$\min_{\mathbf{x} \in X} \left( \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s}) \right) = \min_{\mathbf{x} \in X} \left( \max_{\mathbf{s} \in S} \sum_{e \in E} x_e \cdot c_e^{\mathbf{s}} \right) = \min_{\mathbf{x} \in X} \left( \sum_{e \in E} x_e \cdot c_e^{\bar{s}} \right) = \min_{\mathbf{x} \in X} v(\mathbf{x}, \bar{s}), \quad (3.4)$$

jako że dla scenariusza  $\bar{s}$  i każdego definiowanego przez niego kosztu i dla dowolnego scenariusza  $\mathbf{s} \in S$  zachodzi:  $c_e^{\bar{s}} \geq c_e^{\mathbf{s}}$ . Podobne rozumowanie możemy przeprowadzić dla problemu MAX–MIN. Widzimy zatem, że w przypadku tak postawionego problemu, dla scenariuszy o rozmytych/niepewnych kosztach, bardzo łatwo możemy znaleźć jego rozwiązanie, redukując problem do klasycznego problemu z jednym, krytycznym scenariuszem.

<sup>1</sup>Często dla własnej wygody będziemy zamiennie stosować oznaczenia:  $x_e$  oraz  $x_{e_i}$  (lub  $x_i$ ) dla zmiennej decyzyjnej w wektorze  $\mathbf{x}$  będącym wybranym rozwiązaniem problemu minimalnego drzewa rozpinającego, czy też  $c_e$  i  $c_{e_i}$  (lub  $c_i$ ) dla oznaczeń kosztów krawędzi, chyba że zostanie napisane inaczej. Oznaczenia  $x_{e_i}/x_i$  oraz  $c_{e_i}/c_i$  będziemy stosować w przypadku, gdy będzie wymagane zachowanie kolejności oznaczeń np. dla krawędzi w zbiorze występujących kolejno po sobie ( $x_{e_i}, x_{e_{i+1}}, \dots$ ).

### 3.2.2 Przypadek dyskretny

Powróćmy raz jeszcze do rysunku 3.1. Niech tym razem naszym zbiorem scenariuszy będzie  $S = \{\mathbf{s}_1, \mathbf{s}_3\}$ , gdzie  $\mathbf{s}_1$  i  $\mathbf{s}_3$  odnoszą się odpowiednio do 3.1a oraz 3.1c. **Dyskretnym** zbiorem scenariuszy nazwiemy taki zbiór, którego elementy jesteśmy w stanie ponumerować — tutaj nasz zbiór scenariuszy posiada dokładnie dwa elementy.



Rysunek 3.2: Dyskretnie scenariusze  $\mathbf{s}_1$ ,  $\mathbf{s}_3$  dla grafu nieskierowanego  $G = (V, E)$ ,  $V = \{1, 2, \dots, 6\}$ ,  $E = \{e_i : i \in \{1, \dots, 9\}\}$ .

Pomimo tak prosto zdefiniowanego zadania, przekonamy się że jego rozwiążanie wcale nie jest tak intuicyjne jak byśmy sobie tego życzyli — w tabeli 3.1 przedstawiono kilka z 55 możliwych rozwiązań zadanego problemu<sup>2</sup>. Na uwagę w niej zasługują rozwiązania  $\mathbf{x}_1$  oraz  $\mathbf{x}_2$ , które są rozwiązaniami optymalnymi dla scenariuszy  $\mathbf{s}_3$  (!) i  $\mathbf{s}_1$  rozpatrywanych osobno. Jeżeli spojrzymy na ostatnią kolumnę, zauważmy że optymalne rozwiązanie dla pierwszego z wymienionych scenariuszy jest także rozwiązaniem optymalnym problemu minimaxowego — tak jak w przypadku scenariuszy ciągłych, tak i tutaj mogliśmy skorzystać z posiadanej wiedzy o kosztach definiowanych przez każdy ze scenariuszy; wiedząc, że żaden z nich dla scenariusza  $\mathbf{s}_3$  nie jest mniejszy od odpowiadającego mu w drugim ze scenariuszy nie powinno być dla nas zaskoczeniem, że optymalność rozwiązania problemu dla scenariusza o wyższych kosztach pociąga za sobą optymalność rozwiązania problemu MIN-MAX. Należy jednak podkreślić, że taka sytuacja jest zazwyczaj mało prawdopodobna i bardzo łatwo możemy stworzyć scenariusze, dla których ta prawidłowość nie będzie zachodziła.

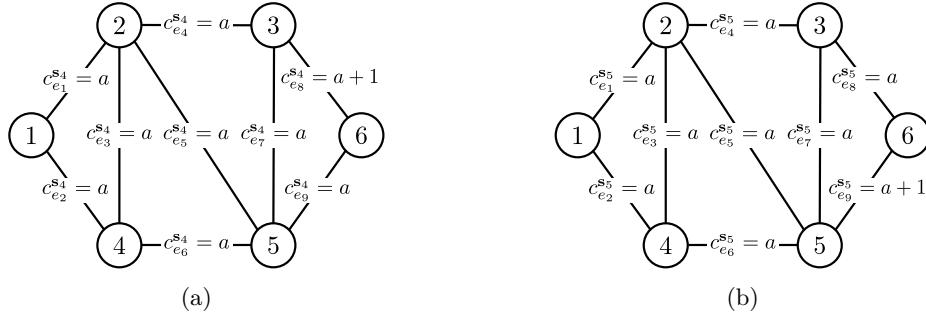
$X$	Scenariusze											
	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$e_8$	$e_9$	$v(\mathbf{x}, \mathbf{s}_1)$	$v(\mathbf{x}, \mathbf{s}_3)$	
$\mathbf{x}_1$	1	0	1	1	0	1	0	1	0	13	25	25
$\mathbf{x}_2$	1	0	1	1	0	1	0	0	1	11	26	26
$\mathbf{x}_3$	1	0	1	0	0	1	0	1	1	13	26	26
$\mathbf{x}_4$	1	0	1	0	0	1	1	1	0	14	27	27
...	...	...	...	...	...	...	...	...	...	...	...	...
$\mathbf{x}_{53}$	1	1	0	0	1	0	1	0	1	24	38	38
$\mathbf{x}_{54}$	0	1	0	0	1	1	1	1	0	26	38	38
$\mathbf{x}_{55}$	0	1	0	0	1	1	1	0	1	24	39	39

Tablica 3.1: Tabela przedstawiająca część z osiągalnych (ang. *feasible*) rozwiązań dla problemu minimalnego drzewa rozpinającego dla scenariuszy  $\mathbf{s}_1$  oraz  $\mathbf{s}_3$  (3.2a i 3.2b), koszty dla każdego z proponowanych rozwiązań dla podanych scenariuszy oraz wartość rozwiązania dla problemu MIN-MAX. Wiersze w tabeli zostały posortowane w kolejności rosnących wartości rozwiązań.

<sup>2</sup>Rozważane przykłady grafów posiadają 9 krawędzi łączących jego wierzchołki — spośród wszystkich możliwych 512 kombinacji zbiorów krawędzi należących do rozwiązania (każda z nich może do niego niezależnie należeć bądź nie, co daje nam  $2^9$  możliwości wyboru rozwiązania) tylko 55 z nich zawiera krawędzie, które tworzą drzewo rozpinające dla danego grafu. Rozwiązania zostały wygenerowane poprzez modyfikację wszystkich kosztów w grafie (ustawieniu ich na 0) i pobranie wszystkich optymalnych rozwiązań dla tak zadanego problemu (w takim przypadku każde dopuszczalne rozwiązanie było także rozwiązaniem optymalnym — więcej o zagadnienniu optymalności opowiem w rozdziale następnym, poświęconym LP).

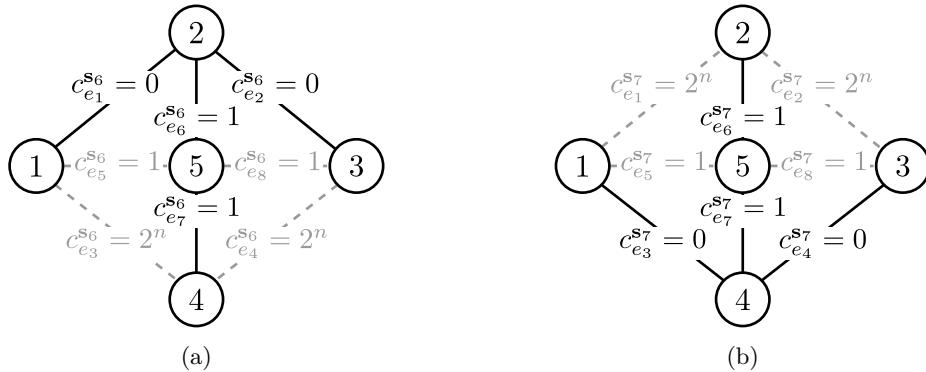


Niech scenariusze  $\mathbf{s}_4$  i  $\mathbf{s}_5$  będą takie, że dla każdego  $e \in \{e_i : 1, \dots, |E| - 2\}$   $c_e^{\mathbf{s}_4} = c_e^{\mathbf{s}_5}$  oraz niech  $c_{e_{|E|-1}}^{\mathbf{s}_4} = c_{e_{|E|}}^{\mathbf{s}_5} + 1$  i  $c_{e_{|E|}}^{\mathbf{s}_4} = c_{e_{|E|}}^{\mathbf{s}_5} - 1$ , tak jak to pokazano na rysunku 3.3. Użyając rozwiązać postawiony przed nami problem postaci:  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s})$  szybko zauważymy, że w tym przypadku odnalezienie optymalnych rozwiązań dla każdego ze scenariuszy i próba wybrania spośród nich jednego jako rozwiązania problemu **min–max** nie jest dobrym rozwiązaniem: dla scenariusza  $\mathbf{s}_4$  optymalnym wyborem jest każdy podzbiór zbioru krawędzi grafu tworzący drzewo rozpinające, niezawierający łuku  $e_8$  o koszcie  $a+1$  (koszt takiego rozwiązania wynosi  $a \cdot (|V| - 1)$ ), dla scenariusza  $\mathbf{s}_5$  uzyskamy podobne wyniki, tym razem wykluczając ze zbioru rozwiązań te podzbiory, do których należy krawędź  $e_9$ . W obydwu przypadkach wartość optymalnego rozwiązania wynosi tyle samo, zaś prawidłowym rozwiązaniem problemu  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s})$  jest rozwiązanie dowolne — analizowany graf i jego koszty zostały tak dobrane, że wartość wyrażenia  $\max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s}) = 5 \cdot a + 1$  dla każdego dopuszczalnego rozwiązania. Wystarczy zauważać, że jedyną istotną decyzją jaką musimy podjąć, jest wybór ostatniej krawędzi, zaś niezależnie od tego wyboru maksymalny koszt wszystkich wybranych krawędzi nie ulegnie zmianie.



Rysunek 3.3: Dyskretnie scenariusze  $\mathbf{s}_4$ ,  $\mathbf{s}_5$  dla grafu nieskierowanego  $G = (V, E)$ ,  $V = \{1, 2, \dots, 6\}$ ,  $E = \{e_i : i = \{1, \dots, 9\}\}$ . Dla tak określonych scenariuszy dłużej już nie zachodzi własność:  $\forall e \in E : c_e^{\mathbf{s}_4} \leq c_e^{\mathbf{s}_5}$ .

Problem jest tym bardziej widoczny, im większe różnice będą zachodzić pomiędzy poszczególnymi scenariuszami. Za przykład niech posłuży nam ilustracja 3.4, gdzie pomimo występowania nadal jedynie dwóch scenariuszy, opieranie się na dotychczasowej intuicji policzenia optymalnych rozwiązań dla każdego z nich i wybranie jednego, prowadzi do znacznie poważniejszych błędów niż miało to miejsce w poprzednim przykładzie, gdzie pomimo zastosowania błędnej logiki, otrzymaliśmy poprawny rezultat. Wprowadźmy kolejne scenariusze:  $\mathbf{s}_6 = [0, 0, 2^n, 2^n, 1, 1, 1, 1]$  oraz  $\mathbf{s}_7 = [2^n, 2^n, 0, 0, 1, 1, 1, 1]$ , gdzie  $n$  jest dowolne<sup>3</sup>.



Rysunek 3.4: Skrajny przykład problemu DESCRIPTIVE MIN-MAX SPANNING TREE, gdzie optymalne rozwiązania dla poszczególnych scenariuszy okazują się być bardzo złe (dla dużych  $n$ ) w porównaniu z rozwiązaniem głównego problemu. Podobny przykład [2, 429–430] można skonstruować dla minimaxowego problemu najkrótszej ścieżki.

<sup>3</sup>Chcemy aby funkcje zmiennej  $n$  przyjmowały wartość większą od zera, co w przypadku funkcji wykładniczej jest spełnione dla dowolnego  $n \in \mathbb{R}$ .

Optymalnymi rozwiązaniami dla poszczególnych scenariuszy są oczywiście:  $E_{\mathbf{s}_6}^* = \{e_1, e_2, e_6, e_7\}$  o całkowitym koszcie równym 2, oraz  $E_{\mathbf{s}_7}^* = \{e_3, e_4, e_6, e_7\}$ , gdzie  $v(\mathbf{s}_7, x_{\mathbf{s}_7}^*)$  ma tą samą wartość co poprzednie rozwiązanie<sup>4</sup>. Oczywiście po podstawieniu otrzymanych danych do wzoru:  $\min_{\mathbf{x} \in X^*} \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s})$ , gdzie  $X^* = \{\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{x}_{\mathbf{s}_7}^*\}$  otrzymamy:

$$\min \left\{ \max_{\mathbf{s} \in S} v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}), \max_{\mathbf{s} \in S} v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}) \right\} = \min \left\{ \max \{v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_6), v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_7)\}, \max \{v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_6), v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_7)\} \right\}.$$

Łatwo zauważać, że  $v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_6) < v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_7)$  oraz  $v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_7) < v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_6)$ . Stąd:

$$\max \{v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_6), v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_7)\} = v(\mathbf{x}_{\mathbf{s}_6}^*, \mathbf{s}_7) = 2 \cdot 2^n + 2 = 2^{n+1} + 2, \quad (3.5)$$

$$\max \{v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_6), v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_7)\} = v(\mathbf{x}_{\mathbf{s}_7}^*, \mathbf{s}_6) = 2 \cdot 2^n + 2 = 2^{n+1} + 2, \quad (3.6)$$

a całe wyrażenie skraca się do  $\min \{2^{n+1} + 2, 2^{n+1} + 2\} = 2^{n+1} + 2$ , podczas gdy optymalną wartością dla problemu MIN-MAX SPANNING TREE dla tych scenariuszy jest  $v(x^*) = v^* = 4$ , gdzie  $x^* = [0, 0, 0, 0, 1, 1, 1, 1]$  (zbiór  $E^* = \{e_5, e_6, e_7, e_8\}$ ). Uzyskaliśmy zatem błąd rzędu wykładniczego.

### 3.2.3 Sposoby aproksymacji problemu

Aby nie musieć uciekać się do rozwiązywania problemu MIN-MAX ze zbiorem scenariuszy  $S$  prosto z definicji tj.

- dla każdego dopuszczalnego rozwiązania  $x \in X$  wygenerować zbiór  $V_S^x = \{v(\mathbf{x}, \mathbf{s}) : \mathbf{s} \in S\}$  (zawierający wartości rozwiązań dla ustalonego wektora  $\mathbf{x}$  dla wszystkich scenariuszy w  $S$ ),
- z tak powstałego zbioru  $\mathcal{V}_S^X = \{\max_{y \in V_S^x} \{y\} : x \in X\}$  wybrać rozwiązanie  $x^* \in X$ , dla którego  $v(x^*, S) = \min_{z \in \mathcal{V}_S^X} \{z\}$ ,

możemy próbować aproksymować problem na podstawie jednego scenariusza, który jest składową wszystkich rozpatrywanych scenariuszy [11] [2, 430]. Przyjrzymy się dwóm podejściom do rozpatrywanego przez nas problemu, w obu przypadkach otrzymamy jego  $k$ -aproksymację, gdzie  $k$  będzie liczbą scenariuszy, jakie bierzemy pod uwagę. Choć będziemy skupiać się tylko na problemie MIN-MAX SPANNING TREE, przedstawione dowody da się z łatwością uogólnić na całą klasę problemów MIN-MAX.

#### Uśrednianie scenariuszy

Pierwszym, najprostszym do udowodnienia pomysłem jest potraktowanie problemu minimalizacyjnego  $\mathcal{P}$  ze zbiorem scenariuszy  $S$  ( $|S| = k$ ) jako problemu z pojedynczym scenariuszem, zdefiniowanym w sposób podany w poniższym twierdzeniu.

**Twierdzenie 3.2.1** Niech  $\mathcal{I}$  będzie instancją problemu DESCRIPTIVE MIN-MAX  $\mathcal{P}$ . Niech problem zawiera  $k$  scenariuszy w zbiorze  $S$  i niech każdy  $\mathbf{s} \in S$  będzie zdefiniowany jako  $\mathbf{s} = [c_1^s, \dots, c_m^s]$ . Dodatkowo niech  $\mathcal{I}'$  będzie instancją problemu  $\mathcal{P}$  z jednym scenariuszem  $\mathbf{s}'$ , którego koszty są średnią kosztów scenariuszy  $\mathbf{s} \in S$ . Wtedy, jeżeli istnieje optymalne rozwiązanie  $\mathbf{x}'$  dla  $\mathcal{I}'$ , to  $\max_{\mathbf{s} \in S} v(\mathbf{x}', \mathbf{s}) \leq \text{opt}(\mathcal{I})$ , gdzie  $\text{opt}(\mathcal{I})$  oznacza optymalną wartość rozwiązania dla  $\mathcal{I}$ .

**Dowód.** Koszty scenariusza  $\mathbf{s}'$  dla  $\mathcal{I}'$  są zdefiniowane następująco:

$$c'_i = \sum_{s=1}^k \frac{c_i^s}{k} \quad \forall i \in \{1, \dots, m\}. \quad (3.7)$$

<sup>4</sup>Z oznaczeniem  $v(\mathbf{s}, \mathbf{x})$  spotkaliśmy się już wcześniej przy omawianiu tabeli 3.1, gdzie wyrażenie to oznaczało całkowity koszt dopuszczalnego rozwiązania problemu optymalizacyjnego dla zadanego scenariusza  $\mathbf{s}$  oraz wybranego zbioru krawędzi reprezentowanego przez wektor  $\mathbf{x}$ . Analogicznie poprzez  $v(\mathbf{s}, x_{\mathbf{s}}^*)$  oznaczać będziemy koszt **optymalnego** rozwiązania dla scenariusza  $\mathbf{s}$ , gdzie  $x_{\mathbf{s}}^*$  to binarny wektor reprezentujący optymalne rozwiązanie dla danego scenariusza (często będziemy skracać ten zapis do  $v_{\mathbf{s}}^*$ ).



Rozpisując wyrażenie  $v(\mathbf{x}, \mathbf{s}') = \sum_{e \in E} x_e \cdot c_e^{\mathbf{s}'}$  otrzymujemy  $\sum_{e \in E} x_e \cdot \left( \sum_{s=1}^k \frac{c_e^s}{k} \right) = \frac{1}{k} \cdot \sum_{s=1}^k \sum_{e \in E} x_e \cdot c_e^s = \frac{1}{k} \sum_{s \in S} v(\mathbf{x}, \mathbf{s})$ . Pokażemy, że optymalne rozwiązanie  $\mathbf{x}'$  dla scenariusza  $\mathbf{s}$ , nie jest gorsze od optymalnego rozwiązania oryginalnego problemu (w przeciwnym wypadku rozwiązanie takiego scenariusza byłoby bezcelowe, gdyż nawet jego optymalne rozwiązanie było by gorsze od pożądanego). Mamy zatem:

$$L = v(\mathbf{x}', \mathbf{s}') = \min_{\mathbf{x} \in X} v(\mathbf{x}, \mathbf{s}') = \min_{\mathbf{x} \in X} \frac{1}{k} \cdot \sum_{s \in S} v(\mathbf{x}, \mathbf{s}) \leq \min_{\mathbf{x} \in X} \frac{1}{k} \cdot k \cdot \max_{s \in S} v(\mathbf{x}, \mathbf{s}) = \min_{\mathbf{x} \in X} \max_{s \in S} v(\mathbf{x}, \mathbf{s}) = \text{opt}(\mathcal{I}). \quad (3.8)$$

W czasie obliczeń skorzystaliśmy z własności  $\sum_{s \in S} v(\mathbf{x}, \mathbf{s}) \leq k \cdot \max_{s \in S} v(\mathbf{x}, \mathbf{s})$  (liczba scenariuszy  $|S| = k$ ). Wartość  $L$  nazywamy **dolnym ograniczeniem**. **Górnym ograniczeniem** na wartość rozwiązania jest oczywiście  $\max_{s \in S} v(\mathbf{x}', s) = U$ . Próbując ograniczać otrzymaną wartość  $U$  otrzymamy:

$$\text{opt}(\mathcal{I}) = \min_{\mathbf{x} \in X} \max_{s \in S} v(\mathbf{x}, \mathbf{s}) \leq \max_{s \in S} v(\mathbf{x}', s) \leq \sum_{s \in S} v(\mathbf{x}', s) = k \cdot \frac{1}{k} \cdot \sum_{s \in S} v(\mathbf{x}', s) = k \cdot L \leq k \cdot \text{opt}(\mathcal{I}), \quad (3.9)$$

gdzie w dwóch ostatnich równościach skorzystaliśmy z tego, że  $L = \min_{\mathbf{x} \in X} \frac{1}{k} \cdot \sum_{s \in S} v(\mathbf{x}, \mathbf{s}) = \frac{1}{k} \sum_{s \in S} v(\mathbf{x}', s)$  oraz, że wcześniej udowodniliśmy nierówność  $L \leq \text{opt}(\mathcal{I})$ . Pokazaliśmy zatem, że  $\max_{s \in S} v(\mathbf{x}', s) \leq k \cdot \text{opt}(\mathcal{I})$ , co oznacza, że wybierając optymalne rozwiązanie  $\mathbf{x}'$  dla scenariusza  $\mathbf{s}'$ , w najgorszym możliwym przypadku (wystąpienia takiego scenariusza, że wartość rozwiązania dla wybranego  $\mathbf{x}'$  będzie najbardziej oddalona od minimum) otrzymane rozwiązanie nie będzie gorsze niż  $k$ -krotność wartości optymalnej. ♦

### Scenariusz najgorszych wartości

Scenariuszem najgorszych wartości będziemy nazywać taki scenariusz, którego każdy koszt, który definiuje, jest największym z możliwych kosztów scenariuszy, z których ten scenariusz powstaje. Innymi słowy, powracając do rysunku 3.4, jeżeli mamy dwa scenariusze  $\mathbf{s}_6 = [0, 0, 2^n, 2^n, 1, 1, 1, 1]$  oraz  $\mathbf{s}_6 = [2^n, 2^n, 0, 0, 1, 1, 1, 1]$ , to scenariuszem najgorszych wartości będzie scenariusz  $\mathbf{s}' = [2^n, 2^n, 2^n, 2^n, 1, 1, 1, 1]$  — tak samo jak w przypadku ciągłym, gdy najgorszy z możliwych scenariuszy był definiowany za pomocą  $\bar{s}$ , tak wszystkie koszty scenariusza  $\mathbf{s}'$  są nie mniejsze niż odpowiadające im wartości w scenariuszach pozostałych. Co więcej, jeżeli wykorzystalibyśmy tak stworzony scenariusz  $\mathbf{s}'$  do rozwiązania problemu przedstawionego na ilustracji 3.4, otrzymalibyśmy rozwiązanie optymalne, podobnie jak by to miało miejsce w przypadku ciągłym. Pokażemy jednak, że — pomimo obiecującego wyniku — taka próba podejścia do problemów MIN-MAX nie zapewnia nam optymalnego rozwiązania, a co najwyżej jego  $k$ -aproksymację, podobnie jak poprzednio zaprezentowana metoda.

**Twierdzenie 3.2.2** Niech  $\mathcal{I}$  będzie instancją problemu DESCRIPTIVE MIN-MAX  $\mathcal{P}$ . Niech problem zawiera  $k$  scenariuszy w zbiorze  $S$  i niech każdy  $\mathbf{s} \in S$  będzie zdefiniowany jako  $\mathbf{s} = [c_1^s, \dots, c_m^s]$ . Dodatkowo niech  $\mathcal{I}'$  będzie instancją problemu  $\mathcal{P}$  z pojedynczym scenariuszem najgorszych wartości  $\mathbf{s}'$ . Wtedy, jeżeli istnieje optymalne rozwiązanie  $\mathbf{x}'$  dla  $\mathcal{I}'$ , to  $\max_{s \in S} v(\mathbf{x}', \mathbf{s}) \leq \text{opt}(\mathcal{I})$ , gdzie  $\text{opt}(\mathcal{I})$  oznacza optymalną wartość rozwiązania dla  $\mathcal{I}$ .

**Dowód.** Koszty scenariusza  $\mathbf{s}'$  dla  $\mathcal{I}'$  są zdefiniowane następująco:

$$c'_i = \max_{\mathbf{s} \in S} c_i^{\mathbf{s}} \quad \forall i \in \{1, \dots, m\}. \quad (3.10)$$

Pokażemy ciąg przekształceń, który da nam górne oszacowanie na wartość wyrażenia  $\max_{s \in S} v(\mathbf{x}', \mathbf{s})$ . Niech  $\mathbf{x}^*$  będzie optymalnym rozwiązaniem oryginalnego problemu  $\mathcal{I}$ :

$$\max_{s \in S} v(\mathbf{x}', \mathbf{s}) \leq \sum_{i=1}^m c'_i \cdot x'_i \leq \sum_{i=1}^m c'_i \cdot x_i^* \leq \sum_{i=1}^m \sum_{s \in S} c_i^s \cdot x_i^* = \sum_{s \in S} \sum_{i=1}^m c_i^s \cdot x_i^* \leq k \cdot \max_{s \in S} \sum_{i=1}^m c_i^s \cdot x_i^* = k \cdot \text{opt}(\mathcal{I}). \quad (3.11)$$

Chcąc dokładniej przyjrzeć się przeprowadzonemu przez nas rozumowaniu:



- $\max_{\mathbf{s} \in S} v(\mathbf{x}', \mathbf{s}) = \sum_{i=1}^m c_i^{\mathbf{s}} \cdot x'_i \leq \sum_{i=1}^m c'_i \cdot x'_i$  otrzymujemy prosto z definicji kosztów scenariusza  $s'$  — dla każdego  $i \in \{1, \dots, m\}$  zachodzi  $c_i^{\mathbf{s}} \leq \max_{\mathbf{s} \in S} c_i^{\mathbf{s}} = c'_i$ , zaś reszta wyrażenia nie ulega zmianie,
- $\sum_{i=1}^m c'_i \cdot x'_i \leq \sum_{i=1}^m c'_i \cdot x^*_i$  — rozwiązanie opisywane przez wektor  $\mathbf{x}'$  jest optymalny dla scenariusza z kosztami  $c'_i$  zatem każde inne rozwiązanie jest nie lepsze od niego ( $\mathbf{x}^*$  jest optymalne dla kosztów oryginalnych, nie dla tych, które występują w nierówności),
- $\sum_{i=1}^m c'_i \cdot x^*_i \leq \sum_{i=1}^m \sum_{\mathbf{s} \in S} c_i^{\mathbf{s}} \cdot x^*_i$  w oczywisty sposób wynika z faktu, że dla każdego  $i \in \{1, \dots, m\}$  zachodzi  $c'_i = \max_{\mathbf{s} \in S} c_i^{\mathbf{s}} \leq \sum_{\mathbf{s} \in S} c_i^{\mathbf{s}}$ ,
- $\max_{\mathbf{s} \in S} \sum_{i=1}^m c_i^{\mathbf{s}} \cdot x^*_i$  jest równe wartości rozwiązań dla najgorszego możliwego przypadku, zakładając że podstawiane do wzoru rozwiązanie  $\mathbf{x}^*$  jest optymalne —  $opt(\mathcal{I})$ .

Pokazaliśmy zatem, tak samo jak w przypadku poprzedniego twierdzenia, że takie podejście do problemu MIN-MAX zapewnia nam jedynie  $k$ -aproksymację rozwiązania. ♦

### 3.3 Problemy minimaksowe ze względną funkcją optymalności

W poprzednim dziale rozważaliśmy problem, który w naturalnym języku wyraża się pytaniem „Jakie wybrać rozwiązanie aby w najgorszym przypadku było ono najlepsze z możliwych?”. Skupiało się zatem ono tylko na niwelowaniu strat, których można było się spodziewać, nie zaś na faktycznym szukaniu rozwiązania, które było by możliwe blisko optymalnemu. W tym dziale zmienimy nieco nasze podejście do postawionego problemu i zadamy sobie pytanie: „Jakie rozwiązanie wybrać, aby w najgorszym przypadku było ono na tyle blisko rozwiązania optymalnego jak to tylko możliwe?”, Aby to osiągnąć, wprowadzimy pojęcie **żalu** (ang. *regret*). Niech  $v_s^*$  oznacza wartość optymalnego rozwiązania dla scenariusza  $\mathbf{s}$ . Wektorem zapewniającym to rozwiązanie optymalne jest wektor  $x_s^*$ . Równocześnie oznacza to, że dla każdego innego wektora  $\mathbf{x} \neq x_s^*$   $v(\mathbf{x}, \mathbf{s}) \geq v(x_s^*, \mathbf{s})$ . Różnicę tych dwóch wartości nazywamy żalem i reprezentuje on strategię jaką ponieśliśmy w konsekwencji wyboru innego rozwiązania niż optymalne dla danego scenariusza. Istotą problemu MIN-MAX REGRET jest wybór takiego rozwiązania  $\mathbf{x}$  aby zminimalizować tę różnicę dla wszystkich scenariuszy, w szczególności dla tego najgorszego, bo to od niego zależy wynik poniższego wyrażenia:

$$\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} v(\mathbf{x}, \mathbf{s}) - v(x_s^*, \mathbf{s}). \quad (3.12)$$

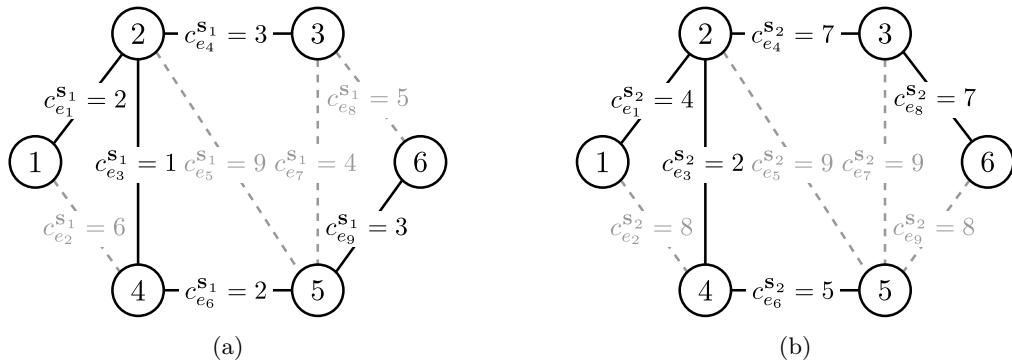
Możemy także rozpatrywać przypadek MAX-MIN REGRET, którego definicja nieco różni się od tej podanej wyżej i nie jest analogiczna do problemu MAX-MIN:  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} v(x_s^*, \mathbf{s}) - v(\mathbf{x}, \mathbf{s})$ . Tak samo jak w przypadku poprzednich problemów, tutaj też możemy wyróżnić podział na scenariusze ciągłe oraz dyskretnie.

#### 3.3.1 Przypadek dyskretny

Tym razem zaczniemy od omówienia przypadku dyskretnego, gdyż jak się będziemy mogli przekonać — wiele z tego, co do tej pory powiedzieliśmy o problemach MIN-MAX, powtórzy się dla problemów MIN-MAX REGRET. Podobnie jak to miało miejsce w przypadku poprzedniej klasy problemów, zaczniemy od omówienia prostego przykładu, który przedstawia rysunek 3.5, na którym zaznaczono optymalne rozwiązania dla problemu MIN-MAX REGRET MINIMUM SPANNING TREE dla dwóch scenariuszy  $s_1$  oraz  $s_2$ , natomiast w tabeli 3.2 — 7 przykładowych wyników, jakie daje zastosowanie różnych drzew rozpinających. Już dwa pierwsze rozwiązania:  $\mathbf{x}_1$  oraz  $\mathbf{x}_2$  wskazują na to, że kierowanie się przy wyborze rozwiązania dla tego problemu tylko kryterium optymalności dla pojedynczego scenariusza jest błędne i nie należy go stosować; obydwa rozwiązania są optymalne odpowiednio dla scenariuszy  $s_1$  oraz  $s_2$ , lecz nie dają tego samego rezultatu przy rozpatrywaniu właściwego problemu.

#### Aproksymacja rozwiązania

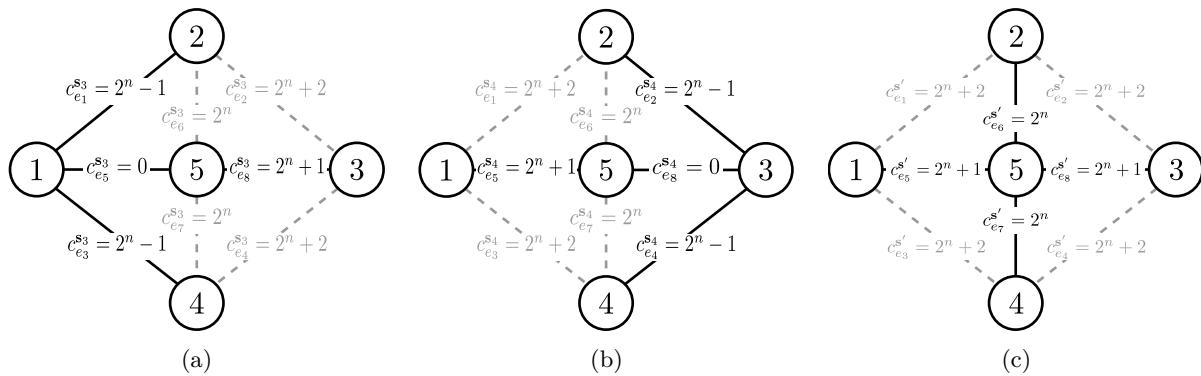
Chcąc otrzymać aproksymację dla problemu MIN-MAX REGRET, możemy odwołać się do udowodnionego twierdzenia 3.2.1 — jego dowód przebiega analogicznie dla tej klasy problemów [2, 430] i zostanie tu pominięty.



Rysunek 3.5: Dyskretne scenariusze  $s_1, s_2$  dla grafu nieskierowanego  $G = (V, E)$ ,  $V = \{1, 2, \dots, 6\}$ ,  $E = \{e_i : i = 1, \dots, 9\}$ . **(a)** Optymalnym rozwiązaniem dla scenariusza  $s_1$  jest zbiór wierzchołków  $E_{s_1}^* = \{e_1, e_3, e_4, e_6, e_9\}$ , któremu odpowiada wektor  $\mathbf{x}_{s_1}^* = [1, 0, 1, 1, 0, 1, 0, 0, 1]$ . Wartość tego rozwiązania wynosi  $v_{s_1}^* = 11$ . **(b)** Optymalnym rozwiązaniem dla scenariusza  $s_2$  jest zbiór wierzchołków  $E_{s_2}^* = \{e_1, e_3, e_4, e_6, e_8\}$ , któremu odpowiada wektor  $\mathbf{x}_{s_2}^* = [1, 0, 1, 1, 0, 1, 0, 1, 0]$ . Wartość tego rozwiązania wynosi  $v_{s_2}^* = 25$ .

X	Scenariusze													
	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$e_8$	$e_9$	$v(\mathbf{x}, \mathbf{s}_1)$	$r_{\mathbf{s}_1}$	$v(\mathbf{x}, \mathbf{s}_2)$	$r_{\mathbf{s}_2}$	$\max\{r_{\mathbf{s}_1}, r_{\mathbf{s}_2}\}$
$\mathbf{x}_1$	1	0	1	1	0	1	0	0	1	11	0	26	1	1
$\mathbf{x}_2$	1	0	1	1	0	1	0	1	0	13	2	25	0	2
$\mathbf{x}_3$	1	0	1	0	0	1	0	1	1	13	2	26	1	2
$\mathbf{x}_4$	1	0	1	0	0	1	1	0	1	12	1	28	3	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
$\mathbf{x}_{53}$	0	1	1	0	1	0	1	1	0	25	14	35	10	14
$\mathbf{x}_{54}$	0	1	0	0	1	1	1	1	0	26	15	38	13	15
$\mathbf{x}_{55}$	1	1	0	0	1	0	1	1	0	26	15	37	12	15

Tablica 3.2: Tabela przedstawiająca część z osiągalnych rozwiązań dla problemu minimalnego drzewa rozpinającego dla scenariuszy  $s_1$  oraz  $s_2$  (3.5a i 3.5b), koszty dla każdego z proponowanych rozwiązań dla podanych scenariuszy, poniesione straty względem optymalnych rozwiązań oraz wartość rozwiązania dla problemu MIN-MAX REGRET. Wiersze w tabeli zostały posortowane w kolejności rosnących wartości rozwiązań.



Rysunek 3.6: Dyskretne scenariusze  $s_3, s_4$  i  $s'$  dla grafu nieskierowanego  $G = (V, E)$ ,  $V = \{1, 2, \dots, 5\}$ ,  $E = \{e_i : i = \{1, \dots, 8\}\}$ , gdzie  $s'$  jest sztucznie wygenerowanym scenariuszem najgorszych kosztów. (a) Optymalne rozwiązanie  $E_{s_3}^* = \{e_1, e_3, e_5, e_8\}$  o całkowitym koszcie:  $3 \cdot 2^n - 1$ . (b) Optymalne rozwiązanie  $E_{s_4}^* = \{e_2, e_4, e_5, e_8\}$  o identycznym koszcie, co  $v_{s_3}^*$ . (c) Optymalne rozwiązanie dla scenariusza  $s'$  o kosztach  $c_i^{s'} = \max_{s \in S} c_s^i$ . Koszt całkowity rozwiązania optymalnego wynosi  $v(x_{s'}^*, s') = 4 \cdot 2^n + 2$ .

Dużo ciekawszy okazuje się natomiast przypadek, w którym tworzyliśmy dodatkowy scenariusz  $s'$ , którego koszty definiowaliśmy jako  $c'_i = \max_{\mathbf{s} \in S} c_i^{\mathbf{s}} \forall i \in \{1, \dots, m\}$ . Pokażemy, że nawet dla dwóch scenariuszy (gdzie spodziewalibyśmy się wyniku co najwyżej 2 razy gorszego niż optymalny) otrzymany rezultat jest dalece gorszy od spodziewanego. W tym celu stworzymy dwa scenariusze:  $\mathbf{s}_3 = [2^n - 1, 2^n + 2, 2^n - 1, 2^n + 2, 0, 2^n + 1]$  oraz  $\mathbf{s}_4 = [2^n + 2, 2^n - 1, 2^n + 2, 2^n - 1, 2^n + 1, 0]$ , tak jak na rysunku 3.6. Będziemy chcieli pokazać, że dla takiej konfiguracji wartość optymalna dla problemu MIN-MAX REGRET wynosi 0, toteż każda inna wartość, jaką otrzymamy przy rozwiązywaniu problemu z wykorzystaniem scenariusza  $s'$ , różna od zera będzie błędna (zgodnie z twierdzeniem 3.2.2  $\max_{\mathbf{s} \in S} v(\mathbf{x}', \mathbf{s}) \leq k \cdot \text{opt}(\mathcal{I})$ , zatem w przypadku, gdy  $\text{opt}(\mathcal{I}) = 0$ , gdzie  $\mathcal{I}$  to instancja oryginalnego problemu, jedyne optymalne rozwiązanie  $\mathbf{x}'$  dla scenariusza  $s'$ , które spełnia tą nierówność, także musi mieć wartość równą 0).

Jak łatwo zauważyc, koszty obydwu scenariuszy dla problemu zostały tak dobrane, aby obydwa optymalne rozwiązania dla każdego scenariusza nie różniły się pod względem wartości, która wynosi  $v_{s_3}^* = v_{s_43}^* = 3 \cdot 2^n - 1$ . Jest to o tyle istotne, że podstawiając te dane do ogólnego wzoru otrzymujemy:

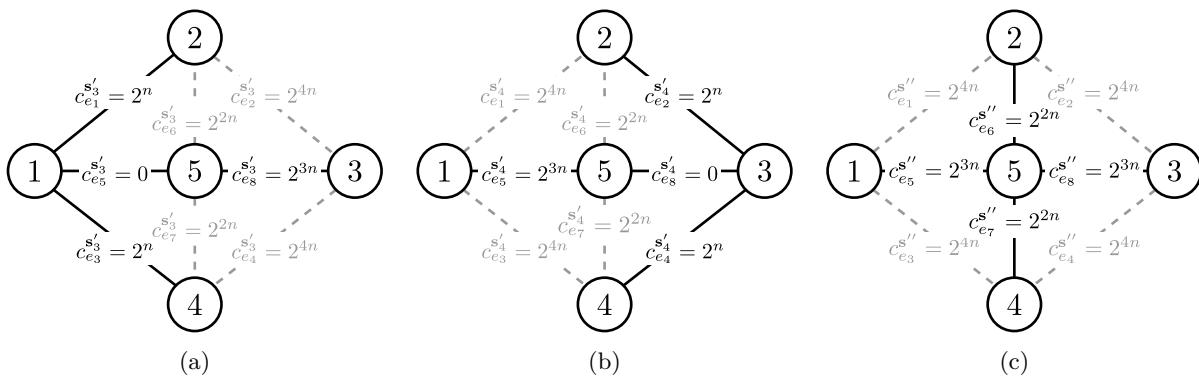
$$\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in \{\mathbf{s}_3, \mathbf{s}_4\}} v(\mathbf{x}, \mathbf{s}) - v(\mathbf{x}_s^*, \mathbf{s}) = \min \left\{ \max_{\mathbf{s} \in \{\mathbf{s}_3, \mathbf{s}_4\}} v(\mathbf{x}_{\mathbf{s}_3}^*, \mathbf{s}) - v(\mathbf{x}_s^*, \mathbf{s}), \max_{\mathbf{s} \in \{\mathbf{s}_3, \mathbf{s}_4\}} v(\mathbf{x}_{\mathbf{s}_4}^*, \mathbf{s}) - v(\mathbf{x}_s^*, \mathbf{s}) \right\}. \quad (3.13)$$

Na tym etapie możemy zauważyc, że bez względu na to, które rozwiązań wybierzymy (czy  $\mathbf{x}_{s_3}^*$ , czy  $\mathbf{x}_{s_4}^*$ ), dla oby rozpatrywanych scenariuszy otrzymamy identyczną wartość żalu równą零. Podstawiając dane do wzoru 3.13 otrzymamy zatem:  $\min \{\max \{0, 0\}, \max \{0, 0\}\} = 0$ . Innych rozwiązań nie rozpatrywaliśmy, gdyż z samej definicji problemu MIN-MAX REGRET wynika, że  $\forall x \in X_v(\mathbf{x}, \mathbf{s}) - v(\mathbf{x}_s^*, \mathbf{s}) \geq 0$ , toteż wartości otrzymane dla pozostałych rozwiązań z pewnością nie mogłyby być lepsze.

Zakładając, że analogiczne twierdzenie dla MIN-MAX REGRET do 3.2.2 jest prawdziwe, spodziewamy się, że wartość optymalnego rozwiązania dla dodatkowego scenariusza  $s'$  będzie nie gorsza niż  $k$ -krotność powyżej otrzymanego rozwiązania (będzie równa 0). Z rysunku 3.6c natomiast jednoznacznie wynika, że rozwiązaniem dla tego scenariusza jest zbiór krawędzi  $E_{s'}^* = \{e_5, e_6, e_7, e_8\}$  o całkowitym koszcie równym  $2^{n+2} + 1$ , zaś po podstawieniu danych do wzoru otrzymamy:

$$\begin{aligned} & \min_{\mathbf{x} \in \left\{ \mathbf{x}_{\mathbf{s}'}^* \right\}} \max_{\mathbf{s} \in \{\mathbf{s}_3, \mathbf{s}_4\}} v(\mathbf{x}, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}}^*, \mathbf{s}) = \max_{\mathbf{s} \in \{\mathbf{s}_3, \mathbf{s}_4\}} v(x_{\mathbf{s}'}^*, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}}^*, \mathbf{s}) = \\ &= \max \{v(x_{\mathbf{s}'}^*, \mathbf{s}_3), v(x_{\mathbf{s}'}^*, \mathbf{s}_4)\} - 3 \cdot 2^n - 1 = \max \{3 \cdot 2^n + 1, 3 \cdot 2^n + 1\} - 3 \cdot 2^n - 1 = 2. \end{aligned}$$

Wynikiem możemy dowolnie manipulować — przykładem takiej manipulacji są scenariusze  $s'_3$ ,  $s''_3$  oraz  $s''$  przedstawione (wraz z optymalnymi rozwiązaniami dla każdego z osobna) na rysunkach 3.7a, 3.7b i 3.7c, gdzie optymalną wartością rozwiązania problemu MIN-MAX REGRET dla tych pierwszych jest ponownie wartość 0, zaś dla trzeciego z nich:  $2^{n+2} \cdot (2^n - 1)$ .



Rysunek 3.7: Scenariusze  $s'_3$  i  $s'_4$ , dla których scenariusz najgorszych kosztów  $s''$  daje bardzo zle rozwiązanie. **(a)** Minimalne drzewo rozpinające dla scenariusza  $s'_3$  o całkowitym koszcie  $v_{s'_3}^* = 2^{3n} + 2 \cdot 2^n$ . **(b)** Minimalne drzewo rozpinające dla scenariusza  $s'_3$  o takim samym koszcie. **(c)** Optymalne drzewo rozpinające dla scenariusza  $s''$ .

Pokazaliśmy zatem, że w tym przypadku nie możemy zastosować analogicznego twierdzenia.



### 3.3.2 Przypadek ciągły

Zacznijmy od następującego twierdzenia:

**Twierdzenie 3.3.1** Niech dany będzie problem minimalizacyjny  $\mathcal{P}$ . Niech  $\underline{s}$  i  $\bar{s}$  będą krytycznymi scenariuszami (odpowiednio najgorszym i najlepszym) definiującymi przestrzeń scenariuszy  $S$ . Wartość żalu dla dowolnego rozwiązania  $\mathbf{x} \in X$  jest największa dla, zależnego od  $\mathbf{x}$ , scenariusza  $s^-(\mathbf{x})$ , którego koszty są zdefiniowane następująco:

$$c_i^{s^-}(\mathbf{x}) = \begin{cases} c_i^{\bar{s}} & \text{gdy } e_i \text{ należy do rozwiązania } (x_i = 1), \\ c_i^{\underline{s}} & \text{w przeciwnym przypadku } (x_i = 0), \end{cases} \quad \forall i \in \{1, 2, \dots, m\}. \quad (3.14)$$

**Dowód.** Dla danego rozwiązania  $\mathbf{x} \in X$ , niech będzie dany zbiór  $\mathbb{1}(\mathbf{x}) = \{i : i \in \{1, \dots, m\} \wedge x_i = 1\}$ . Wartość żalu dla scenariusza  $\mathbf{s} \in S$  i dowolnego rozwiązania  $\mathbf{x}$  wynosi:

$$\begin{aligned} R(\mathbf{x}, \mathbf{s}) &= v(\mathbf{x}, \mathbf{s}) - v_{\mathbf{s}}^* \stackrel{(1)}{=} \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}} \stackrel{(2)}{\leqslant} \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\bar{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\underline{s}} \stackrel{(3)}{=} \\ &\stackrel{(3)}{=} v(\mathbf{x}, s^-(\mathbf{x})) - v(\mathbf{x}_s^*, s^-(\mathbf{x})) \stackrel{(4)}{\leqslant} v(\mathbf{x}, s^-(\mathbf{x})) - v(\mathbf{x}_{s^-(\mathbf{x})}, s^-(\mathbf{x})) = \\ &= v(\mathbf{x}, s^-(\mathbf{x})) - v_{s^-(\mathbf{x})}^* = R(\mathbf{x}, s^-(\mathbf{x})), \end{aligned}$$

co dowodzi poprawności twierdzenia. Aby lepiej zrozumieć ciąg logicznego rozumowania jaki został podjęty, skupimy się teraz na kilku, kluczowych dla dowodu, przekształceniach.

- (1) Niech  $\emptyset(\mathbf{x}) = \{i : i \in \{1, \dots, m\} \wedge x_i = 0\}$ . Łatwo zauważyc, że  $\emptyset(\mathbf{x})$  i  $\mathbb{1}(\mathbf{x})$  są rozłączne, zaś ich suma  $\emptyset(\mathbf{x}) \cup \mathbb{1}(\mathbf{x})$  generuje cały zbiór  $\{i : i \in \{1, \dots, m\}\}$ . Rozpisując wzór na wartość rozwiązania  $\mathbf{x}$  dla danego scenariusza  $\mathbf{s}$  mamy:  $v(\mathbf{x}, \mathbf{s}) = \sum_{e_i \in E} x_i \cdot c_i^{\mathbf{s}} = \sum_{i \in (\emptyset \cup \mathbb{1})(\mathbf{x})} x_i \cdot c_i^{\mathbf{s}} = \sum_{i \in \emptyset(\mathbf{x})} 0 \cdot c_i^{\mathbf{s}} + \sum_{i \in \mathbb{1}(\mathbf{x})} 1 \cdot c_i^{\mathbf{s}} = \sum_{i \in \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}}$ . Analogicznie możemy postąpić dla  $v_{\mathbf{s}}^* = v(\mathbf{x}_s^*, \mathbf{s}) = \sum_{i \in \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}}$ . Zauważmy, że w wyrażeniu  $v(\mathbf{x}, \mathbf{s}) - v_{\mathbf{s}}^*$  wszystkie elementy sumy, które należą do obu zbiorów, zredukują się tj.

$$\begin{aligned} v(\mathbf{x}, \mathbf{s}) - v_{\mathbf{s}}^* &= \sum_{i \in \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} = \left( \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} + \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}) \cap \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} \right) - \sum_{i \in \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}} = \\ &= \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}}. \end{aligned}$$

- (2) Łatwo zauważyc, że suma po prawej stronie nierówności jest większa ze względu na definicje kosztów scenariuszy  $\bar{s}$  oraz  $\underline{s}$  ( $\sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\mathbf{s}} \leqslant \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\bar{s}}$  oraz  $\sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\mathbf{s}} \geqslant \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\underline{s}}$ ).

- (3) Przeprowadzone rozumowanie jest analogiczne do tego z punktu (2) — na uwagę zasługuje fakt, iż:

$$\sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{\bar{s}} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{\underline{s}} = \sum_{i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)} c_i^{s^-} - \sum_{i \in \mathbb{1}(\mathbf{x}_s^*) \setminus \mathbb{1}(\mathbf{x})} c_i^{s^-},$$

co bezpośrednio wynika z definicji scenariusza  $s^-(\mathbf{x})$  — dla wszystkich elementów pierwszej sumy (indeksowanej  $i \in \mathbb{1}(\mathbf{x}) \setminus \mathbb{1}(\mathbf{x}_s^*)$ ) wartości kosztów dla tych elementów  $x_i$  przyjmują wartość  $c_i^{\bar{s}}$  ( $x_i = 1$ ). Analogicznie druga suma jest indeksowana po  $i$ , które w szczególności nie należą do zbioru  $\mathbb{1}(\mathbf{x})$ , tak więc dla każdego elementu tej sumy spełniony jest warunek  $x_i = 0$ . Poczyniwszy tę obserwację, dalsze przekształcanie formuły odbywa się identycznie jak przedstawiono w kroku (2).

- (4) Wynika bezpośrednio z właściwości optymalnego rozwiązania  $\mathbf{x}_{s^-(\mathbf{x})}^*$  dla scenariusza  $s^-(\mathbf{x})$ :

$$v(\mathbf{x}_s^*, s^-(\mathbf{x})) \geqslant v(\mathbf{x}_{s^-(\mathbf{x})}^*, s^-(\mathbf{x})) = v_{s^-(\mathbf{x})}^*. \quad \blacklozenge$$



Udowodniliśmy zatem, że dla dowolnego rozwiązania  $\mathbf{x} \in X$  wartość  $R(\mathbf{x}, \mathbf{s}) = v(\mathbf{x}, \mathbf{s}) - v_{\mathbf{s}}^*$  osiąga swoje maksimum dla  $\mathbf{s} = \mathbf{s}^-(\mathbf{x})$ . W oparciu o twierdzenie 3.3.1 możemy wyciągnąć następujący wniosek, od którego tylko krok dzieli nas od podania metody na rozwiązywanie zagadnienia INTERVAL MIN-MAX REGRET.

**Wniosek 3.3.1** *Dla problemu INTERVAL MIN-MAX REGRET  $\mathcal{P}$ , gdzie  $\mathcal{P}$  jest problemem minimalizacyjnym, optymalnym rozwiązaniem  $\mathbf{x}^*$  jest jedno z rozwiązań  $\mathbf{x} \in X$  dla scenariusza  $\mathbf{s}^-(\mathbf{x})$ .*

Znaczy to dla nas tyle, że zamiast rozwiązywać problem z definicji:  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s})$ , możemy równoważnie rozważyć problem  $\min_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}^-(\mathbf{x}))$ . Tak więc, jeśli istnieje optymalne rozwiązanie problemu  $\min_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}^-(\mathbf{x}))$ , jest ono także optymalnym rozwiązaniem dla  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s})$ .

Choć powyższe twierdzenie da się łatwo uogólnić na klasę problemów MIN-MAX REGRET (co też uczyniliśmy), pierwotny pomysł wywodzi się z rozważań nad problemem MIN-MAX REGRET SPANNING TREE [2, 429–430] [19], który szczególnie nas interesuje. Podobnie możemy postąpić z następującym twierdzeniem, którego potrzebujemy aby udowodnić prawidłowość w drugą stronę: że istnienie optymalnego rozwiązania dla problemu INTERVAL MIN-MAX REGRET pociąga za sobą istnienie optymalnego rozwiązania dla problemu minimalizacyjnego  $\mathcal{P}$  dla jednego z ekstremalnych scenariuszy. Udowodnijmy zatem:  $\exists \mathbf{x}^* = \min \arg_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s}) \Rightarrow \exists \mathbf{s}(\mathbf{x}) \in S : \min \arg_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}(\mathbf{x})) = \mathbf{x}^*$ .

**Twierdzenie 3.3.2** *Niech dany będzie problem minimalizacyjny  $\mathcal{P}$  i optymalne rozwiązanie problemu INTERVAL MIN-MAX REGRET  $\mathcal{P}$   $\mathbf{x}^*$ . Rozwiązanie optymalne  $\mathbf{x}^*$  odpowiada optymalnemu rozwiązaniu problemu  $\mathcal{P}$  dla co najmniej jednego scenariusza ekstremalnego, w szczególności dla  $\mathbf{s}^+(\mathbf{x}^*)$ , którego koszty są zdefiniowane następująco:*

$$c_i^{s^+}(\mathbf{x}^*) = \begin{cases} c_i^s & \text{gdy } e_i \text{ należy do optymalnego rozwiązania } (x_i^* = 1), \\ c_i^{\bar{s}} & \text{w przeciwnym przypadku } (x_i^* = 0), \end{cases} \quad \forall i \in \{1, 2, \dots, m\}. \quad (3.15)$$

Będziemy chcieli zatem udowodnić, że o ile istnieje optymalne rozwiązanie dla problemu INTERVAL MIN-MAX REGRET  $\mathcal{P}$ , musi istnieć co najmniej jeden ekstremalny scenariusz (jest nim  $\mathbf{s}^+(\mathbf{x}^*)$ ), który możemy wykorzystać do wygenerowania szukanego rozwiązania na podstawie twierdzenia 3.3.1 — samo udowodnienie 3.3.1 gwarantuje nam bowiem tylko następującą własność:  $\exists \mathbf{s}(\mathbf{x}) \in S : \min \arg_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}(\mathbf{x})) = \mathbf{x}^* \Rightarrow \exists \mathbf{x}^* = \min \arg_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s})$ . Dopiero gdy udowodnimy powyższe twierdzenie, będziemy mogli wymiennie stosować obydwie techniki, gdyż wtedy:  $\exists \mathbf{s}(\mathbf{x}) \in S : \min \arg_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}(\mathbf{x})) = \mathbf{x}^* \Leftrightarrow \exists \mathbf{x}^* = \min \arg_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s})$ .

**Dowód.** Niech  $\mathbf{x}^*$  będzie oznaczał optymalne rozwiązanie dla problemu INTERVAL MIN-MAX REGRET  $\mathcal{P}$  oraz niech  $\mathbb{1}(\mathbf{x}) = \{i : i \in \{1, \dots, m\} \wedge x_i = 1\}$ . Dla dowolnego scenariusza  $\mathbf{s} \in S$  mamy:

$$\begin{aligned} v(\mathbf{x}^*, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) &= \sum_{i \in \mathbb{1}(\mathbf{x}^*) \setminus \mathbb{1}(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*)} c_i^s - \sum_{i \in \mathbb{1}(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*) \setminus \mathbb{1}(\mathbf{x}^*)} c_i^s \geq \sum_{i \in \mathbb{1}(\mathbf{x}^*) \setminus \mathbb{1}(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*)} c_i^s - \sum_{i \in \mathbb{1}(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*) \setminus \mathbb{1}(\mathbf{x}^*)} c_i^{\bar{s}} = \\ &= v(\mathbf{x}^*, \mathbf{s}^+(\mathbf{x}^*)) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}^+(\mathbf{x}^*)). \end{aligned}$$

Analogiczny do powyższego proces przekształcania wyrażeń został wykorzystany przy dowodzie twierdzenia 3.3.1, tak więc nie będziemy jeszcze raz przytaczać uczynionych w nim spostrzeżeń. Co nas teraz interesuje to fakt pokazania, że dla optymalnego rozwiązania problemu INTERVAL MIN-MAX REGRET  $\mathcal{P}$   $\mathbf{x}^*$  i  $\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*$  nie istnieje inny scenariusz niż  $\mathbf{s}^+(\mathbf{x}^*)$ , którego wartość żalu byłaby mniejsza niż dla  $\mathbf{s}^+(\mathbf{x}^*)$ . Dodatkowo zauważmy, że jeśli  $\mathbf{x}^*$  jest rozwiązaniem optymalnym, to nie istnieje inne rozwiązanie  $\mathbf{x}$  takie, że  $v(\mathbf{x}, \mathbf{s}^+(\mathbf{x})) \leq v(\mathbf{x}^*, \mathbf{s}^+(\mathbf{x}^*))$ , w szczególności  $v(\mathbf{x}^*, \mathbf{s}^+(\mathbf{x}^*)) = v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}^+(\mathbf{x}^*))$  (wartość  $v_{\mathbf{s}^+(\mathbf{x}^*)}^*$  jest optymalna z definicji i nie może istnieć od niej inne rozwiązanie o mniejszym koszcie). Stąd dochodzimy do wniosku, że jeśli  $\mathbf{x}^*$  jest optymalnym rozwiązaniem dla INTERVAL MIN-MAX REGRET  $\mathcal{P}$ , to:  $v(\mathbf{x}^*, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) \geq v(\mathbf{x}^*, \mathbf{s}^+(\mathbf{x}^*)) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}^+(\mathbf{x}^*)) = 0$ .

Załóżmy teraz, że  $\mathbf{x}$  nie jest optymalnym rozwiązaniem problemu  $\mathcal{P}$  dla scenariusza  $\mathbf{s}^+(\mathbf{x}^*)$ . W związku z tym wyrażenie:  $v(\mathbf{x}^*, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s})$  staje się ostro większe od zera, stąd:



$$\begin{aligned} v(\mathbf{x}^*, \mathbf{s}) - v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) &> 0 \\ v(\mathbf{x}^*, \mathbf{s}) &> v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) \\ v(\mathbf{x}^*, \mathbf{s}) - v_{\mathbf{s}}^* &> v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) - v_{\mathbf{s}}^* \end{aligned}$$

dla dowolnego scenariusza  $\mathbf{s} \in S$ , więc:

$$\max_{\mathbf{s} \in S} v(\mathbf{x}^*, \mathbf{s}) - v_{\mathbf{s}}^* > \max_{\mathbf{s} \in S} v(\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^*, \mathbf{s}) - v_{\mathbf{s}}^*.$$

Stoi to w oczywisty sposób w sprzeczności z tym, że  $\mathbf{x}^*$  jest optymalnym rozwiązaniem problemu INTERVAL MIN–MAX REGRET  $\mathcal{P}$  (wybranie rozwiązania  $\mathbf{x}_{\mathbf{s}^+(\mathbf{x}^*)}^* \neq \mathbf{x}^*$  zagwarantowałoby mniejsze odchylenie od optymalnego rozwiązania w przypadku wystąpienia najgorszego scenariusza). ♦

Obydwa powyższe twierdzenia (3.3.1 oraz 3.3.2) pozwalają nam na efektywniejsze obliczenia w celu poszukiwania rozwiązania omawianego problemu. Jak już zdążyliśmy wspomnieć po zakończeniu dowodu pierwszego z wymienionych twierdzeń, zamiast szukać rozwiązania w oparciu o wzór:  $\min_{\mathbf{x} \in X} \max_{\mathbf{s} \in S} R(\mathbf{x}, \mathbf{s})$ , możemy zastąpić go tym:  $\min_{\mathbf{x} \in X} R(\mathbf{x}, \mathbf{s}^-(\mathbf{x}))$ . Problemem nadal oczywiście zostaje liczba ekstremalnych scenariuszy, która w wielu przypadkach może być ponad wielomianowa. Więcej na ten temat można przeczytać w [9], gdzie pokazano, że rozwiązanie problemu  $\mathcal{P}$  dla odpowiedniego scenariusza zapewnia nam dobre górne ograniczenia na wartość INTERVAL MIN–MAX REGRET  $\mathcal{P}$  oraz [3], gdzie bliżej przyjrzano się omawianemu przez nas podejściu dla stałej liczby ekstremalnych scenariuszy bądź ograniczonej w taki sposób, aby złożoność obydwu problemów była taka sama. W [2, 433–434] możemy zobaczyć podsumowanie stopnia złożoności szerokiej gamy problemów MIN–MAX oraz MIN–MAX REGRET dla obu przypadków (ciągłych oraz dyskretnych), dodatkowo osobno rozpatrywanych dla stałych liczb scenariuszy i dla ich liczby, która jest częścią wejścia (zmienna). Dla wszystkich przypadków problemu minimalnego drzewa rozpinającego uzyskujemy jednak stopień trudności NP–trudny, bądź NP–zupełny [11] [3].

## 3.4 Optymalizacja z możliwością poprawy

Dotychczas omówione zagadnienia ogólnie nazwaliśmy problemami **optymalizacji odpornej**, to jest niewrażliwej na niekorzystne dla siebie zmiany — bez względu na scenariusz, który okazywał się tym, który zaszedł w rzeczywistości, stosując podejście minimaxowe mogliśmy zapewnić sobie, że otrzymany wynik będzie zawsze ten sam, niezależnie od tego, czy zrealizowany scenariusz był tym najgorszym, czy definiował koszty w najlepszy dla nas sposób. Warunek był jeden — musieliśmy z wyprzedzeniem znać wszystkie możliwe scenariusze, by móc chociaż zacząć rozwiązać tego typu problemy (np. przy omawianiu problemu INTERVAL MIN–MAX lub INTERVAL MIN–MAX REGRET swoje rozwiązania opieraliśmy w głównej mierze na znajomości krytycznych wartości kosztów scenariuszy). W tej części omówimy problemy, które tych ograniczeń się pozbywają, bądź radzą sobie z nimi w sposób częściowy.

### 3.4.1 Problem przyrostowy

Pierwszym tego typu problemem, na jaki zwróciśmy uwagę, będzie **problem przyrostowy** (ang. *incremental problem*), z którym zetknimy się we wszystkich podejściach do problemu odpornej optymalizacji, jakie wymienimy w tej części. Swoją nazwę problem zawdzięcza swojej specyfice działania — tak jak w przypadku problemów z rodziny MIN–MAX mogliśmy odnieść wrażenie, że nasza odpowiedzialność za podjęcie optymalnej decyzji zaczyna się w momencie otrzymywania kompletnej informacji o wszystkich scenariuszach  $\mathbf{s} \in S$  i dozwolonych wyborach  $\mathbf{x} \in X$  i jednocześnie kończy się wraz z momentem ich przeanalizowania i zwróceniu rozwiązań, tak w tym przypadku możemy obserwować specyficzne „narastanie” problemu i stopniowego wzrostu skomplikowania podejmowanych decyzji. Założymy, że na podstawie posiadanych informacji o kosztach opisujących początkowy stan problemu  $\mathcal{P}$ , wyznaczyliśmy optymalne rozwiązanie  $\mathbf{x}^*$ . Niech po

pewnym czasie koszty te ulegną na tyle znaczącym zmianom, iż rozwiązywanie  $\mathbf{x}^{*''}$  przestanie być optymalne<sup>5</sup>. W ramach definicji problemu przyrostowego, pod wpływem pojawienia się nowych danych, możemy zmienić naszą decyzję, lecz nie może to być zupełnie nowe rozwiązanie — wybrany wektor  $\mathbf{x}^{*''}$  musi być w pewnym określonym stopniu podobny do rozwiązania poprzedniego  $\mathbf{x}^{*''}$ . Niech po pewnym czasie koszty te ulegną na tyle znaczącym zmianom, iż rozwiązywanie  $\mathbf{x}^{*''}$  przestanie być optymalne. Nasz wybór kolejnego rozwiązania podlega tym samym zasadom co poprzednim razem — widzimy więc, że rozwiązanie  $\mathbf{x}^{*''}$  zależy od początkowo wybranego rozwiązania  $\mathbf{x}^{*''}, \mathbf{x}^{*'''}$  zależy od dwóch poprzednich rozwiązań, każde kolejne od wszystkich swoich poprzedników (pośrednio lub nie). Aby formalnie zapisać problem INCREMENTAL, musimy wprowadzić kilka dodatkowych pojęć:

- $\mathbf{x}', \mathbf{x}'', \dots$  — oznaczają kolejne rozwiązania problemu, wybierane z **otoczenia** poprzedniego,
- $X_{\mathbf{x}}^k$  — **otoczenie** wektora  $\mathbf{x}$  charakteryzuje zbiór rozwiązań, z którego jesteśmy zobligowani wybrać nasze nowe rozwiązanie w przypadku chęci zmiany rozwiązania pierwotnego ( $\mathbf{x}$ ). Parametrem  $k$  będziemy oznaczać stopień w jakim dane dwa rozwiązania mogą się od siebie różnić; w ogólnym przypadku wartość ta będzie determinowała zbiór rozwiązań za pomocą specjalnej funkcji  $f : X \times X \rightarrow k$ , która dla dwóch wybranych rozwiązań zwracać będzie ich „odległość” od siebie. W przypadku rozwiązywania problemów grafowych, funkcja zwykle zwraca liczbę krawędzi, które nie należą do obu rozwiązań. Formalna definicja zbioru:  $X_{\mathbf{x}}^k = \{\mathbf{y} \in X : f(\mathbf{y}, \mathbf{x}) \leq k\}$ .

Naszym celem przy rozwiązywaniu problemu INCREMENTAL dla zadanego rozwiązania początkowego  $\mathbf{x}$  jest minimalizacja wyrażenia:

$$\min_{\mathbf{x}' \in X_{\mathbf{x}}^k} v(\mathbf{x}', \mathbf{s}'), \quad (3.16)$$

gdzie  $\mathbf{s}'$  reprezentuje nowe koszty, dla których poszukujemy optymalnego rozwiązania. Niech **miarą unikalności** rozwiązania będzie funkcja  $f(T', T) = |T' \setminus T|$ , gdzie  $T'$  oraz  $T$  są zbiorami krawędzi przedstawiającymi minimalne drzewo rozpinające odpowiednio dla scenariuszy  $\mathbf{s}'$  i  $\mathbf{s}$ , tak jak przedstawiono na rysunkach 3.8a oraz 3.8b. Dodatkowo niech  $k = 1$ . Zgodnie z definicją naszej funkcji  $f(T', T)$ , zbiorem dopuszczalnych rozwiązań dla tak zdefiniowanego problemu INCREMENTAL MINIMUM SPANNING TREE są wszystkie te zbiory krawędzi, które zawierają co najwyżej jeden element nie będący elementem zbioru krawędzi charakteryzującego początkowe rozwiązanie  $T$  (wyrażenie  $|T' \setminus T|$  oznacza **moc**<sup>6</sup> zbioru  $T' \setminus T$ , czyli liczbę wszystkich krawędzi, które należą do  $T'$ , zaś nie znajdują się w zbiorze  $T'$ ). Łatwo zauważyc, że w tym konkretnym przypadku zachodzi  $|T \setminus T'| = |T' \setminus T|$ , jako że obydwa zbiory są równoliczne (z definicji liczba ich elementów wynosi  $|V| - 1$ ) a ich elementy nie powtarzają się (np. zgodnie z rysunkiem 3.8:  $T = \{e_2, e_5, e_6, e_8, e_9\}$ ,  $T' = \{e_1, e_2, e_6, e_7, e_8\}$ ,  $|T \setminus T'| = |\{e_2, e_5, e_6, e_8, e_9\} \setminus \{e_1, e_2, e_6, e_7, e_8\}| = |\{e_5, e_9\}| = 2 = |\{e_1, e_7\}| = |\{e_1, e_2, e_6, e_7, e_8\} \setminus \{e_2, e_5, e_6, e_8, e_9\}| = |T' \setminus T|$ ). Niestety, podane przez nas nowe rozwiązanie  $T'$ , będące minimalnym drzewem rozpinającym dla scenariusza  $\mathbf{s}'$ , nie jest optymalnym rozwiązaniem w myśl definicji rozpatrywanego przez nas problemu INCREMENTAL MINIMUM SPANNING TREE — co więcej, nie jest ono nawet rozwiązaniem dopuszczalnym, jako że  $= f(T', T) \not\leq k = 1$ .

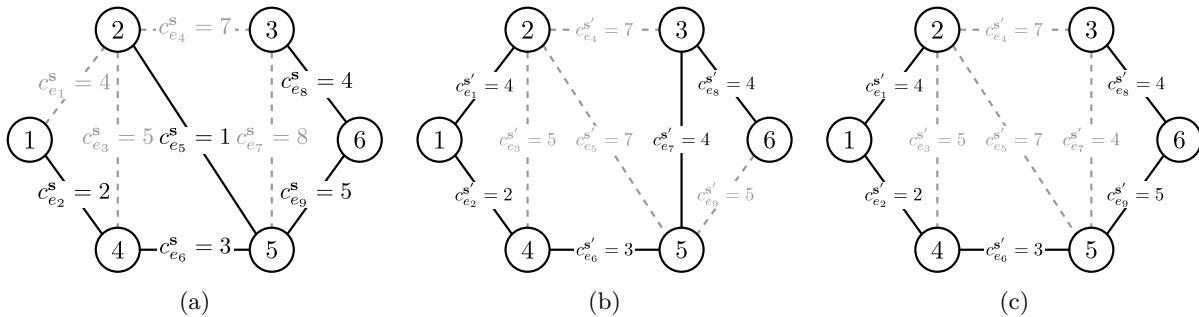
## Naiwne rozwiązanie

Aby otrzymać poprawne rozwiązanie  $T^*$  problemu INCREMENTAL MINIMUM SPANNING TREE —  $\mathcal{P}$ , spełniającego ograniczenie  $f(T^*, T) \leq 1$  — naturalnym wydaje się rozpoczęć jego konstrukcję od uzyskania zbioru krawędzi, który jest optymalny dla zadanego scenariusza ( $T'$  — niedopuszczalne w  $\mathcal{P}$ ), a następnie przekształcić je w rozwiązanie optymalne, tak jak to pokazano na rysunkach 3.8b i 3.8c — z uzyskanego zbioru krawędzi  $T'$ , będącego optymalnym rozwiązaniem dla scenariusza  $\mathbf{s}'$ , ze zbioru krawędzi nie należących do pierwotnego rozwiązania (będącego przyczyną jego niedopuszczalności,  $T' \setminus T = \{e_1, e_7\}$ ) usunięto krawędź o najwyższy koszcie ( $e_7$ ) i zastąpiono ją krawędzią o najniższym koszcie ze zbioru  $T \setminus T' = \{e_5, e_9\}$ . Tym samym otrzymywane rozwiązanie  $T''$  spełnia  $f(T', T) - 1 = f(T'', T) = k$ , zaś sposób jego konstrukcji gwarantuje nam jego optymalność, co w danym przypadku jest trywialne do zauważenia. Zatem  $T'' = T^*$  dla tak zdefiniowanego problemu  $\mathcal{P}$ .

<sup>5</sup> Warunek ten oczywiście nie musi zachodzić w ogólnym przypadku — gdybyśmy dopuścili taką sytuację, że po zmianie kosztów stare rozwiązanie nadal jest optymalne, niepotrzebne byłoby szukanie następnego rozwiązania.

<sup>6</sup> Liczbę elementów w zbiorze.

<sup>7</sup> Aby w pełni oddać charakter danego zbioru krawędzi jako zawierającego elementy minimalnego drzewa rozpinającego, będziemy korzystać z oznaczenia  $T$  w zamian  $E$ , tak jak to miało miejsce do tej pory.



Rysunek 3.8: Różnice między rozwiązaniami problemów MINIMUM SPANNING TREE i INCREMENTAL MST z parametrem  $k = 2$ . **(a)** Optymalne rozwiązanie problemu minimalnego drzewa rozpinającego dla scenariusza  $s = \{4, 2, 5, 7, 1, 3, 8, 4, 5\}$  o całkowitym koszcie rozwiązania  $v_{\text{MST}}(T, s) = 15$ . **(b)** Niedopuszczalne rozwiązanie  $T' = \{e_1, e_2, e_6, e_7, e_8\}$  problemu INCREMENTAL MINIMUM SPANNING TREE będące jednocześnie optymalnym dla scenariusza  $s'$  w problemie MST o koszcie  $v_{\text{MST}}(T, s') = 17$  **(c)** Optymalne rozwiązanie problemu INCREMENTAL MST i scenariusza  $s'$ :  $T^* = \{e_1, e_2, e_6, e_8, e_9\}$  o całkowitym koszcie  $v_{\text{S}}^* = 18$ .

Niestety wraz ze wzrostem grafu oraz przede wszystkim parametru  $k$ , rozwiązanie to staje się nieefektywne — jak mogliśmy zauważyć, opisany algorytm działa sekwencyjnie, co każdą iterację zmniejszając wartość funkcji  $f(T^i, T)$ , gdzie  $T^i$  to rozwiązanie uzyskane podczas  $i$ -tej iteracji. Ogólnie:  $f(T^{i-1}, T) - 1 = f(T^i, T)^8$ . Naszym zadaniem zatem jest konstrukcja takiego ciągu drzew rozpinających  $T^1, \dots, T^l$ , aby  $f(T^l, T^0) = k$ , gdzie  $T^0 = T$  i jest początkowym rozwiązaniem problemu  $\mathcal{P}$  dla pierwotnego scenariusza, zaś  $l = f(T^1, T^0) - k + 1$  ( $k = f(T^l, T^0) = f(T^{l-1}, T^0) - 1 = \dots = f(T^2, T^0) - (l-2) = f(T^1, T^0) - (l-1) = k$  — ogólnie  $f(T^{l-i}, T^0) - i = k$ ). Każde kolejne rozwiązanie powinno zatem składać się z coraz to większej liczby krawędzi należących do  $T^0$ . Aby to osiągnąć, dla każdej  $i$ -tej iteracji musielibyśmy rozpatrzyć wszystkie możliwe podziały drzewa  $T^{i-i}$  powstałe w wyniku usunięcia jednej z krawędzi  $e^{i-1} \in T^{i-1} \setminus T^0$  — innymi słowy wszystkie takie pary drzew  $(T_1^{i-1}, T_2^{i-1})$ , których łączna liczba krawędzi nie należących do drzewa  $T^0$  jest o 1 mniejsza niż dla drzewa  $T^{i-1}$ . Takich par w  $i$ -tej iteracji jest dokładnie  $f(T^1, T^0) - k - (i-1)$  (iteracje numerujemy rozpoczynając od  $i=1$ ). Dla każdej takiej pary musimy następnie znaleźć wszystkie krawędzie należące do zbioru  $\mathcal{Q}(T^{i-1}, e^{i-1}) \cap T^0$ , łączące ze sobą dane poddrzewa<sup>9</sup>, gdzie krawędź  $e^{i-1} \in T^{i-1}$  jest krawędzią usuwaną z  $T^{i-1}$ , nienależącą do  $T^0$ . Na samym końcu musimy stworzyć nowe drzewo, do którego dołączymy krawędź  $e' \in \mathcal{Q}(T^{i-1}, e^{i-1}) \cap T^0$  o najkorzystniejszym stosunku kosztów do usuniętej krawędzi. Formalnie:  $T^i = (\{e \in T^{i-1}\} \setminus e^{i-1}) \cup e' : e' = \min arg_{e'} \frac{c_{e'}}{c_{e^{i-1}}}$ .

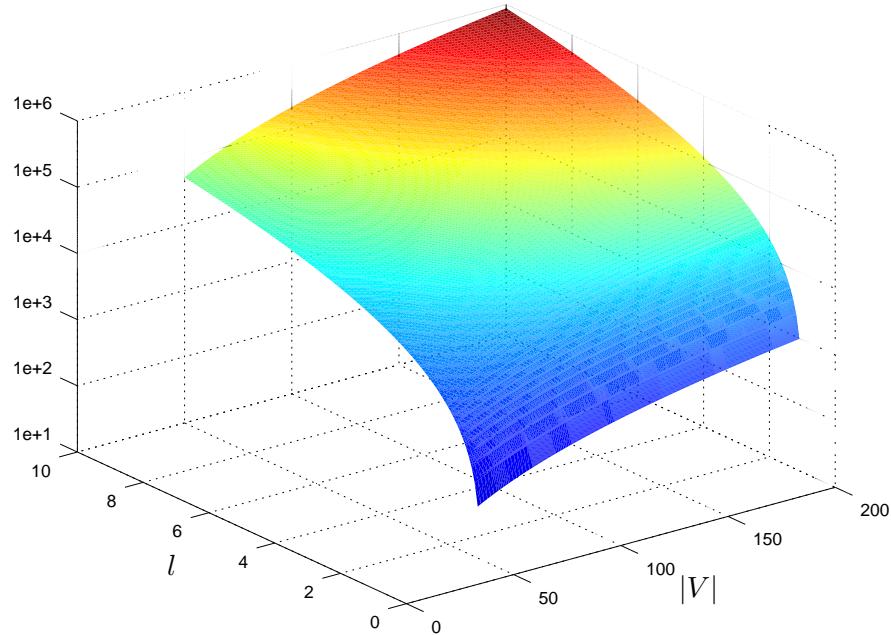
Złożoność takiego rozwiązania to  $O(|V| \cdot (l^3 + |V|))$ , gdzie  $l$  to liczba iteracji algorytmu (miara tego, jak daleko od dopuszczalności dla INCREMENTAL MST jest optymalne rozwiązanie dla problemu MST). W pierwszej iteracji, liczba potencjalnych krawędzi do usunięcia wynosi  $l - 1$  i stopniowo maleje. Dla każdej takiej krawędzi musimy zaś zdeterminować zbiór możliwych do dodania łuków. Bez straty ogólności możemy założyć, że po uzyskaniu podziału drzewa  $T^{i-1}$  na  $(T_1^{i-1}, T_2^{i-1})$ , aby ustalić właściwy zbiór krawędzi, będziemy szukać odpowiednich łuków wychodzących z wierzchołków należących do drzewa o mniejszej ich liczbie (będziemy analizować krawędzie wychodzące z wierzchołków  $v \in T_1^{i-1}$  jeśli  $|v \in T_1^{i-1}| < |v \in T_2^{i-1}|$ , z wierzchołków  $v \in T_2^{i-1}$  w przeciwnym przypadku). Możemy założyć, że w najgorszym z możliwych przypadków wszystkie krawędzie tworzące podział na poddrzewa  $T_1^{i-1}$  oraz  $T_2^{i-1}$  dzielą oryginalne drzewo  $T^{i-1}$  na możliwie równe części — każdorazowe ustalenie zbioru krawędzi  $\mathcal{Q}(T^{i-1}, e^{i-1})$  (a tym samym  $\mathcal{Q}(T^{i-1}, e^{i-1}) \cap T^0$ ) zatem wymagać będzie od nas przeglądnienia wszystkich krawędzi wychodzących z  $\frac{|V|-1}{2} - \frac{i-j}{2}$  wierzchołków, gdzie  $j$  to indeks krawędzi  $e_j$ , według której dokonujemy podziału. Możemy także założyć, że w pesymistycznym przypadku będziemy mieli do czynienia z grafem pełnym, gdzie stopnień każdego wierzchołka  $v \in V$  wynosi  $d(v) = |V| - 1$ . Stąd otrzymujemy wzór na ogólną liczbę porównań wybranych par krawędzi  $(e^{i-1}, e')$ , gdzie  $e^{i-1}$  jest krawędzią wychodzącą z rozwiązania, zaś  $e'$  — wchodzącą:

<sup>8</sup> W rozdziale 5 omówimy algorytm, który także działa iteracyjnie ze względu na wartość tej funkcji, lecz przedstawiona własność nie jest zachowana, co przekłada się na dużo lepsze otrzymywane czasy działania

<sup>9</sup>O zbiorze krawędzi łączącym dwa poddrzewa rozpinające powstałe w wyniku cięcia oryginalnego drzewa pisaliśmy w poprzednim rozdziale (2.3).

$$\begin{aligned} \sum_{i=1}^{l-1} \sum_{j=1}^i (|V|-1) \cdot \left( \frac{|V|-1}{2} - \frac{i-j}{2} \right) &= \frac{1}{12} \cdot (l-3) \cdot (|V|-1) \cdot (l^2 - 3 \cdot l \cdot |V| + 2) = \\ &= \frac{l^3 \cdot (|V|-1)}{12} - \frac{l^2 \cdot (|V|^2 - 1)}{4} + \frac{3 \cdot l \cdot |V|^2}{4} - \frac{7 \cdot l \cdot |V|}{12} - \frac{l}{6} - \frac{|V|}{2} + \frac{1}{2}. \end{aligned}$$

Dodając do tego koszt odnalezienia potencjalnych krawędzi, które będziemy usuwać ( $|V|-1$  — wystarczy raz przejść po wszystkich krawędziach drzewa  $T^0$ , z każdą następną iteracją zbiór ten pomniejsza się o ustaloną krawędź, więc nie musimy za każdym razem rozpoczynać analizy krawędzi drzewa od nowa) otrzymujemy wymieniony na samym początku stopień złożoności, który ilustruje wykres 3.9.



Rysunek 3.9: Stopień złożoności naiwnego algorytmu rozwiązywającego problem minimalnego drzewa rozpinającego w wersji INCREMENTAL głównie zależy od liczby krawędzi, o które optymalne rozwiązanie dla problemu MST różni się od dopuszczalnej liczby nowych krawędzi względem starego rozwiązania w problemie INCREMENTAL MST.

### 3.4.2 Zagadnienia oparte na problemie INCREMENTAL

#### Problem adwersarza

Przyjrzymy się teraz nieco bardziej rozbudowanemu problemowi, znanego jako **problem adwersarza** (ang. adversarial problem). W tym przypadku zamiast skupiać się na minimalizacji rozwiązania, naszym celem (celem adwersarza) jest maksymalizowanie jego wartości. Osiągnąć cel możemy poprzez wybór takiego scenariusza, dla którego najlepsza wartość rozwiązania problemu INCREMENTAL będzie najgorsza:

$$\max_{\mathbf{s}' \in S} \min_{\mathbf{y} \in X_{\mathbf{x}}^k} v(\mathbf{y}, \mathbf{s}') \quad (3.17)$$

Latwo zauważać, że oba te wybory — adwersarza i osoby wybierającej rozwiązanie dla problemu INCREMENTAL — nie zależą od siebie, przez co problem sprowadza się do omawianego wcześniej z tą różnicą, że zamiast dwóch scenariuszy (gdzie na podstawie pierwszego wybieraliśmy rozwiązanie w postaci wektora  $\mathbf{x}$ ), mamy ich większą liczbę i adwersarz dla każdego z nich musi policzyć wartość  $\min_{\mathbf{y} \in X_{\mathbf{x}}^k} v(\mathbf{y}, \mathbf{s}')$ , by móc wybrać najbardziej sprzyjający mu scenariusz (skutkujący zwróceniem największej wartości). Problem można



zatem bez trudu zrównoleglić, przez co nie da się o nim napisać wiele więcej ponad to, co już napisaliśmy przy omawianiu wcześniejszego problemu.

### Odporny problem przyrostowy

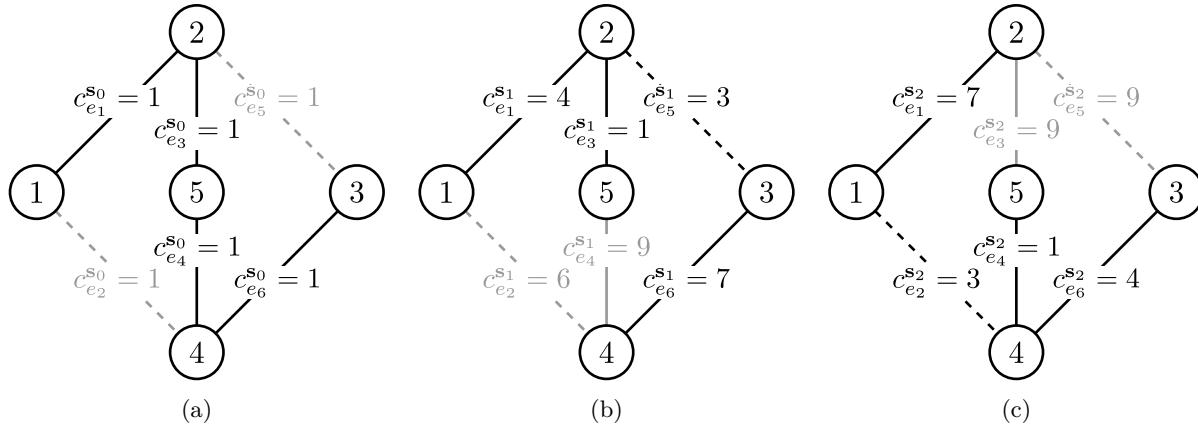
Kolejnym rozszerzeniem poprzednio omawianego problemu jest problem **odpornej optymalizacji przyrostowej** (ang. *robust incremental optimization*). W odróżnieniu od problemów typu INCREMENTAL oraz problemu adwersarza, w tym przypadku chcemy cofnąć się o jeden krok w procesie poszukiwania rozwiązania i zminimalizować jego wartość uwzględniając dodatkowo naszą pierwszą decyzję — tę odnoszącą się do wektora  $\mathbf{x}$ , będącą podstawą do rozwiązywania problemu INCREMENTAL. Jak możemy się domyślać, problem ten definiujemy w następujący sposób:

$$\min_{\mathbf{x} \in X} \left( v(\mathbf{x}, \mathbf{s}) + \max_{\mathbf{s}' \in S} \min_{\mathbf{y} \in X_{\mathbf{x}}^k} v(\mathbf{y}, \mathbf{s}') \right) \quad (3.18)$$

Niech początkowym scenariuszem, na podstawie którego wybierzemy wektor  $\mathbf{x}$ , będzie  $\mathbf{s}_0 = \{1, 1, 1, 1, 1, 1\}$ , zaś scenariuszami, spośród których będzie mógł wybierać adwersarz:

- $\mathbf{s}_1 = \{4, 6, 1, 9, 3, 7\}$  oraz
- $\mathbf{s}_2 = \{7, 3, 9, 1, 9, 4\}$ .

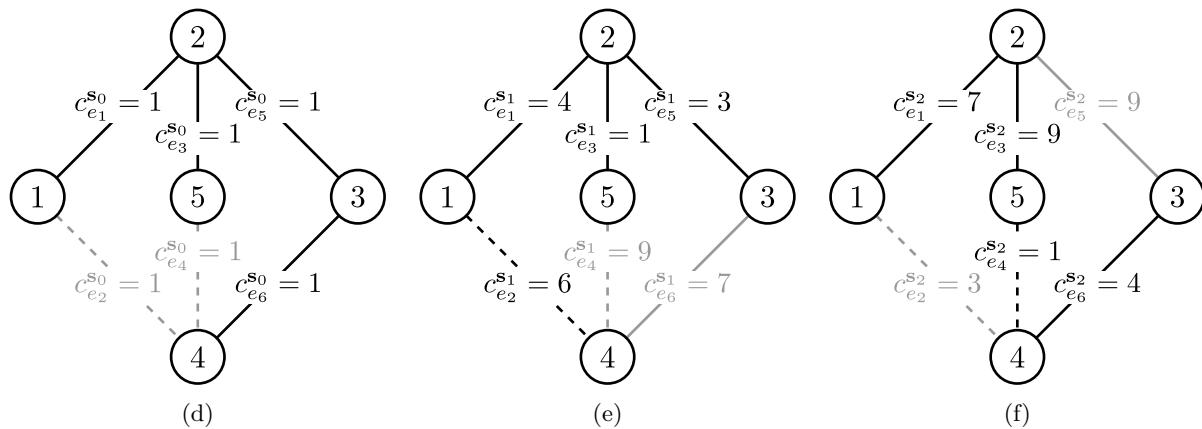
Oczywiście wybór scenariusza  $\mathbf{s}_0$  nie jest przypadkowy — naszym celem jest pokazanie możliwej do osiągnięcia różnicy w jakości rozwiązań dla dwóch różnych sposobów podejścia do zadanego problemu: klasycznego, polegającego na prostym wybraniu rozwiązania o najmniejszym koszcie w pierwszym kroku z pominięciem problemu adwersarza, tożsamym z rozwiązywaniem zagadnienia minimalnego drzewa rozpinającego dla pojedynczego scenariusza ( $\min_{\mathbf{x} \in X} v(\mathbf{x}, \mathbf{s})$ ), oraz odpornego. Łatwo zauważać, że w przypadku (tak jak to pokazano na rysunkach od 3.10a do 3.10f oraz 3.11). Dodatkowo, niech parametr dla problemu INCREMENTAL MINIMUM SPANNING TREE wynosi  $k = 1$ .



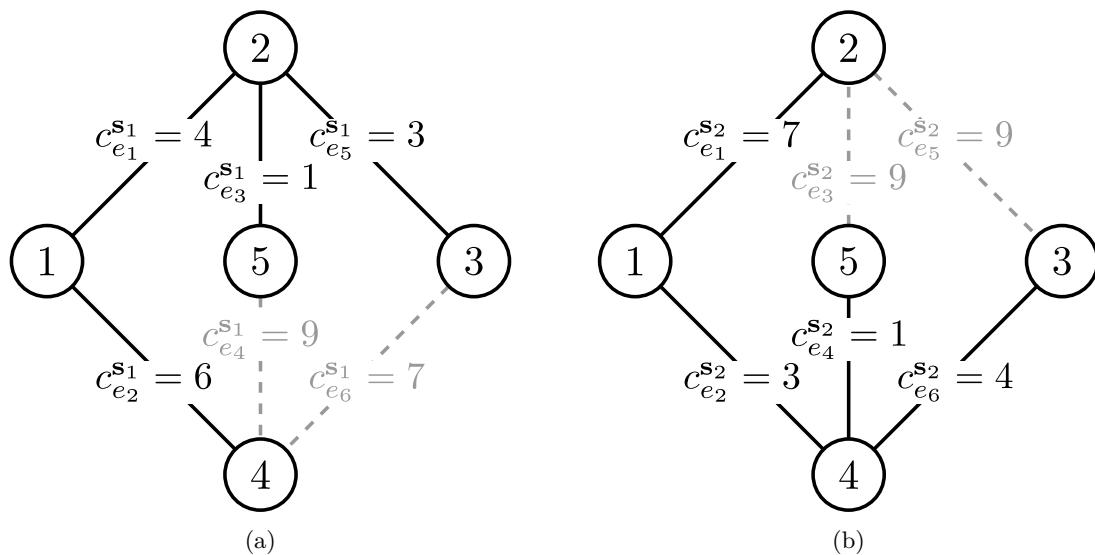
Rysunek 3.10: Pierwsze rozwiązanie. (a) Początkowe rozwiązanie (5 w tabeli: 0 1 0 1 1 1 15 15 15) (b) Incremental dla 1 scenariusza (szara ciągła - usunięto z początkowego rozwiązania, czarna przerywana - dodano,  $k = 1$ ) (c) Inc dla 2 scenariusza

Wszystkie rozwiązania:  $x : \text{adwersarz dla } 1 \text{ scenariusza, dla } 2, \max (\text{no i dodać trzeba stały koszt początkowego rozwiązania: } 4)$ .  $e_{23}e_{34}e_{41}e_{12}e_{25}e_{54}$

1 0 1 1 0 1 14 15 15 1 1 0 1 0 1 15 15 15 1 1 1 0 0 1 17 15 17 1 1 1 0 1 0 14 17 17 0 1 0 1 1 1 15 15 0 1  
1 0 1 1 17 15 17 0 1 1 1 0 14 15 15 1 1 0 1 0 14 21 21 1 0 0 1 1 1 14 20 20 1 0 1 0 1 1 14 17 17 1 0 1 1 1 0  
14 20 20 0 1 1 1 0 1 18 15 18



Rysunek 3.10: Inne rozwiązanie (8 w tabeli: 1 1 0 1 1 0 14 21 21) (d) O (e) O (f) O



Rysunek 3.11: Optymalne rozwiązanie dla scenariuszy adwersarza bez limitu  $k = 1$ . (a) O (b) O



**Wariant dwukrokowy**

**Optymalizowanie z możliwością poprawy**

TODO - liniowy model

---

### 3.5 Podsumowanie rozdziału

# Problemy programowania liniowego

---

Chcąc podjąć próbę formalnego opisania przedstawionych wcześniej problemów należy najpierw zapoznać się z ogólnymi zasadami ich zapisywania. Jak mogliśmy zaobserwować w poprzednim rozdziale, opisując problem typu INCREMENTAL (3.4.2), uciekliśmy się do przedstawienia go za pomocą serii nierówności oraz równości — innymi słowy zapisaliśmy go w postaci **modelu programowania liniowego**. Możliwość przekształcenia dowolnie zadanej problemu do takiej postaci może nam zagwarantować odnalezienie jego istniejącego rozwiązania w czasie wielomianowym i takie narzędzie może stanowić doskonały punkt wyjściowy do dalszych rozważań. My także z niego skorzystamy do konstrukcji optymalnego algorytmu rozwiązywającego problem odpornej optymalizacji dla wyszukiwania minimalnego drzewa rozpinającego w rozdziale następnym, a opierając się na jego własnościach wykażemy poprawność takiego podejścia.

---

## 4.1 Programowanie liniowe

Mając na uwadze, że nie chcemy zajmować się zagadnieniem programowania liniowego samym w sobie, gdyż nie jest to naszym celem, w poniższym rozdziale zostaną przytoczone tylko podstawowe fakty, niezbędne do zrozumienia dalszej jego części. Punktem wyjścia, do pozyskania głębszej wiedzy na poruszany w tym rozdziale temat, może być książka [15], która omawia wszystkie aspekty tej dziedziny, począwszy od programowania wkleślego, przez liniowe, do całkowitoliczbowego, tłumacząc zasady działania każdego z tych modeli.

Przez pojęcie **programowania liniowego** będziemy rozumieli zbiór liniowych równań oraz nierówności wraz z określona dla nich **funkcją celu**, która także musi przybrać formę liniowego wyrażenia arytmetycznego. Zbiór tych ograniczeń wraz z funkcją celu będziemy nazywali **modelem prymalnym**<sup>1</sup> programowania liniowego. Schemat takiego modelu wygląda następująco:

$$\min f(\mathbf{x}, \dots), \quad (4.1a)$$

$$\text{s.t. } g_1(\mathbf{x}, \dots) \diamond b_1, \quad (4.1b)$$

$$g_2(\mathbf{x}, \dots) \diamond b_2, \quad (4.1c)$$

$$\dots \quad (4.1d)$$

$$g_n(\mathbf{x}, \dots) \diamond b_m, \quad (4.1e)$$

$$x_i \geq x_i^{\text{LB}}, \quad x_i \in \mathbf{x}, \quad (4.1f)$$

$$x_i \leq x_i^{\text{UB}}, \quad x_i \in \mathbf{x}, \quad (4.1g)$$

gdzie znak  $\diamond$  zastępuje dowolny znak ze zbioru  $\{=, \leq, \geq\}$ , funkcja  $f(\mathbf{x}, \dots)$  jest funkcją celu i zwykle przyjmuje postać podobną do  $\sum_{i=1}^{i=n} c_i \cdot x_i$ ,  $\mathbf{x}$  jest wektorem symbolizującym rozwiązanie,  $\mathbf{c}$  zbiorem kosztów, zaś  $|\mathbf{x}| = |\mathbf{c}| = n$ . Wektor  $\mathbf{x}$  będziemy nazywać zbiorem **zmiennych decyzyjnych** modelu. Dla wszystkich do tej pory opisanych problemów w poprzednim rozdziale wykorzystywaliśmy podobne oznaczenia — zmienna decyzyjna  $x_i$  przyjmowała odpowiednio wartość 0, gdy krawędź  $e_i$  nie należała do poszukiwanego rozwiązania

<sup>1</sup>Terminem „prymalny” będziemy określać modele programowania liniowego/całkowitoliczbowego, których funkcja celu jest minimalizacyjna. Problemem **dualnym** nazwiemy model, który skupia się na maksymalizowaniu wartości. Zagadnienia związane z obydwooma modelami zostały dokładniej opisane w książce [15, 67–73].



oraz 1 w przeciwnym przypadku. Tutaj generalizujemy tę ideę, dopuszczając przyjmowanie przez te zmienne wartości z zakresu od  $x_i^{\text{LB}}$  do  $x_i^{\text{UB}}$ , gdzie odpowiednio  $x_i^{\text{LB}}$  oznacza dolne ograniczenie na wartość zmiennej  $x_i$  (ang. *lower bound*), zaś  $x_i^{\text{UB}}$  — górne (ang. *upper bound*). Od rodzaju wartości jakie mogą przyjmować te zmienne zależy z jakiego typu problemem mamy do czynienia — w przypadku, gdy wartości  $x_i \in (x_i^{\text{LB}}, x_i^{\text{UB}})$ , model w którym występują tylko takie zmienne będziemy nazywali modelem programowania liniowego. W przeciwnym przypadku, gdy w modelu pojawią się zmienne o wartościach dyskretnych ( $x_i \in [x_i^{\text{LB}}, x_i^{\text{UB}}]$ ), mówimy o modelu **całkowitoliczbowym**. Ma to o tyle duże znaczenie, że w pierwszym przypadku jesteśmy w stanie zagwarantować, że o ile dla zdefiniowanego modelu istnieje **rozwiążanie optymalne** (minimalizujące funkcję celu  $f(\mathbf{x}, \dots)$ ), otrzymamy je w czasie wielomianowym od wielkości modelu. Informację o braku takiego rozwiązania również uzyskamy w takim czasie (często dużo szybciej, gdy algorytm służący do rozwiązywania tak zdefiniowanych problemów znajdzie się w stanie, który będzie tożsamym z brakiem istnienia jakiegokolwiek rozwiązania<sup>2</sup>). W przypadku zaś modeli całkowitoliczbowych nie mamy takiej gwarancji<sup>3</sup> i często czas potrzebny do znalezienia rozwiązania wykracza poza wielomianowy. W przypadku gdy w modelu występują oba typy zmiennych, mamy do czynienia z modelem mieszanym **MIP** (ang. *Mixed Integer Programming*), którego czas rozwiązania również jest ponad-wielomianowy. Aby zilustrować schemat działania tak opisanego modelu, posłużymy się prostym przykładem:

$$\begin{aligned} \min \quad & 2 \cdot x_1 + 3 \cdot x_2, \\ \text{s.t.} \quad & 3 \cdot x_1 + 2 \cdot x_2 \leq 12, \end{aligned}$$

$$1 \cdot x_1 - 4 \cdot x_2 \geq 2,$$

$$0 \leq x_1 \leq 3$$

$$0 \leq x_1 \leq 3, \quad x_2 \in \mathbf{R},$$

$$0 \leq x_2 \leq 4, \quad x_i \in \mathbf{x},$$

$$f(\mathbf{x}, \mathbf{c}) = \sum_{i=1}^{i=2} c_i \cdot x_i,$$

$$g_1(\mathbf{x}, \mathbf{a}_1) = \sum_{i=1}^{i=2} a_{1,i} \cdot x_i,$$

$$g_2(\mathbf{x}, \mathbf{a}_2) = \sum_{i=1}^{i=2} a_{2,i} \cdot x_i,$$

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{c}), \\ \text{s.t.} \quad & g_1(\mathbf{x}, \mathbf{a}_1) \leqslant \end{aligned}$$

$$g_2(\mathbf{x}, \mathbf{a}_2) \geq b_2,$$

$$0 \leq x_1 \leq 3$$

$$0 \leq x_1 \leq s, \quad x_2 \in \mathbf{X},$$

$$0 \leq x_2 \leq 4, \quad x_i \in \mathbf{x},$$

(a)

(b)

(c)

Rysunek 4.1: Przykładowy problem sformułowany jako model programowania liniowego. **(a)** Jawnie zapisany model — w takiej formie będzie rozwiązywany przez wyspecjalizowany algorytm do rozwiązywania zagadnień programowania liniowego, **(b)** Reprezentacja wszystkich elementów modelu w zwartej formie funkcji z następująco zdefiniowanymi zbiorami wartości:  $\mathbf{x} = \{x_1, x_2\}$ ,  $x_1 \in (0, 3)$ ,  $x_2 \in (0, 4)$ ,  $\mathbf{c} = [2, 3]$ ,  $\mathbf{b} = [12, 2]$ ,  $\mathbf{a}_1 = [3, 2]$  i  $\mathbf{a}_2 = [1, -4]$ . **(c)** Zwarty zapis modelu.

Oczywistym rozwiązaniem problemu 4.1 jest para wartości  $(2, 0)$ . Przypisanie takich liczb do zmiennych  $x_1$  oraz  $x_2$  spełnia wszystkie ograniczenia zawarte w modelu, wartości przypisane zmiennym należą do ich dziedzin, zaś wartość funkcji celu dla tak zadanych zmiennych osiąga swoje minimum. Zwykle możemy spotkać się z macierzowym zapisem modelu, gdzie przedstawione przez nas wektory  $\mathbf{a}_1$  oraz  $\mathbf{a}_2$  traktuje się jako macierz  $A = [\mathbf{a}_1, \mathbf{a}_2]$  a układ ograniczeń zadanego modelu prezentuje się w postaci  $A \cdot \mathbf{x} = \mathbf{b}$ .

#### 4.1.1 Problem minimalnego drzewa rozpinającego

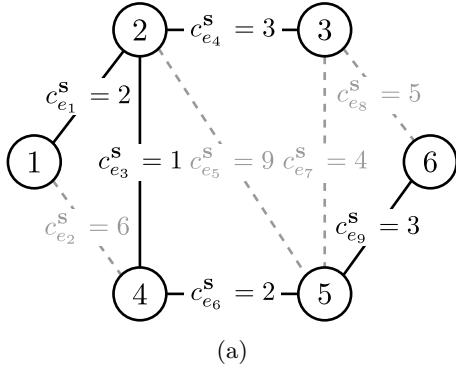
W dalszej części rozdziału poznamy szereg różnych modeli przedstawiających problem minimalnego drzewa rozpinającego, poczynając od bezpośredniej próby przełożenia jego definicji na model matematyczny, poprzez modele całkowitaliczbowe aż do modelu programowania liniowego, którego specyficzna budowa zapewni nam całkowitaliczbowe rozwiązanie. Na tej podstawie będziemy chcieli skonstruować modele odpowiadające problemowi minimalnego drzewa rozpinającego w wersji INCREMENTAL, które posłużą nam jako punkt wyjścia do dalszych rozważań.

<sup>2</sup>Dla przykładu, ograniczenia w postaci funkcji  $g(\mathbf{x}, \dots)$  mogą tak ograniczyć zbiór wartości, które może przyjąć dana zmienna decyzyjna, do zbioru pustego.

<sup>3</sup>Algorytm rozwiązuający dany problem w pierwszej kolejności znajdzie rozwiązanie problemu ignorując ograniczenia na całkowitoliczbowość zmiennych decyzyjnych, a następnie zacznie szukać takiego rozwiązania, które będzie je spełniać, będąc jednocześnie możliwie najlepszym. W wielu przypadkach kończy się to na inteligentnym przeglądaniu wszystkich możliwych rozwiązań metodą np. podziału i ograniczeń (ang. *Branch and bound*) [15, 433–448].

### 4.1.2 Naiwne podejście

Pierwszym i zarazem najbardziej naturalnym sposobem podejścia do zamodelowania interesującego nas problemu jest model przedstawiony poniżej. Opiera się on na ogólnie znanych właściwościach drzew rozpinających i ich podstruktury — jeśli dane drzewo  $T$  jest drzewem rozpinającym dla grafu  $G = (V, E)$  (między innymi składa się z  $|V| - 1$  krawędzi), to każdy podzbiór wierzchołków  $S \subseteq V$  grafu jest połączony ze sobą dokładnie  $|S| - 1$  krawędziami (zobacz rysunek 4.2a). Niech zbiór  $E(S)$  będzie zdefiniowany jako  $E(S) = \left\{ \{i, j\} : v_i \xrightarrow{1} v_j \in E \wedge v_i \in S \wedge v_j \in S \right\}$  i oznacza zbiór wszystkich krawędzi łączących dowolne dwa różne wierzchołki ze zbioru  $S$ . Niech dodatkowo dla każdej krawędzi w grafie  $e \in E$  będzie zdefiniowana zmienna decyzyjna  $x_e$ , której wartość odpowiada decyzji o przynależności danej krawędzi do minimalnego drzewa rozpinającego, będącego rozwiązaniem  $T$  dla modelu 4.2b (innymi słowy, minimalne drzewo rozpinające  $T$  składa się z krawędzi  $e$ , dla których  $x_e = 1$  —  $T = \{e : x_e = 1\}$ ). W takiej sytuacji, równanie 4.4c w naturalny sposób opisuje wskazaną przez nas własność. W szczególności dla zbioru  $S = V$  mamy  $E(S) = E$  a wspomniane równanie sprowadza się do wyrażenia 4.4b, które w dobitny sposób wyraża podstawową własność posiadaną przez drzewa rozpinające dowolny graf. Teraz, aby uzyskać minimalne takie drzewo, wystarczy byśmy zdefiniowali funkcję celu, która będzie dążyć do zminimalizowania kosztów znalezionego rozwiązania (4.4b). Niestety model, którego idea wydaje się być bardzo prosta, jest równie intuicyjny co niepraktyczny w zastosowaniu — wystarczy spojrzeć na definicję zbioru  $E(S)$  bądź na rysunek 4.2a oraz jego opis, by dojść do wniosku, że liczba wymaganych przez model ograniczeń jest ponad wielomianowa — znacznie większa niż czas, w którym spodziewalibyśmy się otrzymać rozwiązanie takiego modelu programowania liniowego (przypomnijmy, że modelem LP nazywamy model, którego zmienne decyzyjne nie są ograniczone do dyskretnego zbioru liczb)<sup>4</sup>



(a)

$$\min \sum_{e \in E} c_e \cdot x_e, \quad (4.4a)$$

$$\text{s.t. } \sum_{e \in E} x_e = |V| - 1, \quad (4.4b)$$

$$\sum_{e \in E(S)} x_e = |S| - 1, \quad S \subseteq V, \quad (4.4c)$$

$$x_e \geq 0, \quad e \in E, \quad (4.4d)$$

(b)

Rysunek 4.2: (a) Przykładowe drzewo rozpinające  $T = \{e_1, e_3, e_4, e_6, e_9\}$  dla grafu  $G = (V, E)$ . Dla dowolnego podzbioru  $S \subseteq V$  liczba krawędzi  $\{e_{ij} : e_{ij} \in T \wedge v_i \in S \wedge v_j \in S\}$  jest równa  $|S| - 1$ . (b) Model liniowego programowania dla problemu minimalnego drzewa rozpinającego.

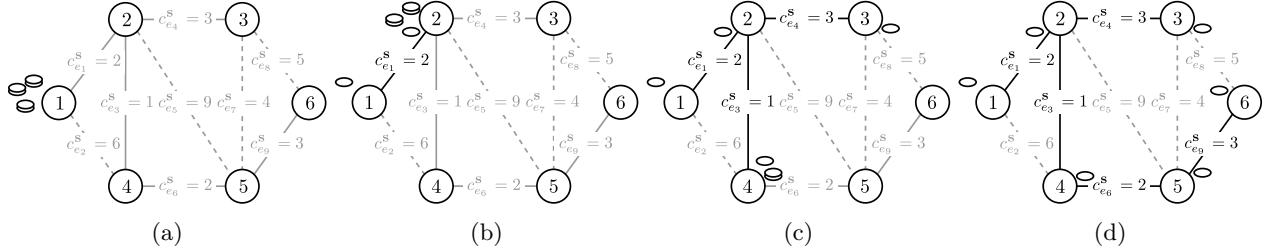
### 4.1.3 Przepływy

Odmiennym podejściem do problemu minimalnego drzewa rozpinającego jest jego zdefiniowanie za pomocą **przepływów** [12, 38–44]. W tej interpretacji węzłami są miejsca, w których dozwolony jest skład towarów, zaś krawędzie między nimi symbolizują ścieżki, którymi dane towary możemy przemieszczać pomiędzy dwoma sąsiednimi punktami, ponosząc tego koszty proporcjonalnie do wag łączących ich krawędzi. Założymy, że mamy dany graf  $G$ , taki jak przedstawiono na rysunku 4.3. Dodatkowo niech jeden z węzłów w grafie ( $v_1$ ) będzie wyróżniony i posiada tyle jednostek towarów ile w grafie jest wierzchołków. Naszym celem jest dystrybucja

<sup>4</sup>Z wiedzy ogólnej wiemy, że wszystkich podzbiorów zbioru składającego się z  $n$  elementów jest  $2^n$  ( $\sum_{i=0}^n \binom{n}{i}$ ). Wzór ten doskonale opisuje naszą sytuację — chcemy mieć ograniczenie dla wszystkich możliwych zbiorów  $S$  jedno-, dwu-, ...,  $n$ -elementowych. Z drugiej zaś strony powiedzieliśmy (rozdzielając pojęcia programowania liniowego od całkowitoliczbowego), że istnienie modelu LP gwarantuje nam możliwość otrzymania jego rozwiązania w czasie wielomianowym **od rozmiaru problemu**. W związku z czym algorytm rozwiązujący dany model będzie mimo wszystko zwracał jego rozwiązanie w nieakceptowalnym dla nas czasie.



wszystkich towarów z wykorzystaniem dostępnych krawędzi w taki sposób, aby każdy z transportów dotarł w efekcie końcowym do innego wierzchołka — biorąc pod uwagę, że życzymy sobie aby przy okazji ponieść tego jak najmniejsze koszty, okaże się że droga, która zostanie w ten sposób wytyczona z punktu startowego  $v_1$  do wszystkich  $v_j \in V \setminus \{v_1\}$ , jest minimalnym drzewem rozpinającym ten graf a fakt rozdzielenia towarów pomiędzy wszystkie węzły w oczywisty sposób świadczy o spójności tak wygenerowanego drzewa.



Rysunek 4.3: Rysunek ilustrujący wygenerowany przepływ dla problemu znalezienia minimalnego drzewa rozpinającego, gdzie szarymi ciągłymi liniami zostało zaznaczone optymalne rozwiązanie problemu, przerywanymi — pozostałe krawędzie w grafie, zaś czarnymi liniami oznaczona są łuki, które do tej pory były wykorzystane do transportu towarów. (a) Sytuacja początkowa z wszystkimi pięcioma elementami zgromadzonymi przy węźle  $v_1$ . (b) Z wierzchołka  $v_1$  została przekazana piątka elementów z wykorzystaniem krawędzi  $e_1$ . (c) Z wierzchołkiem  $v_2$  połączone są jeszcze dwie krawędzie należące do minimalnego drzewa rozpinającego danego grafu przy tak zdefiniowanych kosztach — wykorzystując krawędzie  $e_4$  oraz  $e_5$ , do nowych wierzchołków zostały w sumie przemieszczone cztery elementy, pozostawiając jeden w wierzchołku  $v_2$ . (d) W wyniku rozdysystrybuowania wszystkich elementów tak, aby każdy wierzchołek posiadał dokładnie jeden spośród nich, otrzymano minimalne drzewo rozpinające dla prezentowanego grafu.

Niech teraz z każdą krawędzią  $e_{ij}$  ( $e_{ij} \equiv v_i \xrightarrow{1} v_j$ ) będą związane dwie dodatkowe zmienne:  $f_{ij}$  oraz  $f_{ji}$  symbolizujące przepływ (ang. *flow*) na tej krawędzi w danym kierunku (wartość zmiennej  $f_{ij}$  oznacza więc liczbę elementów, które z wierzchołka  $v_i$  przesunieliśmy do  $v_j$  przy wykorzystaniu krawędzi  $e_{ij}$ , zaś zmieniąca odwrotną zależność — liczbę transportowanych elementów po tym samym łuku, lecz z  $v_j$  do  $v_i$ ). Tak jak to sobie powiedzieliśmy i jak mogliśmy zaobserwować na rysunku 4.3a, naszym celem jest wyjście z sytuacji, gdzie w jednym wierzchołku znajduje się  $n = |V|$  elementów, i zakończenie w takiej, w której do każdego wierzchołka przyporządkowany jest jeden z nich — innymi słowy chcemy aby z wyróżnionego wierzchołka wysłanych zostało  $n - 1$  posiadanego przez niego elementów przy jednoczesnym upewnieniu się, że żaden z nich do niego nie wróci. W równaniu 4.5b możemy wyróżnić dwie sumy, które kolejno oznaczają:

- $\sum_{(i,j) \in E} f_{ij}$  — sumę wartości wszystkich przepływów, których kierunek jest wyznaczony w stronę wierzchołków innych niż  $v_i$ ,
- $\sum_{(j,i) \in E} f_{ji}$  — sumę wartości przepływów skierowanych w stronę wierzchołka  $v_i$ .

Innymi słowy, wartość pierwszego wyrażenia będzie równa liczbie ładunków, które wysłaliśmy z wierzchołka  $v_i$  do pozostałych węzłów, zaś wartość drugiej sumy oznacza liczbę tych ładunków, które do wierzchołka  $v_i$  dotarły. Widzimy zatem, że wartość wyrażenia z równania 4.5b dla parametru  $i = 1$  odpowiada opisanej przez nas sytuacji — wierzchołek początkowy wysyła w kierunku pozostałych wierzchołków łącznie  $n - 1$  ładunków (sobie pozostawiając jeden —  $\sum_{(1,j) \in E} f_{1j} = n - 1$ ), zaś sam nie przyjmuje żadnych —  $\sum_{(j,1) \in E} f_{j1} = 0$ . Dla wszystkich pozostałych przypadków, tak jak to mogliśmy zaobserwować na rysunkach od 4.3a–4.3d, każdy wierzchołek z otrzymanych od sąsiada elementów wybierał dla siebie jeden a resztę przekazywał dalej, co natychmiast prowadzi do równości  $\sum_{(j,i) \in E} f_{ji} - \sum_{(i,j) \in E} f_{ij} = 1$  (i oczywiście  $\sum_{(i,j) \in E} f_{ij} - \sum_{(j,i) \in E} f_{ji} = -1$ , gdyż w takiej formie najwygodniej jest nam zastosować tą równość w prezentowanym modelu).

$$\min \sum_{e \in E} c_e \cdot x_e, \quad (4.5a)$$

$$\text{s.t. } \sum_{(i,j) \in E} f_{ij} - \sum_{(j,i) \in E} f_{ji} = \begin{cases} n-1 & \text{jeżeli } i=1, \\ -1 & \text{w przeciwnym przypadku,} \end{cases} \quad (4.5b)$$

$$f_{ij} \leq (n-1) \cdot x_{ij}, \quad \forall \{i,j\} \in E, \quad (4.5c)$$

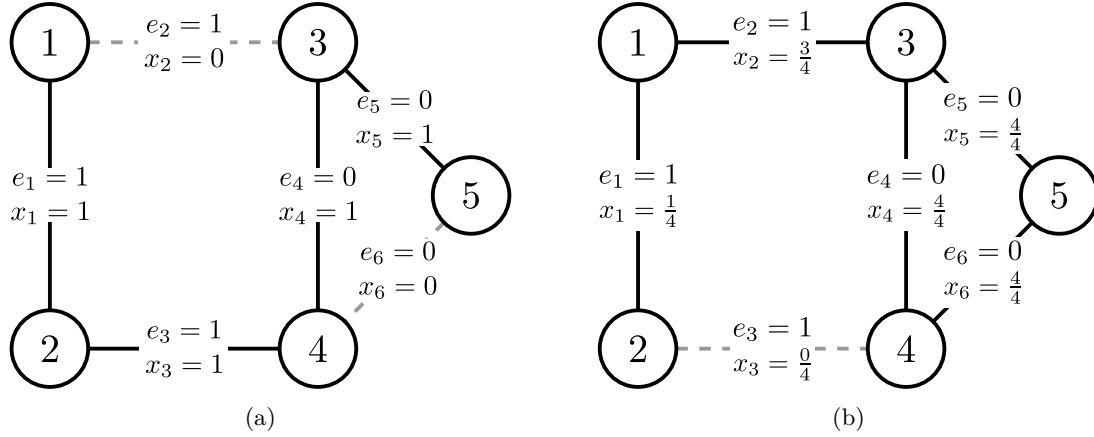
$$f_{ji} \leq (n-1) \cdot x_{ij}, \quad \forall \{i,j\} \in E, \quad (4.5d)$$

$$\sum_{e \in E} x_e = n-1, \quad (4.5e)$$

$$f_e \geq 0, \quad \forall e \in E, \quad (4.5f)$$

$$x_e \in \{0, 1\}, \quad \forall e \in E. \quad (4.5g)$$

Reszta ograniczeń w 4.5 wydaje się być intuicyjna — poprzez ograniczenia takie jak 4.5c czy 4.5c gwarantujemy sobie, że w przypadku krawędzi  $e$  należącej do minimalnego drzewa rozpinającego ( $x_e = 1$ ), liczba przesyłanych przezeń elementów wynosić może maksymalnie  $n-1$  (czyli tyle ile przesyłałby wierzchołek początkowy gdyby posiadał tylko jednego sąsiada), w przeciwnym zaś razie wykorzystanie tej krawędzi jest zabronione ( $x_{ij} = 0 \rightarrow f_{ij} = 0$ ). Dodatkowo posiadamy ograniczenie, które determinuje liczbę krawędzi tworzących rozwiązanie oraz wreszcie ograniczenia wartości samych zmiennych decyzyjnych. Te ostatnie widzimy, że ograniczają ich dziedzinę do jedynie dwóch wartości — 0 oraz 1, co wskazuje na to, że przedstawiony w 4.5 model opisuje problem programowania całkowitoliczbowego, w związku z czym oczekiwany czas otrzymania rozwiązania tak postawionego problemu przy wykorzystaniu takiego modelu nie jest w żaden sposób ograniczony funkcją wielomianową od wielkości problemu. Możemy oczywiście zignorować ograniczenia całkowitoliczbowości, licząc na prawidłowe zachowanie się modelu bez tego ograniczenia — choć brzmi to niedorzecznie, w rzeczywistości (np. dla grafu i kosztów zaprezentowanych na rysunkach 4.3a–4.3d) może się okazać, że otrzymamy identyczne rozwiązanie co w przypadku zastosowania ograniczeń na całkowitoliczbowość, co może sugerować, że tak zbudowany model również jest poprawny.



Rysunek 4.4: Optymalne rozwiązania dla modelu przedstawionego w 4.5. (a) Minimalne drzewo rozpinające o całkowitym koszcie 2 zwrócone przez model opisany ograniczeniami 4.5a–4.5g. Wartości zmiennych decyzyjnych  $\{x_i : e_i \in E\}$  są ograniczone do wartości 0 (krawędź nie należy do rozwiązania) albo 1. (b) Rozwiązanie nie będące drzewem rozpinającym, zwrócone przez model opisany ograniczeniami 4.5a–4.5f ( $\forall e_i \in E : x_i \geq 0$ ).

Niestety, jak widzimy na rysunkach 4.4a oraz 4.4b [12, 41], nawet w najprostszych przypadkach zniesienie wspomnianego ograniczenia może spowodować, że zwracane rozwiązania będą niedopuszczalne w myśl problemu minimalnego drzewa rozpinającego (choć oczywiście wszystkie ograniczenia tak zbudowanego modelu



są spełnione — po prostu nie modeluje on już więcej problemu, którego rozwiązania byśmy oczekiwali [12, 39]). W związku z tym, będziemy chcieli zaproponować trzeci wariant rozwiązania, który w znacznym stopniu opiera się na idei powyższego modelu, jednak w odróżnieniu od swojego poprzednika, optymalne rozwiązania przez niego zwracane będą poprawne nawet jeżeli zaniechamy ograniczenia na całkowitoliczbowość zmiennych decyzyjnych  $\{x_i : e_i \in E\}$ .

#### 4.1.4 Przepływy z wymuszoną unimodularnością

##### Unimodularność

**Modelem unimodularnym** będziemy nazywać taki model, którego wszystkie współczynniki funkcji generujących ograniczenia  $(g_1(\mathbf{x}, \dots), \dots, g_n(\mathbf{x}, \dots))$  — patrz 4.1b–4.1e) należą do zbioru  $\{-1, 0, 1\}$ . Niech współczynnikami każdej funkcji  $g_i(\mathbf{x}, \dots)$  będzie wektor  $\mathbf{p}_i$  (tak, że  $g_i : \mathbf{x} \times \mathbf{p}_i \rightarrow \mathbb{Z}$ ). Macierz  $M = [\mathbf{p}_1, \dots, \mathbf{p}_n]^T$  nazywamy macierzą **całkowicie unimodularną** jeżeli jej współczynniki są liczbami całkowitymi zaś wyznacznik każdej z jej kwadratowych nieosobliwych podmacierzy<sup>5</sup> równa się 1. Model o takich właściwościach będziemy chcieli przedstawić poniżej [12, 42–46]<sup>6</sup>. Zapewnienie sobie takich własności pozwoli nam na niestosowanie ograniczeń z poprzedniego modelu (odnoszących się do całkowitoliczbowości zmiennych decyzyjnych) bez wpływu na zwracane rozwiązania. Mówiąc jeszcze prostszym językiem — przekształcając poprzednio otrzymany model MIP do modelu LP (w którym to przypadku mamy pewność, że czas otrzymania optymalnego rozwiązania nie przekroczy czasu ograniczonego z góry przez pewną funkcję wielomianową) z takimi własnościami wiemy, że zwracane przez oba model rozwiązań będą prawidłowe (zmienne  $x_i$  dla każdej krawędzi  $e_i \in E$  będą przyjmować wartości  $\{0, 1\}$  pomimo, że w drugim modelu ograniczymy je tylko przez  $x_i \geq 0 : \forall e_i \in E$ ).

##### Wielotowarowy model przepływu dla grafów skierowanych

Zasadniczą różnicą pomiędzy tym (ang. *directed multicommodity flow model*) a poprzednio przedstawionym modelem jest odmienny sposób organizacji przepływu pomiędzy wierzchołkami grafu. Spójrzmy na rysunki 4.5a–4.5d — przedstawiają one niemal tą samą sytuację co w 4.3, lecz skupiają się one jedynie na pojedynczym elemencie do przetransportowania, w odróżnieniu od przytoczonego przykładu.

$$\min \sum_{(i,j) \in E} c_{ij} \cdot (y_{ij} + y_{ji}), \quad (4.6a)$$

$$\text{s.t. } \sum_{(j,s) \in E} f_{js}^k - \sum_{(s,j) \in E} f_{sj}^k = -1, \quad \forall k \in V \setminus \{v_s\}, \quad (4.6b)$$

$$\sum_{(j,i) \in E} f_{ji}^k - \sum_{(i,j) \in E} f_{ij}^k = 0, \quad \forall i, k \in V \setminus \{v_s\} \wedge i \neq k, \quad (4.6c)$$

$$\sum_{(j,k) \in E} f_{jk}^k - \sum_{(k,j) \in E} f_{kj}^k = 1, \quad \forall k \in V \setminus \{v_s\}, \quad (4.6d)$$

$$f_{ij}^k \leq y_{ij}, \quad \forall (i,j) \in E \wedge \forall k \in V \setminus \{v_s\}, \quad (4.6e)$$

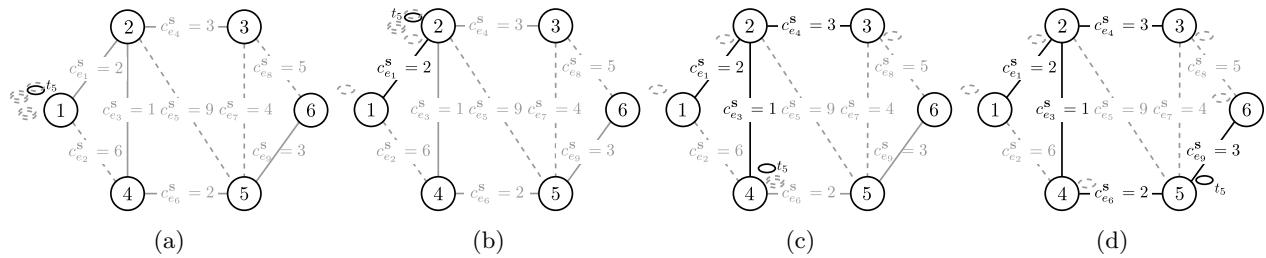
$$\sum_{(i,j) \in E} y_{ij} = n - 1, \quad (4.6f)$$

$$f_{ij} \geq 0, \quad \forall (i,j) \in E, \quad (4.6g)$$

$$y_{ij} \geq 0, \quad \forall (i,j) \in E. \quad (4.6h)$$

<sup>5</sup>Macierzy o tej samej liczbie wierszy oraz kolumn, która powstała w wyniku usunięcia dowolnej liczby wierszy lub kolumn z oryginalnej macierzy. Macierz nieosobliwą nazywamy macierz, której wyznacznik jest różny od zera. Bardzo dobrym przykładem takiej macierzy jest **macierz incydencji** [16, 4]

<sup>6</sup>Nie będziemy tu przytaczać dowodu całkowitej unimodularności macierzy kosztów przedstawianego modelu, podobnie jak zainteresowanych podstawowym zakresem wiedzy na temat macierzy odsyłamy do literatury tej tematyce poświęconej — dla nas kluczowym jest tylko podkreślenie, że model o takich własnościach, rozwiązania przez niego zwracane, są całkowitoliczbowe, nawet dla modeli programowania liniowego (LP)



Rysunek 4.5: Przepływ towarów z perspektywy wyróżnionego elementu  $t_5$ . Wierzchołkiem początkowym jest wierzchołek  $v_1 = v_s$ , końcowym dla towaru —  $v_5$ . **(a)** Towar  $t_5$  został przesunięty w kierunku następnego wierzchołka. Suma przepływów wychodzących dla danego towaru z wierzchołka startowego wynosi 1. Równolegle przesyłane są pozostałe towary, jednak zmienne decyzyjne ich dotyczące nie są na rysunku rozpatrywane. **(b)** Wszystkie wierzchołki pośrednie (nie będące ani początkowym, ani końcowym dla branego pod uwagę towaru) otrzymują towar  $t_5$  **(c)** i natychmiast przesyłają go dalej — bilans przepływu dla takiego wierzchołka względem towaru  $t_5$  wynosi 0. **(d)** Suma przepływów wychodzących oraz wchodzących do wierzchołka będącego odbiorcą swojego towaru wynosi 1 — jeśli wierzchołek nie jest wierzchołkiem początkowym, akceptuje on tylko przeznaczony dla siebie element, resztę zaś przesyła dalej w chwili ich otrzymania.

Na wspomnianych rysunkach widzimy zatem wyraźnie sposób konstrukcji ograniczeń 4.6b, 4.6b oraz 4.6b. Niech zmienne decyzyjne  $f_{ij}$  oznaczają przepływ tak jak w poprzednim modelu. Wprowadźmy teraz nowe zmienne  $f_{ij}^k$ , które wykorzystamy do konstrukcji modelu 4.6. Niech wyrażenie  $f_{ij}^k$  oznacza przepływ skierowany pomiędzy wierzchołkami  $v_i$  oraz  $v_j$  dla towaru  $t_k$  ( $v_i \xrightarrow{1,t_k} v_j$ ), gdzie  $i, j, k \in \{1, \dots, n\}$  (oczywiście  $\sum_{k=1}^n f_{ij}^k = f_{ij}$ ). Suma przepływów wchodzących do wierzchołka początkowego równa się 0, jako że to właśnie w nim na początku znajdują się wszystkie towary, które wierzchołek tylko przekazuje dalej. Jasnym jest, że rozpatrując każdy z towarów osobno  $\sum_{(s,j) \in E} f_{sj}^k = 1$  (gdy  $j \neq s$ , gdyż towar  $t_s$  z założenia znajduje się już na swoim miejscu). Analogicznie, wszystkie wierzchołki  $v_i$  nie będące wierzchołkiem początkowym oczekują na przyjęcie tylko jednego towaru ( $t_i$ ), toteż ich bilans przepływów dla konkretnego towaru  $t_i$  także sumuje się do jedynki. Dla wszystkich pozostałych wierzchołków, które nie są ani wierzchołkiem początkowym, ani docelowym węzłem danego towaru  $t_i$ , suma przepływów wchodzących do danego wierzchołka i z niego wychodzących dla towaru  $t_i$  powinna być taka sama. Właściwości te są zachowane dla wszystkich towarów poza towarem znajdującym się w wierzchołku początkowym ( $t_s$ ). Oczywistym jest również, że w przypadku, gdy dana krawędź nie należy do rozwiązania, przepływ dla tej krawędzi i dowolnego towaru nie występuje ( $y_{ij} = 0 \rightarrow \forall f_{ij}^k = 0$ ), zaś ograniczenie 4.6e redukuje nam przestrzeń dopuszczalnych przez model rozwiązań do drzew rozpinających zadany graf. Dzięki funkcji celu spośród nich wybierane jest zaś rozwiązanie optymalne — minimalne drzewo rozpinające. Warto zwrócić uwagę na fakt, że do problemu, który zdefiniowaliśmy tylko dla grafów nieskierowanych<sup>7</sup>, wykorzystujemy model, który rozróżnia kierunki przepływu na krawędziach (który jest przystosowany do grafów skierowanych). Wiąże się to z koniecznością zastosowania funkcji celu, która za wartość zmiennej decyzyjnej  $x_{ij}$  (z poprzedniego modelu) przyjmuje sumę zmiennych  $y_{ij}$  oraz  $y_{ji}$  — dzięki tak prostemu zabiegowi otrzymujemy wyrażenie, którego wartość jest niezależna od kierunku przepływów i może być traktowana jako wartość rozwiązania problemu minimalnego drzewa rozpinającego dla grafu nieskierowanego.

#### 4.2 Oporne minimalne drzewo rozpinające

Naszym następnym naturalnym krokiem po zapoznaniu się z podstawowymi schematami modelującymi problem minimalnego drzewa rozpinającego jest postawienie pytania o modele, z których pomocą mogliby-

<sup>7</sup>Problem minimalnego drzewa rozpinającego dla grafów skierowanych nosi nazwę optymalnego rozgałęziania (ang. *optimum branching*) i jest także rozwiązywalny w czasie porównywalnym do czasu działania algorytmu Prima dla grafów nieskierowanych [17].)



śmy rozwiązywać bardziej złożone zagadnienia, które są oparte o wyżej omówiony problem. W tej części postaramy się zatem skonstruować modele rozwiązujące problem minimalnego drzewa rozpinającego w wersji INCREMENTAL, a następnie wykorzystamy uzyskane rezultaty do sformułowania twierdzeń, które będą dla nas punktem wyjściowym do stworzenia „tradycyjnego” iteracyjnego algorytmu dla tego problemu.

Przypomnijmy, że problem minimalnego drzewa rozpinającego w wersji INCREMETAL (dalej IMST) stawał przed nami zadanie odnalezienia takiego drzewa  $T^*$  (w obliczu pojawienia się nowych kosztów krawędzi w grafie), którego część wspólna ze starym rozwiązaniem jest dostatecznie duża (jest regulowana przez parametr  $k$ , należącego do obowiązkowego opisu problemu). Zatem, tak jak zdefiniowaliśmy to w 3.4.1, naszym celem jest znalezienie drzewa  $T^*$ , które spełnia

$$\min_{T' \in \mathcal{T}_{T_s^*}^k} v(T', \mathbf{s}'), \quad (4.7)$$

gdzie za  $T_s^*$  przyjmujemy drzewo będące minimalnym drzewem rozpinającym grafu  $G$ , którego koszty krawędzi definiowane są przez scenariusz  $\mathbf{s}$ ,  $\mathcal{T}_{T_s^*}^k$  jest zbiorem zbiorów krawędzi (drzew) zdefiniowanym jako  $\{T : f(T, T_s^*) \leq k\}$  (gdzie  $f : X \times X \rightarrow \mathbb{N}$  była zdefiniowana jako  $f(T^0, T^1) = |T^0 \setminus T^1|^8$ ), zaś my szukamy pośród tego zbioru takiego drzewa  $T^*$ , którego suma kosztów krawędzi dla nowego scenariusza  $\mathbf{s}'$  będzie jak najmniejsza. Przyjmując, że pierwsze rozwiązanie problemu minimalnego drzewa rozpinającego  $T_s^*$  opisują zmienne decyzyjne  $x_{ij}$  ( $T_s^* = \{e_{ij} : x_{ij} = 1\}$ ), nowego zaś —  $y_{ij}$  ( $T_s^* = \{e_{ij} : y_{ij} = 1\}$ ), to bez głębszego zastanowienia możemy napisać następujący model:

$$\min \sum_{e \in E} c_e \cdot y_e \quad (4.8a)$$

$$\text{s.t. } A \cdot \mathbf{y} = \mathbf{b}, \quad (4.8b)$$

$$\sum_{e \in E} |x_e - y_e| \leq 2 \cdot k, \quad (4.8c)$$

$$y_e \geq 0, \quad \forall e \in E, \quad (4.8d)$$

gdzie równanie 4.8b symbolizuje dowolny układ ograniczeń służący nam w poprzednim rozdziale do modelowania problemu minimalnego drzewa rozpinającego. Nierówność 4.8c w naturalny sposób opisuje dodatkowe ograniczenie w problemie typu INCREMENTAL — jesteśmy zainteresowani rozwiązaniami, które nie zawierają więcej niż  $k$  krawędzi, których nie ma w rozwiązaniu bazowym ( $x_{ij} = 0 \wedge y_{ij} = 1 \rightarrow |x_e - y_e| = 1$ )<sup>9</sup>. Przyglądając się tak zdefiniowanemu modelowi od razu możemy dostrzec pierwszy poważny problem — funkcji  $g(x_{ij}, y_{ij}) = |x_{ij} - y_{ij}|$  nie jesteśmy w stanie przedstawić za pomocą dostępnych nam narzędzi, gdyż w zależności od wartości zmiennych decyzyjnych, zmianom ulega nasza macierz współczynników dla modelu (gdy  $x_{ij} \geq y_{ij}$  to współczynniki dla tego ograniczenia to  $[1, -1]$ , zaś w przypadku gdy  $x_{ij} < y_{ij}$  to sytuacja jest odwrotna), co oczywiście nie może mieć miejsca (nie możemy uzależniać struktury modelu od jego własnego rozwiązania). Aby rozwiązać ten problem wprowadźmy dodatkowe zmienne decyzyjne:

- $z_{ij}^+$  — dla każdej krawędzi jej wartość wynosi 1, gdy dana krawędź  $e_{ij} \in T_{s'}^*$  ( $y_{ij} = 1$ ), lecz  $e_{ij} \notin T_s^*$  ( $x_{ij} = 0$ ). W przeciwnym wypadku  $z_{ij}^+ = 0$ . Wartość tak zdefiniowanej zmiennej zatem będzie udzielać informacji czy dana krawędź nie występowała w oryginalnym rozwiązaniu i została dodana do nowego rozwiązania w wyniku zmiany kosztów w grafie,
- $z_{ij}^-$  — analogicznie do wyżej zdefiniowanej zmiennej, wartość tej wskazuje na krawędzie, które zostały usunięte z rozwiązania na rzecz tych, dla których  $z_{ij}^+ = 1$  (w przypadku, gdy dana krawędź  $e_{ij} \notin T_{s'}^*$ , ale  $e_{ij} \in T_s^*$  —  $z_{ij}^- = 1$ ),

które mają następujące własności:

<sup>8</sup>Zbiorem  $X$  w tym przypadku będziemy oznaczać przestrzeń wszystkich możliwych drzew rozpinających graf.

<sup>9</sup>Weźmy pod uwagę, że skoro oba rozwiązania zawierają z definicji tą samą liczbę krawędzi, to fakt pojawienia się łuku należącego tylko do jednego z nich pociąga za sobą fakt pojawienia się w drugim rozaniu krawędzi, która nie jest częścią rozwiązania pierwszego. Innymi słowy jesteśmy zmuszeni do podwojenia wartości parametru  $k$  ( $x_{ij} = 0 \wedge y_{ij} = 1 \rightarrow \exists e_{kl} \in E x_{kl} = 1 \wedge y_{kl} = 0$ ).

- jeśli dana krawędź  $e_{ij}$  należy do pierwotnego rozwiązania (dla starych kosztów grafu), wtedy  $z_{ij}^+ = 0$  (nie możemy ponownie dodać do rozwiązania tego samego łuku — 4.9h),
- jeśli krawędź  $e_{ij}$  nie należy do drzewa  $T_s^*$ , wtedy  $z_{ij}^- = 0$  (analogicznie, nie możemy z rozwiązania usunąć krawędzi, której w nim nie ma — 4.9i),
- nowa zmienna decyzyjna, generująca nam optymalne rozwiązanie problemu IMST ( $T^* = \{e_{ij} : z_{ij} = 1\}$ ) jest zależna od obu tych zmiennych i przybiera formę  $z_e = z_e^+ - z_e^- + x_e$ .

Widzimy, że w łatwy sposób możemy związać ze sobą informacje o zmianach w zbiorze krawędzi będących rozwiązaniem, z informacjami niesionymi przez wektor zmiennych  $\mathbf{x}$  — do modelu przedstawionego w 4.5 dołączymy dodatkowe ograniczenia: 4.9f, 4.9g, 4.9h oraz 4.9i a także za wektor zmiennych wyznaczających rozwiązanie dla modelu przyjmujemy  $\mathbf{z}$  (wektor  $\mathbf{x}$  w tym przypadku symbolizuje stare rozwiązanie). Jak wskazaliśmy wyżej, jedynym miejscem, w którym wiążemy nowe zależności ze starym modelem jest zbiór równości 4.9g (gdybyśmy je usunęły, otrzymywane rozwiązania dla tego modelu niczym by się nie różniły od tych otrzymywanych dla 4.5 — ograniczenia od 4.9f do 4.9i po prostu nie miałyby na rozwiązanie żadnego wpływu). Bardzo łatwo też można dostrzec, że jeżeli  $\forall e_{ij} \in E z_{ij}^+ = 0 \wedge z_{ij}^- = 0$ , to otrzymany model niczym nie różni się od przedstawionego wyżej 4.5 — wartości zmiennych decyzyjnych  $z_{ij}$  będą po prostu przerównane do starych wartości a model bez zastanowienia powinien zwrócić nam identyczne rozwiązanie. W zależności od parametru  $k$  dla problemu IMST wiele ograniczeń zajdzie właśnie w takiej formie —  $z_{ij} = 0 - 0 + x_{ij}$  (jeśli  $k$  jest małe w stosunku do liczby krawędzi w grafie, zaś  $\sum_{e \in E} z_e^+ = \sum_{e \in E} z_e^- = k$ ).

$$\min \sum_{e \in E} c_e \cdot z_e, \quad (4.9a)$$

$$\text{s.t. } \sum_{(i,j) \in E} f_{ij} - \sum_{(j,i) \in E} f_{ji} = \begin{cases} n-1 & \text{jeżeli } i=1, \\ -1 & \text{w przeciwnym przypadku,} \end{cases} \quad (4.9b)$$

$$f_{ij} \leq (n-1) \cdot z_{ij}, \quad \forall \{i,j\} \in E, \quad (4.9c)$$

$$f_{ji} \leq (n-1) \cdot z_{ij}, \quad \forall \{i,j\} \in E, \quad (4.9d)$$

$$\sum_{e \in E} z_e = n-1, \quad (4.9e)$$

$$\sum_{e \in E} z_e^+ + z_e^- \leq 2 \cdot k, \quad (4.9f)$$

$$z_e = z_e^+ - z_e^- + x_e \quad \forall e \in E. \quad (4.9g)$$

$$z_e^+ \leq 1 - x_e \quad \forall e \in E. \quad (4.9h)$$

$$z_e^- \leq x_e \quad \forall e \in E. \quad (4.9i)$$

$$f_e, z_e^+, z_e^- \geq 0, \quad \forall e \in E, \quad (4.9j)$$

$$z_e \in \{0, 1\}, \quad \forall e \in E. \quad (4.9k)$$

Wracając zaś do problemu wyrażenia wartości bezwzględnej w modelu programowania LP/MIP, widzimy że dla tak zdefiniowanych zmiennych zachodzi  $|x_{ij} - y_{ij}| = z_{ij}^+ - z_{ij}^-$  — wystarczy zauważyc, że w przypadku gdy  $x_{ij} < y_{ij}$  ( $e_{ij} \notin T_s^*$  i  $e_{ij} \in T_s^*$ ) bądź  $y_{ij} < x_{ij}$  (krawędź  $e \equiv v_i \rightsquigarrow v_j$  należy tylko do pierwszego bądź tylko do drugiego rozwiązania), suma zmiennych  $z_e^+$  oraz  $z_e^-$  wynosi dokładnie 1 — dzięki ograniczeniom 4.9h oraz 4.9i mamy pewność, że gdy jedna z tych zmiennych równa się 1, w tym czasie wartość drugiej jest z góry ograniczona przez 0, gdzie  $z_e^+$  i  $z_e^-$  są zmiennymi o wartościach nieujemnych. Dodatkowo, aby któraś z nierówności ( $x_e < y_e$  lub  $y_e < x_e$ ) mogła zajść, wartość którejkolwiek ze zmiennych  $z_e^+$  lub  $z_e^-$  musi się różnić od zera — zakładając zaś, że wartości zmiennych  $\mathbf{x}$  ze starego rozwiązania należą do przedziału  $\{0, 1\}$ , oraz że domeny zmiennych należących do wektora  $\mathbf{z}$  wymuszają te same zakresy dopuszczalnych wartości (4.9k),



dochodzimy do wniosku, że jedynymi dopuszczalnymi w takim wypadku wartościami zmiennych  $z_e^+$  lub  $z_e^-$  są te należące do  $\{0, 1\}^{10}$ . Oczywiście, gdy  $|x_e - y_e| = 0$ ,  $x_e = y_e$ , co pociąga za sobą  $z_e^+ = z_e^- = 0$ .

Możemy całkowicie zrezygnować z ograniczenia 4.9g, pozbywając się w ten sposób nadmiaru zmiennych  $z$  modelu oraz znacznie redukując liczbę potrzebnych ograniczeń, tak jak to pokazano poniżej (4.11), gdzie w miejsce zmiennych  $z_e$  dla każdej krawędzi pojawiły się jego podwyrażenia z poprzedniego modelu —  $\sum_{e \in E} c_e \cdot x_e + \sum_{e \in E} c_e \cdot (z_e^+ - z_e^-) = \sum_{e \in E} c_e \cdot z_{ij}$ , gdzie  $z_{ij}$  to zmienne z poprzedniego modelu opisane przez 4.9g. Równość 4.11b także pokrywa się z równością 4.8b poprzedniego modelu:

$$A \cdot \mathbf{x} = b \wedge A \cdot \mathbf{z} = b \Leftrightarrow A \cdot (\mathbf{z}^+ - \mathbf{z}^- + \mathbf{x}) = b \Leftrightarrow A \cdot (\mathbf{z}^+ - \mathbf{z}^-) = \mathbf{0}, \quad (4.10)$$

zaś pozostałe ograniczenia wyrażają te same zależności co wcześniej.

$$\min \sum_{e \in E} c_e \cdot x_e + \sum_{e \in E} c_e \cdot (z_e^+ - z_e^-), \quad (4.11a)$$

$$\text{s.t. } A \cdot (z_e^+ - z_e^-) = \mathbf{0}, \quad (4.11b)$$

$$\sum_{e \in E} z_e^+ + z_e^- \leq 2 \cdot k, \quad (4.11c)$$

$$z_e^- \leq x_e \quad \forall e \in E. \quad (4.11d)$$

$$z_e^+, z_e^- \geq 0, \quad \forall e \in E. \quad (4.11e)$$

### 4.2.1 Relaksacja Lagrange'a

W celu efektywnego rozwiązywania modeli programowania liniowego/całkowitoliczbowego możemy posłużyć się jeszcze jednym bardzo ciekawym narzędziem, jakim jest **relaksacja** ograniczeń. Przyglądając się modelowi zdefiniowanemu w 4.8 (przytoczymy go poniżej)

$$\min \sum_{e \in E} c_e \cdot y_e \quad (4.12a)$$

$$\text{s.t. } A \cdot \mathbf{y} = \mathbf{b}, \quad (4.12b)$$

$$\sum_{e \in E} |x_e - y_e| \leq 2 \cdot k, \quad (4.12c)$$

$$y_e \geq 0, \quad \forall e \in E, \quad (4.12d)$$

możemy zauważyć, że gdyby nie obecność ograniczenia 4.12c, prezentowany model niczym nie różniłby się od dowolnego modelu rozwiązywanego problem minimalnego drzewa rozpinającego (byłyby dokładne jego uogólnionym zapisem — jak powiedzieliśmy przy omawianiu tego modelu, równość 4.12b symbolizuje ograniczenia dla tego problemu). Naturalnym pytaniem zatem jest czy możemy za pomocą posiadanego już modelu rozwiązywać inne, podobne do siebie problemy? Zanim odpowiemy na to pytanie, przekształćmy jeszcze model 4.12 do najprostszej, najbardziej wymownej postaci jaką możemy uzyskać — zauważmy, że zgodnie z definicją funkcji  $f : X \times X \rightarrow \mathbb{N}$ , jej wartość zależy tylko od dwóch zbiorów krawędzi (w naszym przypadku od drzew  $T_s^*$  oraz  $T^*$ , gdzie pierwszym z nich jest optymalne minimalne drzewo rozpinające dla grafu z kosztami krawędzi zgodnymi ze scenariuszem początkowym  $s$ , drugie zaś jest optymalnym rozwiązaniem problemu IMST). Skoro zaś  $f(T_s^*, T^*) = |T^* \setminus T_s^*|$  (oraz  $f(T_s^*, T^*) = |T_s^* \setminus T^*|^{11}$ ) to możemy dojść do wniosku, że wyliczanie wartości  $|x_e - y_e|$  dla wszystkich krawędzi nie jest najszybszym sposobem narzucenia ograniczeń zgodnie z wymogami problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL. Szybko możemy

<sup>10</sup>Dla przykładu: jeśli  $z_{ij} = 1$ ,  $x_{ij} = 0$ , to wiemy że  $z_e^+ - z_e^- = 1$ , zaś z ograniczenia 4.9i wiemy dodatkowo, że  $z_e^- \leq 0$ , co razem z 4.9j ( $z_e^- \geq 0$ ) doprowadza nas do jedynego słusznego wniosku, że  $z_e^+ = 0$ .

<sup>11</sup>Zatem kolejność argumentów funkcji  $f : X \times X \rightarrow \mathbb{N}$  nie ma żadnego znaczenia.

zauważać, że tak naprawdę jesteśmy zainteresowani tylko wartościami zmiennymi decyzyjnymi dla dowolnego z podanych zbiorów krawędzi:  $T^* \setminus T_s^*$  lub  $T_s^* \setminus T^*$ , a dokładniej ich sumą:

$$\sum_{e \in E} |x_e - y_e| \leq 2 \cdot k \Leftrightarrow \sum_{e_i \in E \setminus T_s^*} z_i \leq k \Leftrightarrow \sum_{e_i \in T^* \setminus T_s^*} z_i \leq k.^{12} \quad (4.13)$$

Przyjrzyjmy się zatem następującemu modelowi<sup>13</sup>.

$$\min \sum_{e \in E} c_e \cdot x_e \quad (4.14a)$$

$$\text{s.t. } A \cdot \mathbf{x} = \mathbf{b}, \quad (4.14b)$$

$$\sum_{e \in T^* \setminus T_s^*} x_e \leq k, \quad (4.14c)$$

$$x_e \geq 0, \forall e \in E, \quad (4.14d)$$

a następnie dokonajmy **relaksacji** jego ograniczeń — innymi słowy usuniemy z niego ograniczenie 4.14c, tak aby powstały w ten sposób model był identyczny co model rozwiązujący problem minimalnego drzewa rozpinającego, a następnie dodamy to ograniczenie do wartości funkcji celu:

$$\min \sum_{e \in E} c_e \cdot x_e + \lambda \cdot \left( \sum_{e \in T^* \setminus T_s^*} x_e - k \right) \quad (4.15a)$$

$$\text{s.t. } A \cdot \mathbf{x} = \mathbf{b}, \quad (4.15b)$$

$$x_e \geq 0, \quad \forall e \in E, \quad (4.15c)$$

gdzie  $\lambda$  jest parametrem. Idea stojąca za takim postępowaniem jest bardzo prosta. Założymy, że ograniczenie 4.14c nie zaszło (czyli  $\sum_{e \in T^* \setminus T_s^*} x_e > k$ ) — sami na to zezwoliliśmy, usuwając dane ograniczenie z modelu. Bezpośrednią konsekwencją niespełnienia danego ograniczenia jest wzrost wartości rozwiązania, jaką zwróci nam tak zmodyfikowana funkcja celu — zwracana wartość będzie tym większa, im bardziej będziemy oddalać się od przestrzeni rozwiązań dopuszczalnych ( $\sum_{e \in T^* \setminus T_s^*} x_e$  będzie coraz bardziej oddalone od  $k$ ) lub im większą wartość parametru  $\lambda$  przyjmiemy. Naszą motywacją w tak podjętym działaniu jest naturalne wymuszenie na modelu odrzucania niedopuszczalnych rozwiązań (ściślej mówiąc — traktowania ich jako rozwiązanie nieoptymalne, dla których funkcja celu nie zwraca najmniejszej wartości spośród wszystkich dostępnych rozwiązań) przy jednoczesnym zachowaniu uproszczonego modelu. Pokażemy teraz, słuszność takiego postępowania w ogólnym przypadku.

**Lemat 4.2.1** *Niech dany będzie problem optymalizacyjny  $\mathcal{P}$ , do którego zastosowano relaksację Lagrange'a, relaksując ograniczenia w postaci  $A \cdot \mathbf{x} \leq b$ . Niech  $\mathbf{x}^*$  będzie optymalnym rozwiązaniem takiego problemu dla pewnego wektora  $\boldsymbol{\lambda} \geq \mathbf{0}$ . Jeśli  $\mathbf{x}^*$  jest rozwiązaniem dopuszczalnym w  $\mathcal{P}$  i spełnia założenia komplementarności  $\boldsymbol{\lambda} \cdot (A \cdot \mathbf{x}^* - b) = \mathbf{0}$ , wtedy rozwiązanie  $\mathbf{x}^*$  jest rozwiązaniem optymalnym dla problemu  $\mathcal{P}$ .*

W naszym przypadku, jak można było zauważać w modelu 4.15, użyty przez nas parametr  $\lambda$  nie jest wektorem (zwykliśmy je oznaczać pogrubioną czcionką), co ma swoje uzasadnienie w tym, że wartość przez którą go mnożymy też nim nie jest (usunięte z modelu 4.14 ograniczenie nie jest aggregatorem dla większej liczby ograniczeń — tyczy się pojedynczej sumy). Naszym celem zatem jest pokazać, że o ile model 4.15 zwróci rozwiązanie spełniające podane w lematce kryteria, będzie ono szukanym przez nas rozwiązaniem optymalnym,

<sup>12</sup>Zauważaliśmy tutaj, że sumy  $\sum_{e_i \in E \setminus T_s^*} z_i$  oraz  $\sum_{e_i \in T^* \setminus T_s^*} z_i$  są identyczne — wynika to z prostego faktu, że dla wszystkich krawędzi  $e \in E \setminus T^*$   $z_i = 0$ .

<sup>13</sup>Jako że nie będziemy już rozróżniać zmiennych z wektorów  $\mathbf{x}$ ,  $\mathbf{y}$  i  $\mathbf{z}$  (odpowiednio jako zmienne wskazujące krawędzie należące do zbiorów  $T_s^*$ ,  $T_{s'}^*$  — dla problemu MST — oraz  $T^*$  — dla problemu IMST), w dalszej części będziemy na powrót posługiwać się oznaczeniami  $\mathbf{x}$  by podkreślić odrębność omawianego zagadnienia.



co pozwoli nam się następnie skupić na sposobie obliczania samego parametru  $\lambda$  dla minimalizującej funkcji celu.

**Dowód.** Niech dany będzie model w postaci macierzowej:

$$v^* = \min \mathbf{c} \cdot \mathbf{x} \quad (4.16a)$$

$$\text{s.t. } A \cdot \mathbf{x} = \mathbf{b}, \quad (4.16b)$$

$$x_i \in X, \quad (4.16c)$$

którego postać po relaksacji Lagrange'a jest następująca:

$$L(\boldsymbol{\lambda}) = \min \mathbf{c} \cdot \mathbf{x} + \boldsymbol{\lambda} \cdot (A \cdot \mathbf{x} - \mathbf{b}) \quad (4.17a)$$

$$\text{s.t. } x_i \in X. \quad (4.17b)$$

Zauważmy oczywisty fakt, że zbiory opisane przez 4.16 oraz 4.17 łączy wzajemna relacja zawierania się, to jest zbiór rozwiązań opisany przez  $\{\mathbf{x} : A \cdot \mathbf{x} = \mathbf{b}\} \subseteq \{\mathbf{x}\}$  (drugi model nie narzuca nam żadnych ograniczeń co do rozwiązania). Wiemy zatem, że pomiędzy wartościami dla rozwiązań  $\mathbf{x}^*$  a  $L(\boldsymbol{\lambda})$  zachodzi prawidłowość:

$$v^* = \min_{\mathbf{x} \in X} \{\mathbf{c} \cdot \mathbf{x} : A \cdot \mathbf{x} = \mathbf{b}\} \geq \min_{\mathbf{x} \in X} \{\mathbf{c} \cdot \mathbf{x} + \boldsymbol{\lambda} \cdot (A \cdot \mathbf{x} - \mathbf{b})\} = L(\boldsymbol{\lambda})^{14} \quad (4.18)$$

Możemy zatem o wartości  $L(\boldsymbol{\lambda})$  powiedzieć, że jest dolnym ograniczeniem wartości  $v^*$ , jako że zawsze  $L(\boldsymbol{\lambda}) \leq v^*$ . Wartość wyrażenia  $v^*$  jest najmniejszą spośród wszystkich innych rozwiązań ( $v^* = \mathbf{c} \cdot \mathbf{x}^*$ , gdzie  $\mathbf{x}^*$  jest optymalnym rozwiązaniem problemu opisanego przez model 4.16), zatem  $L(\boldsymbol{\lambda})$  — jeśli jest mniejsza od  $v^*$  — jest rozwiązaniem na pewno niedopuszczalnym. Co nas w związku z tym powinno interesować to jak najdokładniejsze dolne ograniczenie na wartość  $v^*$ :  $L^* = \max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$ , które spełnia  $L(\boldsymbol{\lambda}) \leq L^* \leq v^*$ . Dodatkowo z faktu optymalności rozwiązania  $\mathbf{x}^*$  wiemy, że dla dowolnego  $\mathbf{x} \in X$  zachodzi także  $L(\boldsymbol{\lambda}) \leq L^* \leq v^* = \mathbf{c} \cdot \mathbf{x}^* \leq \mathbf{c} \cdot \mathbf{x}$ . Założymy teraz, że rozwiązanie  $\mathbf{x} \in X$  spełnia warunek komplementarności, to znaczy  $\boldsymbol{\lambda} \cdot (A \cdot \mathbf{x} - \mathbf{b}) = \mathbf{0}$ , w związku z tym wyrażenie  $L(\boldsymbol{\lambda})$  upraszcza się do  $\mathbf{c} \cdot \mathbf{x}$ . Jeśli, zgodnie z lematem, przedstawione rozwiązanie  $\mathbf{x}$  o takich własnościach jest dopuszczalne to znaczy, że jego wartość na pewno nie jest mniejsza od wartości rozwiązania reprezentowanego przez wektor zmiennych decyzyjnych  $\mathbf{x}^*$ . W związku z tym mamy, że:

$$L(\boldsymbol{\lambda}) = \boldsymbol{\lambda} \cdot (A \cdot \mathbf{x} - \mathbf{b}) = \mathbf{c} \cdot \mathbf{x} \quad (4.19)$$

$$L(\boldsymbol{\lambda}) \leq L^* \leq v^* = \mathbf{c} \cdot \mathbf{x}^* \leq \mathbf{c} \cdot \mathbf{x} = L(\boldsymbol{\lambda}). \quad (4.20)$$

Z powyższego zaś wynika, że drugi z podanych wzorów przedstawia szereg równości, tak więc ostatecznie otrzymujemy, że  $L(\boldsymbol{\lambda}) = v^* = \mathbf{c} \cdot \mathbf{x}$ , gdzie  $\mathbf{x} = \mathbf{x}^*$ .  $\blacklozenge$

Powyższy lemat jest bardzo potężnym narzędziem — okazuje się bowiem, że dzięki spełnieniu warunku komplementarności możemy bezkarnie uprościć nasz model rozwiązywający problem minimalnego drzewa rozpinającego w wersji INCREMENTAL<sup>15</sup>. W naszym przypadku zastosowanie tej metody jest bardzo proste, gdyż liczba relaksowanych przez nas ograniczeń to zaledwie 1 — w związku z tym przedstawimy teraz cały proces transformacji modelu rozwiązywującego problem IMST do prostszego, radzącego sobie z problemem MST, zwracającego zaś te same rozwiązania.

### Minimalne drzewo rozpinające w wersji incremental

W modelu 4.15 w wyniku relaksacji ograniczeń otrzymaliśmy następujące wyrażenie:

<sup>14</sup>Optymalne rozwiązanie (o najmniejszej wartości) może znajdować się w zbiorze większym zaś nie należeć do zbioru będącego tylko jego podzbiorem.

<sup>15</sup>Powstało bardzo dużo prac naukowych na temat użyteczności zaprezentowanego lematu i polach do zastosowania relaksacji Lagrange'a, których nie będziemy tu przytaczać — głównym polem, na którym eksplotowana jest ta metoda, jest problem transformacji modeli programowania IP/MIP do postaci LP tak, aby bez pogorszenia stopnia użyteczności danego modelu móc uzyskiwać rozwiązania problemów w rozsądny, wielomianowym czasie.



$$\sum_{e \in E} c_e \cdot x_e + \lambda \cdot \left( \sum_{e \in T^* \setminus T_s^*} x_e - k \right) \dots \quad (4.21)$$

Aby móc je uprościć by miało dla nas wartość merytoryczną, musimy pozbyć się z niego wyrażenia  $T^*$ , które definiuje nam zbiór krawędzi będących elementami minimalnego drzewa rozpinającego w wersji INCREMENTAL (w funkcji celu pozwalającej nam na odnalezienie optymalnego rozwiązania problemu IMST pojawia nam się samo rozwiązanie). Aby wyeliminować tą zależność między powyższym wzorem a rozwiązaniem  $T^*$  spójrzmy jeszcze raz na ograniczenie, które wciążemy do wyrażenia funkcji celu dla modelu. Możemy zauważać, że

$$\sum_{e \in T^* \setminus T_s^*} x_e \leq k \rightarrow \sum_{e \in T_s^*} x_e \geq n - 1 - k. \quad (4.22)$$

Powyższa implikacja wynika z faktu, że suma zmiennych  $x_i$  dla dowolnego drzewa rozpinającego graf o  $n$  wierzchołkach jest równa co najwyżej  $n - 1$  (dla zmiennych binarnych), zatem jeżeli liczba krawędzi  $e$  należących do takiego rozwiązania, gdzie  $e \notin T_s^*$  nie przekracza  $k$ , to pozostałych krawędzi w rozwiązaniu ( $e \in T_s^*$ ) musi być tyle, aby suma ich liczebności dawała  $n - 1$ . Bardziej formalnie:

$$k \geq \sum_{e \in T^* \setminus T_s^*} x_e \leftrightarrow |e : e \in T^* \setminus T_s^* \wedge x_e = 1| \leftrightarrow \quad (4.23)$$

$$\leftrightarrow |e : e \in T^* \wedge x_e = 1| - |e : e \in T_s^* \wedge x_e = 1| \quad (4.24)$$

$$|e : e \in T^* \wedge x_e = 1| = n - 1 \quad (4.25)$$

$$\sum_{e \in T_s^*} x_e = |e : e \in T_s^* \wedge x_e = 1| \leftrightarrow \quad (4.26)$$

$$\leftrightarrow |e : e \in T^* \wedge x_e = 1| - |e : e \in T^* \setminus T_s^* \wedge x_e = 1| \geq (n - 1) - k. \quad (4.27)$$

Korzystając zatem z równoważności powyższych warunków możemy przekształcić równanie 4.21 w następujący sposób:

$$\sum_{e \in E} c_e \cdot x_e + \lambda \cdot \left( \sum_{e \in T^* \setminus T_s^*} x_e - k \right) \leftrightarrow \quad (4.28)$$

$$\leftrightarrow \sum_{e_i \in E} c_i \cdot x_i + \lambda \cdot \left( (n - 1 - k) - \sum_{e_i \in T_s^*} x_i \right) \stackrel{(1)}{\leftrightarrow} \quad (4.29)$$

$$\stackrel{(1)}{\leftrightarrow} \left[ \sum_{e_i \in E \setminus T_s^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} c_i \cdot x_i \right] + \lambda \cdot \left( - \sum_{e_i \in T_s^*} x_i \right) \leftrightarrow \quad (4.30)$$

$$\leftrightarrow \sum_{e_i \in E \setminus T_s^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} c_i \cdot x_i - \sum_{e_i \in T_s^*} \lambda \cdot x_i \leftrightarrow \quad (4.31)$$

$$\leftrightarrow \sum_{e_i \in E \setminus T_s^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} (c_i - \lambda) \cdot x_i, \quad (4.32)$$

gdzie w kroku oznaczonym jako (1) pozbyliśmy się stałego wyrażenia  $\lambda \cdot (n - 1 - k)$ , które nie wpływa w żaden sposób na rozwiązanie (dla każdego wektora  $\mathbf{x}$ , który możemy podstawić do otrzymanego wzoru, relatywna wartość funkcji dla rozwiązań  $\mathbf{x}$  i  $\mathbf{x}'$  się nie zmienia) oraz rozobiliśmy sumę zmiennych decyzyjnych dla zbioru  $e$  na dwie części.



### 4.3 Podsumowanie rozdziału

Otrzymaliśmy zatem dokładny przepis na rozwiązywanie problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL. Tak jak zapowiadaliśmy na początku tego rozdziału, w następnym wykorzystamy zgromadzone informacje do implementacji algorytmu nie polegającego na zagadnieniu programowania całkowito-liczbowego. Wykorzystując podane przez nas warunki optymalności rozwiązania:

$$f(T^*, T_s^*) \leq k \wedge \lambda = 0 \quad \text{lub} \quad (4.33)$$

$$f(T^*, T_s^*) = k \wedge \lambda \neq 0, \quad (4.34)$$

oraz policzone wyżej wyrażenie będące definicją funkcji celu dla zrelaksowanego problemu IMST, podamy algorytm, który następnie posłuży nam do rozwiązywania kolejnych, bardziej złożonych problemów: adwersarza (AMST) oraz odpornego drzewa rozpinającego z możliwością poprawy (RRMST). W tym rozdziale natomiast mieliśmy okazję zapoznać się z terminem liniowego programowania, poznać jego zalety, wady jak i ograniczenia. Przygotowaliśmy dla siebie naprawdę potężne narzędzia do pracy z modelami i wykorzystaliśmy je by otrzymać przedstawione powyżej wyniki.





# Algorytm odpornej optymalizacji regeneracyjnej

---

W tym rozdziale skonstruujemy efektywny algorytm rozwiązujący jedno z zagadnień odpornej optymalizacji dyskretnej — problem minimalnego drzewa rozpinającego dla okresowo zmieniających się kosztów z ograniczoną możliwością modyfikacji pierwotnego rozwiązania (problem INCREMENTAL MINIMUM SPANNING TREE). Mając w ręku odpowiednie narzędzia przekonamy się, że konstrukcja takiego algorytmu jest prosta, w oparciu o wcześniej przedstawione pomysły będziemy mogli sprowadzić całe zagadnienie do prostego wyszukiwania binarnego, pokazać słuszność takiego postępowania oraz sposoby na jego udoskonalenie.

## 5.1 Konstrukcja algorytmu i warunki optymalności

Konstruowanie algorytmu rozwiązującego zagadnienie typu INCREMENTAL dla minimalnego drzewa rozpinającego rozpoczęmy od powtórzenia (4.32) wyniku, otrzymanego w poprzednim rozdziale po zastosowaniu relaksacji Lagrange'a dla modelu 4.8:

$$\min \left\{ \sum_{e_i \in E \setminus T_s^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} (c_i - \lambda) \cdot x_i \right\}, \quad (5.1)$$

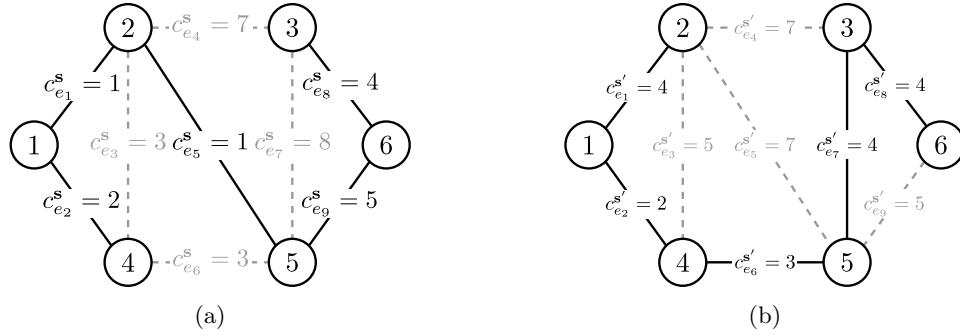
gdzie przypomnijmy, że  $T_s^*$  oznaczał zbiór krawędzi należący do optymalnego rozwiązania problemu minimalnego drzewa rozpinającego dla starych kosztów w grafie, zdefiniowanych przez scenariusz  $s$ .

Pokazaliśmy, że dla problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL z początkowym minimalnym drzewem rozpinającym  $T_s^*$  i parametrem  $k$  przedstawiającym liczbę dopuszczalnych zmian krawędzi względem pierwotnego rozwiązania, w wyniku zmiany kosztów w grafie, z twierdzeniem o różnicach dopełniających zastosowanego do tak opisanego problemu (4.3), wynika, że optymalnym jego rozwiązaniem jest zbiór krawędzi, który zawiera dokładnie  $k$  łuków nie należących do początkowego rozwiązania  $T_s^*$ . Otrzymany wynik jest w pełni zgodny z tym, czego byśmy oczekiwali podchodząc do zadanego problemu, opierając się tylko o nasze intuicje — niech  $N(T, k)$  oznacza zbiór wszystkich drzew rozpinających różniących się od drzewa  $T$  co najwyżej  $k$  krawędziami, w szczególności  $N(T, 0) = \{T\}$ . Niech dane będzie  $k' < k$ . Nietrudno zauważać, że  $N(T, k') \supseteq N(T, k)$ , zatem jeżeli optymalne rozwiązanie problemu IMST  $T^*$  znajduje się w zbiorze  $N(T, k')$ , tym bardziej zawiera się w zbiorze większym —  $N(T, k)$ . Odwrotna relacja oczywiście nie zachodzi — rozwiązanie należące do zbioru  $N(T, k)$ , niekoniecznie musi należeć także do zbioru mniejszego. Aby zatem mieć pewność, że znajdziemy rozwiązanie optymalne dla problemu IMST z parametrem  $k$ , musimy szukać go wśród drzew rozpinających graf o takiej samej liczbie nowych krawędzi względem starego rozwiązania (innymi słowy, co wydaje się oczywiste, wśród wszystkich możliwych dopuszczalnych rozwiązań problemu, a te należą do zbioru  $N(T, k)$  — rozwiązania, które charakteryzują się większą liczbą łuków niż  $k$  są niedopuszczalne). Opierając się o wcześniej przypomniany wzór 5.1 możemy zatem zauważać, że rozsądna strategią na szukanie optymalnego rozwiązania dla badanego zagadnienia jest taka manipulacja parametrem  $\lambda$ , aby tak zdefiniowane tymczasowe koszty grafu dla nowego scenariusza  $s'$ :

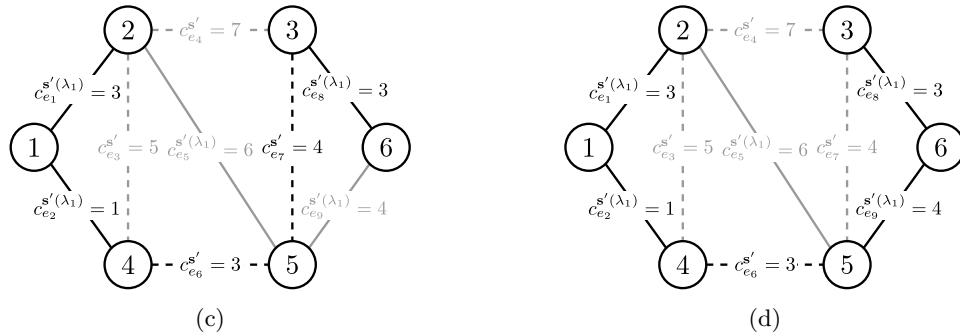
$$c_i^{s'(\lambda)} = \begin{cases} c_i^{s'} & \text{gdy } e_i \text{ nie należy do } T_s^*, \\ c_i^{s'} - \lambda & \text{gdy } e_i \text{ jest krawędzią należącą do oryginalnego rozwiązania,} \end{cases} \quad (5.2)$$



dla każdej krawędzi w grafie stopniowo wymuszały na algorytmie rozwiązyującym problem minimalnego drzewa rozpinającego rozwiążanego, których część wspólna ze starym rozwiązaniem jest coraz większa. Innymi słowy, zmniejszając koszty poszczególnych krawędzi w grafie podnosimy ich szansę na to, że znajdą się w zbiorze krawędzi minimalnego drzewa rozpinającego, jako że ich koszty w pewnym momencie staną się mniejsze od tych, które do tej pory trafiały do tego zbioru. Taką sytuację (oraz problemy związane z takim podejściem) doskonale odzwierciedlają rysunki od 5.1a do 5.1f. Przypomnijmy tylko, że liniami ciągłymi dla problemu IMST oznaczaliśmy oryginalne minimalne drzewo rozpinające  $T_s^*$ , zaś kolorem czarnym — aktualnie przedstawiane na rysunku rozwiązanie (to znaczy, że czarną linią przerywaną są zaznaczone te krawędzie, które w wyniku zmiany kosztów krawędzi w grafie weszły do rozwiązania problemu IMST  $T^*$  i nie należą do  $T_s^*$ , zaś szara linia ciągła oznacza krawędź, która z tych samych przyczyn nie należy do znalezionej rozwiązania, jest natomiast elementem zbioru  $T_s^*$ ). Pozwoli nam to na łatwe stwierdzenie, czy przedstawiane rozwiązanie jest dopuszczalne, optymalne, czy nie należące do zbioru dopuszczalnych rozwiązań (odpowiednio  $f(T^*, T_s^*) = |T^* \setminus T_s^*| = |T_s^* \setminus T^*| \leq k$ ,  $f(T^*, T_s^*) = k$ ,  $f(T^*, T_s^*) > k$ ).



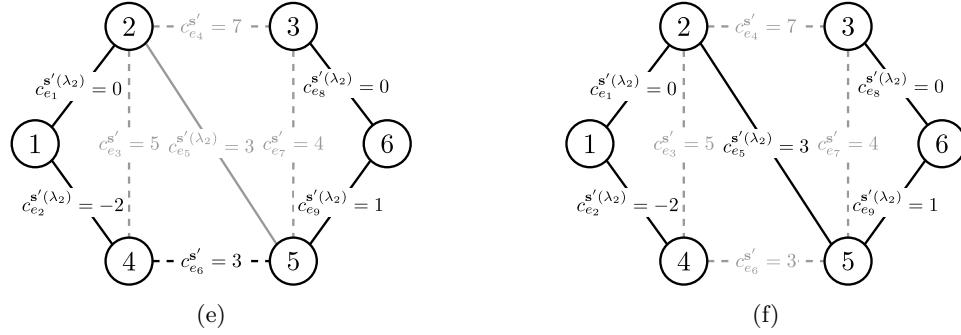
Rysunek 5.1: (a) Optymalne rozwiązanie problemu MST dla scenariusza  $s = [1, 2, 3, 7, 1, 3, 8, 4, 5]$ . Całkowity jego koszt wynosi 13. Krawędzie do niego należące tworzą początkowe rozwiązanie dla problemu IMST z parametrem  $k = 1$ :  $T_s^* = \{e_1, e_2, e_5, e_8, e_9\}$ . (b) Nieograniczone parametrem  $k$ , optymalne rozwiązanie problemu MST dla scenariusza  $s' = [4, 2, 5, 7, 7, 3, 4, 4, 5]$ , gdzie pochyloną czcionką zaznaczono wagi krawędzi, które uległy zmianie względem scenariusza  $s$ . Wartość takiego rozwiązania wynosi 17 i jest tym samym rozwiązaniem jakie otrzymalibyśmy dla scenariusza  $s'(\lambda)$  w przypadku, gdy  $\lambda = 0$ . Należące do tego rozwiązania krawędzie tworzą zbiór  $T_{s'}^* = \{e_1, e_2, e_6, e_7, e_8\}$ , gdzie krawędzie  $e_6$  i  $e_7$  nie należą do  $T_s^*$  ( $f(T_{s'}^*, T_s^*) = 2 > k$ ).



Rysunek 5.1: (c) Rozwiązanie optymalne problemu MST dla wygenerowanego scenariusza  $s'(\lambda_1) = [3, 1, 5, 7, 6, 3, 4, 3, 4]$ , gdzie pogrubioną czcionką zaznaczono, obniżone o współczynnik  $\lambda_1$  względem scenariusza  $s'$  i rozwiązania początkowego  $T_s^*$ , koszty na podstawie których algorytm rozwiązyujący problem minimalnego drzewa rozpinającego zwrócił nam przedstawione rozwiązanie  $T^1 = \{e_1, e_2, e_6, e_7, e_8\}$ . Fikcyjny koszt takiego rozwiązania wynosi 14. (d) Inne optymalne rozwiązanie dla scenariusza  $s'(\lambda_1)$ .

Po przyjrzeniu się bliżej rysunkom 5.1c oraz 5.1d zauważamy pierwszy poważny problem w naszej strategii — algorytm do rozwiązywania problemu minimalnego drzewa rozpinającego (w tym przypadku celowo zastosowaliśmy algorytm Prima) ma skłonność do zwracania różnych odpowiedzi. Co gorsza, w kontekście problemu

IMST rozwiązania te nie są takie same, pomimo że ich koszt jest identyczny. Zauważmy, że o ile w przypadku 5.1c dla scenariusza  $\mathbf{s}'(1)$  zostało zwrócone identyczne rozwiązanie co na rysunku 5.1b (niedopuszczalne, jako że zawierające więcej niż jedną krawędź nienależącą do  $T_s^*$ ), to już rozwiązanie zwrócone na rysunku 5.1d jest rozwiązaniem dopuszczalnym (co więcej, na mocy wcześniejszych lematów jest ono rozwiązaniem optymalnym). Skąd wzięła się ta różnica? Jeśli wróćmy do przedstawionego pseudokodu (2), zobaczymy że algorytm Prima jako parametr wejściowy przyjmuje arbitralnie wybrany przez nas węzeł początkowy. Jako że w przypadku klasycznego problemu minimalnego drzewa rozpinającego jesteśmy zainteresowani jedynie sumą kosztów krawędzi zwracanego rozwiązania, nie ma dla nas różnicy, od którego węzła algorytm zacznie jego konstruowanie (zwykle po prostu jest to pierwszy węzeł, jeśli wierzchołki grafu przechowujemy w dowolny sposób uporządkowanej strukturze danych). Skutkuje to, tak jak pokazano na rysunkach 5.1c i 5.1d, w przypadku których wierzchołkami startowymi są odpowiednio wierzchołki  $v_1$  i  $v_6$ , dwoma różnymi rozwiązaniami. Jakby tego było mało, na rysunkach 5.1e i 5.1f przedstawiono podobną sytuację, w której dla tego samego parametru  $\lambda_2 = 4$ , znalezionymi rozwiązaniami są odpowiednio: rozwiązanie optymalne  $T^2 = \{e_1, e_2, e_5, e_8, e_9\}$  o koszcie równym 18 oraz dopuszczalne o sumie kosztów krawędzi należących do rozwiązania równej 22 (koszt obydwoi rozwiązań dla tymczasowego scenariusza  $\mathbf{s}'(\lambda_2)$  są identyczne i wynoszą 2). Ponownie otrzymaliśmy dwa różne rozwiązania tylko dlatego, że za początkowy wierzchołek obraliśmy  $v_1$  (rysunek 5.1e) oraz  $v_2$  (5.1f).



Rysunek 5.1: (e) Minimalne drzewo rozpinające grafu dla parametru  $\lambda_2 = 4$  i wygenerowanego na jego podstawie scenariusza  $\mathbf{s}'(\lambda_2) = [0, -2, 5, 7, 3, 3, 4, 0, 1]$ ). (f) Inne optymalne rozwiązanie dla scenariusza  $\mathbf{s}'(\lambda_2)$ . Koszt optymalnego rozwiązania wynosi 2.

Widzimy zatem, że tak opisana strategia szukania rozwiązania dla problemu IMST posiada wadę, która czyni naszą strategię bezużyteczną, jako że typ zwracanego rozwiązania (optymalny, dopuszczalny bądź niedopuszczalny) nie zależy w pełni od świadomie wybieranego parametru  $\lambda$  lecz od pośredniczącego algorytmu wyliczającego minimalne drzewo rozpinające. Należy podkreślić, że problem ten nie dotyczy tylko algorytmu Prima — zdecydowaliśmy się na ten algorytm, gdyż na jego przykładzie najdokładniej widać istotę problemu. Problem ten będzie występował (może występować) zawsze, gdy w grafie, dla którego chcemy znaleźć minimalne drzewo rozpinające, będą występować co najmniej dwie krawędzie o takich samych kosztach, niezależnie od wybranego algorytmu.

## 5.2 Struktura drzewa i koszty krawędzi

Aby rozwiązać powyższy problem, będziemy chcieli zaburzyć koszty grafu w taki sposób, aby zapewnić sobie unikalność wag wszystkich krawędzi w grafie, nie zaburzając jednocześnie ich kolejności występowania tj. dla grafu  $G = (V, E)$  i dla kosztów krawędzi w grafie  $\mathbf{s} = [c_i : \forall e_i \in E]$ , gdzie założymy dodatkowo, że  $\exists e, e' \in E : c_e = c_{e'} \wedge e \neq e'$ , będziemy chcieli wygenerować nowe koszty  $\mathbf{s}' = [c'_i : \forall e_i \in E]$ , takie że  $\forall e, e' \in E : e \neq e' \implies c'_e \neq c'_{e'}$  a dodatkowo  $\forall e, e' \in E : c_e < c_{e'} \implies c'_e < c'_{e'}$ . Zanim jednak zdefiniujemy takie przekształcenie, poczytajmy inną obserwację.

**Lemat 5.2.1** [20] Niech drzewa  $T_s^*$  i  $T_{s'}^*$  oznaczają odpowiednio optymalne rozwiązania dla problemu MST i scenariuszy  $\mathbf{s}$  oraz  $\mathbf{s}'$  problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL z parametrem  $k$ .



Dla dowolnego  $T_{s'}^*$  w grafie  $G = (V, E)$  istnieje minimalne drzewo rozpinające dla problemu INCREMENTAL  $T^k$  takie, że wszystkie jego krawędzie należą do zbioru  $E^* = T_{s'}^* \cup T_s^*$ .

Z lematu zatem wynika, że aby znaleźć optymalne rozwiązanie dla problemu IMST, możemy zamiast grafu  $G = (V, E)$ , wziąć pod uwagę dużo rzadszy graf  $G^* = (V, E^*)$ . Wynika to z faktu, że o ile istnieje optymalne rozwiązanie  $T^k$  dla omawianego problemu w grafie  $G$ , takie samo rozwiązanie istnieje w  $G^*$ , zaś po drodze do rozwiązania ani razu nie będziemy brali pod uwagę krawędzi spoza zbioru  $E^*$  (upraszczając, naszym celem jest wyjście od rozwiązania optymalnego  $T_{s'}^*$  dla nowego scenariusza  $s'$  i stopniowe zamienianie wybranych jego krawędzi na te należące do drzewa  $T_s^*$  — nie ma więc mowy, abyśmy potrzebowali krawędzi spoza zbioru  $T_{s'}^* \cup T_s^* = E^*$ ). Ta obserwacja pozwala nam często w znacznym stopniu zredukować strukturę grafu, gdyż w najgorszym przypadku moc nowo otrzymanego zbioru krawędzi będzie wynosić  $|E^*| = 2 \cdot n - 2$  (w przypadku gdy oba zbiory krawędzi są rozłączne, zaś każdy z nich z definicji bycia drzewem zawiera dokładnie  $|V| - 1$  krawędzi, gdzie  $|V| = n$ ). Nawet w takim przypadku, nasza obserwacja może okazać się wielce pomocna, na przykład gdy mamy do czynienia z grafem pełnym — w takim przypadku możemy mówić o zredukowaniu mocy zbioru krawędzi do co najmniej  $\frac{2 \cdot (n-1)}{\binom{n}{2}} = \frac{4}{n}$  części jego oryginalnej liczebności.

**Dowód.** Niech drzewa  $T_s^*$  i  $T_{s'}^*$  będą zdefiniowane tak jak w lemacie. Dodatkowo niech  $T^k$  będzie optymalnym rozwiązaniem dla problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL, takim że zawiera ono największą liczbę krawędzi ze zbioru  $E^*$  spośród pozostałych rozwiązań. Jeśli liczba takich krawędzi w drzewie  $T^k$  równa się jego mocy ( $|T^k| = n - 1$ ) to dowód możemy zakończyć, gdyż  $\forall e \in T^k \ (e \in T^k \implies e \in E^*)$ , więc  $T^k \subseteq E^*$ .

Założymy zatem, że tak nie jest, czyli że istnieje taka krawędź  $e \in T^k$ , która nie należy do zbioru  $E^*$  —  $e \in T^k \setminus E^*$ . Aby  $T^k \subseteq E^*$ , usuńmy z niego tą krawędź. W wyniku usunięcia krawędzi  $e$  z drzewa  $T^k$  otrzymamy podział na dwa zbiory krawędzi będące drzewami:  $S$  i  $\bar{S}$  oraz zbiór  $\mathcal{Q}(T^k, e)$  (zbiór zdefiniowaliśmy w 2.3 i oznaczał on zbiór wszystkich takich krawędzi  $e_{ij}$ , gdzie  $v_i \in S$  oraz  $v_j \in \bar{S}$ ). Skorzystajmy teraz z własności optymalnego cięcia (2.2.1), które mówi, że drzewo jest minimalnym drzewem rozpinającym  $T$  wtedy i tylko wtedy, gdy należące do niego krawędzie  $e'$  mają najmniejszy koszt spośród wszystkich innych krawędzi, które należą do zbiorów  $\mathcal{Q}(T, e')$  (dla każdej krawędzi  $e' \in T$ ). Wybierzmy zatem ze zbioru  $\mathcal{Q}(T^k, e)$  krawędź  $e^*$  o najmniejszym koszcie w tym zbiorze. Bezpośrednio z powyższej własności wiemy, że wybrana krawędź należy do minimalnego drzewa rozpinającego  $T_{s'}^*$ . W związku z tym, że usuwana z drzewa  $T^k$  krawędź  $e \notin E^*$ , z definicji  $e \notin T_{s'}^* \wedge e \notin T_s^*$ . Z faktu zaś, że  $e \notin T_{s'}^*$  otrzymujemy natychmiast prosty wniosek, że  $e \neq e^*$  ( $e \notin T_{s'}^*$ ,  $e^* \in T_{s'}^*$ ). W związku z tym nic nie stoi na przeszkodzie aby z drzewa  $T^k$  stworzyć nowe drzewo  $T^{k'}$ , różne od  $T^k$ , poprzez usunięcie krawędzi  $e \in T^k \setminus E^*$  a dodanie do niego  $e^* \in T_{s'}^* \in E^*$ . Zauważmy, że z własności  $e^*$  wynika, że suma kosztów krawędzi w drzewie  $T^{k'}$  jest co najwyżej równe tej samej wartości dla drzewa  $T^k$ . Jako że drzewo  $T^k$  było rozwiązaniem optymalnym, nie istnieje drzewo o mniejszej sumie kosztów krawędzi do niego należących. Stąd prosty wniosek, że suma wag łuków w drzewie  $T^{k'}$  jest taka sama jak w  $T^k$ . Dodatkowo oczywistym jest, że otrzymane drzewo nadal jest rozwiązaniem dopuszczalnym dla problemu IMST. Z założenia optymalności  $T^k$  i definicji problemu INCREMENTAL mamy, że  $f(T^k, T_s^*) = k$ . Usuwana z drzewa  $T^k$  krawędź  $e \in T^k \setminus E^*$  z założenia nie należy do zbioru  $T_s^*$ , więc  $f(T^k \setminus \{e\}, T_s^*) = k - 1 < k$  (zatem bez względu na dodawaną krawędź  $e^*$ , drzewo  $T^{k'}$  jest rozwiązaniem dopuszczalnym). Zatem, skoro nowe drzewo jest również rozwiązaniem dopuszczalnym i zarazem optymalnym, a przy okazji zawiera o jedną krawędź wspólną ze zbiorem  $E^*$  niż drzewo  $T^k$ , otrzymaliśmy sprzeczność z założeniem, że  $T^k$  zawiera największą liczbę takich krawędzi spośród wszystkich optymalnych rozwiązań, co kończy dowód. ♦

Tym samym od razu możemy zredukować wyprowadzone przez nas równanie 5.1, zauważając, że zbiór  $E \setminus T_s^* = (T_s^* \cup T_{s'}^*) \setminus T_s^* = T_{s'}^*$ :

$$\min \left\{ \sum_{e_i \in E \setminus T_s^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} (c_i - \lambda) \cdot x_i \right\} = \min \left\{ \sum_{e_i \in T_{s'}^*} c_i \cdot x_i + \sum_{e_i \in T_s^*} (c_i - \lambda) \cdot x_i \right\}. \quad (5.3)$$

Pokażemy teraz jak dla tak zredukowanego zbioru krawędzi (z  $E$  do  $E^*$ ) stworzyć sztuczny scenariusz, w którym wszystkie koszty krawędzi będą unikalne i jednocześnie zachowywały oryginalną relację w stosunku do pozostałych kosztów.



**Lemat 5.2.2** [6] Dla danych kosztów krawędzi  $c_{e_i}$  w grafie  $G = (V, E)$ , dla każdego  $i \in \{1, \dots, |E|\}$  dla zmodyfikowanych kosztów zdefiniowanych jako  $c'_{e_i} = c_{e_i} + \phi(i)$ , gdzie  $\phi(i) = \frac{m \cdot i^2 + i}{(m+1)^3}$ , zachodzi:

$$c_{e_i} < c'_{e_i} < c_{e_i} + 1, \quad (5.4)$$

gdzie  $m = |E|$ .

Dla tak zdefiniowanych kosztów dodatkowo pokażemy, że dla dowolnych  $i$  oraz  $j$ , takich że  $i \neq j$  zachodzi dodatkowa własność:

$$\forall e_i, e_j \in E \left( i \neq j \implies c'_{e_i} \neq c'_{e_j} \right). \quad (5.5)$$

Dzięki tym dwóm własnościom będziemy mieli pewność, że po nałożeniu na wszystkie krawędzie w grafie takiego wzoru, każda krawędź będzie miała unikalny koszt, zaś po uporządkowaniu krawędzi względem ich kosztów oryginalnych i zaburzonych, otrzymamy wa identyczne ciągi (w wyniku zaburzenia kosztów, koszt żadnej z krawędzi nie przewyższy ani nie zostanie przewyższony przez koszt innej, której oryginalna waga przewyższała lub była przewyższana przez koszt pierwszej z nich). Taka modyfikacja rozwiąże obydwa problemy, o których wspomnieliśmy przy omawianiu rysunków od 5.1c do 5.1f. Oczywiście takie podejście niesie za sobą pewne ograniczenia — oryginalne koszty krawędzi muszą być całkowitoliczbowe, gdyż w przeciwnym przypadku własność 5.4 nie będzie dawała nam żadnej gwarancji, że dla zmodyfikowanych kosztów będziemy otrzymywać te same rozwiązania problemu drzewa rozpinającego co dla kosztów oryginalnych. Jeżeli zależy nam na zachowaniu zmienoprzecinkowych wag na krawędziach — możemy albo dodatkowo zmodyfikować koszty początkowe poprzez pomnożenie ich przed odpowiednią potęgą liczby 10, bądź też wprowadzić odgórny porządek na krawędziach, co wiązać się dodatkowo będzie z koniecznością modyfikacji każdej części naszych algorytmów, w których stają one przed wyborem jednej z dwóch lub większej liczby krawędzi o tym samym koszcie.

**Dowód.** Dowód własności 5.4 sprowadzimy do problemu pokazania, że dla dowolnego parametru, wartość  $\phi(i)$  zawiera się w przedziale otwartym  $(0, 1)$ . Niech zatem  $\phi(i) = \frac{m \cdot i^2 + i}{(m+1)^3}$ , gdzie  $m = |E|$ .

$$\begin{aligned} c_{e_i} &< c'_{e_i} < c_{e_i} + 1, \\ 0 &< (c_{e_i} + \phi(i)) - c_{e_i} < 1, \\ 0 &< \frac{m \cdot i^2 + i}{(m+1)^3} < 1, \\ 0 &< m \cdot i^2 + i < (m+1)^3, \\ 0 &< m \cdot i^2 + i \leq m^3 + m < m^3 + 3 \cdot m^2 + 3 \cdot m + 1 = (m+1)^3. \end{aligned} \quad (5.6)$$

W ostatniej nierówności skorzystaliśmy z faktu, że  $i$  oznacza numer porządkowy krawędzi w grafie,  $i \in \{1, \dots, m\}$ . Pokażemy teraz dodatkową własność 5.5 dla dowolnego  $i \neq j$ :

$$\begin{aligned} c'_{e_i} &\neq c'_{e_j}, \\ c_{e_i} + \phi(i) &\neq c_{e_j} + \phi(j), \\ c_{e_i} + \frac{m \cdot i^2 + i}{(m+1)^3} &\neq c_{e_j} + \frac{m \cdot j^2 + j}{(m+1)^3}. \end{aligned} \quad (5.7)$$

Jeżeli  $c_{e_i} = c_{e_j}$ , 5.7 w oczywisty sposób jest spełnione. W przeciwnym przypadku mamy spełniony warunek  $c_{e_i} + 1 \leq c_{e_j}$  (zakładając bez straty ogólności, że uporządkowaliśmy koszty rosnąco według numeru porządkowego krawędzi a  $i < j$  oraz, że kosztami są liczby całkowite) i stosując 5.6 otrzymamy:

$$c_{e_i} < c_{e_i} + \frac{m \cdot i^2 + i}{(m+1)^3} < c_{e_i} + 1 \leq c_{e_j} < c_{e_j} + \frac{m \cdot j^2 + j}{(m+1)^3}.$$

Pokazaliśmy zatem poprawność jednego ze sposobów na rozwiązywanie problemu z nieunikatowymi kosztami krawędzi w grafie, a co za tym idzie — zapewnić jednoznaczność rozwiązań zwracanych przez algorytm



Prima (bądź inny rozwiązujący zagadnienie minimalnego drzewa rozpinającego) dla dowolnego scenariusza i dowolnej wartości parametru  $\lambda$ . Przyglądając się formule 5.1, doszliśmy też do wniosku, że wraz ze stopniowym zwiększeniem wartości tego parametru otrzymujemy kolejne rozwiązania, którym coraz bliżej do spełnienia warunków, które określiliśmy w poprzednim rozdziale (4.33, 4.34). Pokażemy teraz formalnie, że takie postępowanie istotnie doprowadzi nas do rozwiązania.

**Twierdzenie 5.2.1** [20, 589] *Drzewo rozpinające  $T^*$  jest unikalnym minimalnym drzewem rozpinającym dla problemu INCREMENTAL wtedy i tylko wtedy, gdy spełnia własności 5.4, 5.5 oraz istnieje parametr  $\lambda \geq 0$ , dla którego  $T^*$  jest optymalne oraz:*

$$f(T^*, T_s^*) \leq k \wedge \lambda = 0 \quad \text{lub} \quad (5.8)$$

$$f(T^*, T_s^*) = k \wedge \lambda \neq 0. \quad (5.9)$$

**Dowód.** Unikalność rozwiązania  $T^*$  bezpośrednio wynika z własności 5.5, zaś pierwsza część dowodu (jeśli istnieje  $\lambda \geq 0$ , dla której  $T^*$  jest optymalne i spełnione są warunki 5.8 i 5.9, wtedy  $T^*$  jest unikalnym minimalnym drzewem rozpinającym dla problemu IMST) jest w zasadzie powtórzeniem twierdzenia o różnicach dopełniających, którego słuszność pokazaliśmy w poprzednim rozdziale (4.3). Pozostaje nam zatem do udowodnienia druga część dowodu (tylko wtedy, gdy istnieje  $\lambda \geq 0$ , dla której  $T^*$  jest optymalne i spełnione są warunki 5.8 i 5.9,  $T^*$  jest unikalnym minimalnym drzewem rozpinającym dla problemu IMST). Oczywiście, gdy  $\lambda = 0$ , drzewo  $T^* = T_s^*$  jest minimalnym drzewem rozpinającym. Co więcej, jeżeli spełnia nierówność  $f(T^*, T_s^*) \leq k$ , jest także optymalnym rozwiązaniem dla problemu IMST. Założymy zatem, że tak nie jest i  $(T^*, T_s^*) = k' > k$ . Co powinno być już dla nas oczywiste, jeżeli zaczniemy zwiększać wartość parametru  $\lambda$ , równolegle do niej będziemy zmniejszać koszty wszystkich krawędzi  $e \in T_s^*$ . W pewnym momencie parametr  $\lambda$  osiągnie taką wartość, że koszt pewnej pojedynczej krawędzi  $e_0 \in T_s^*$  zrówna się z kosztem krawędzi  $e_0^*$  należącej do aktualnego rozwiązania. Niech wartość tą reprezentuje wyrażenie  $\lambda_1$ , zaś drzewo o kosztach krawędzi zależnych od niego —  $T(\lambda_1)$ . Skoro  $c_{e_0} - \lambda_1 = c_{e_0^*}$ , to zamieniając ze sobą krawędzie miejscami ( $e_0 \in T_s^* \setminus T(\lambda_1)$  dołączając do rozwiązania, zaś  $e_0^* \in T(\lambda_1) \setminus T_s^*$  z niego usuwając), otrzymamy nowy zbiór krawędzi  $E'(\lambda_1)$ <sup>1</sup>, który będzie spełniał równość  $f(E'(\lambda_1), T_s^*) = k' - 1$ , zaś suma wszystkich krawędzi w nowym zbiorze będzie równa tej sumie w  $T(\lambda_1)$  (z punktu widzenia algorytmu rozwiązyującego problem MST nic więc się nie zmieni). Dodatkowo warto zauważyc, że ze względu na własności 5.4 i 5.5, które założyliśmy że spełniają wszystkie krawędzie w grafie<sup>2</sup>, taka para krawędzi  $(e_0, e_0^*)$  jest unikalna, więc w każdym kroku będziemy wymieniać co dokładnie jedną krawędź pomiędzy zbiorami. Powtarzając więc powyżej opisane kroki, podnosząc sukcesywnie wartość parametru  $\lambda$ , otrzymamy zbiór wartości  $\lambda_1, \lambda_2, \dots, \lambda_{k'-k}$  oraz analogicznie ciąg zbiorów  $E'(\lambda_1), E'(\lambda_2), \dots, E'(\lambda_{k'-k})$ , gdzie każdy z nich spełnia równość  $f(E'(\lambda_i), T_s^*) = k' - i$ . To oznacza, że w którymś momencie (dokładnie dla  $\lambda_{k'-k}$ ) otrzymamy  $f(E'(\lambda_{k'-k}), T_s^*) = k' - (k' - k) = k$  (widzimy, że funkcja  $f : X \times X \rightarrow \mathbb{N}$  jest monotoniczna, malejąca), co oznaczać będzie, że spełniliśmy jeden z dwóch opcjonalnych warunków optymalności rozwiązania (5.9). ◆

Mamy zatem pewność, że nasza strategia stopniowego zwiększenia wartości parametru  $\lambda$  i równoległego obniżania kosztów krawędzi, należących do oryginalnego rozwiązania  $T_s^*$ , doprowadzi nas do znalezienia optymalnego rozwiązania problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL z parametrem  $k$ . Pozostaje nam zatem już tylko kwestia odpowiedniego dobierania wartości wspomnianego parametru, jako że do tej pory milcząco zakładaliśmy, że zawsze jesteśmy w stanie znaleźć odpowiednie jego wartości (dla których każde kolejne rozwiązanie będzie zawierało malejącą liczbę krawędzi  $e \notin T_s^*$ , aż do momentu, gdy ich liczba wyniesie dokładnie  $k$ ). Pierwszym naiwnym pomysłem jest oczywiście zadanie pewnego interwału  $\Delta > 0$  i rozpoczęcie generowania kolejnych wartości parametru  $\lambda$ , rozpoczynając od  $\lambda_{\min}$ , kończąc na pewnym  $\lambda_{\max}$  (gdzie  $\lambda_{\min}$  może równać się 0 — wtedy pierwsze wygenerowane drzewo  $T(\lambda_{\min})$  odpowiadając będzie oczywiście drzewu  $T_s^*$  — za  $\lambda_{\max}$  możemy przyjąć chociażby koszt krawędzi w grafie o największej wadze)<sup>3</sup>.

<sup>1</sup> Trzeba tutaj zaznaczyć, że operacja wymiany krawędzi między zbiorami nic nie wspomina o własnościach dodawanego łuku względem starego — nie jesteśmy zatem w stanie powiedzieć czy dodawana krawędź zapewni nam to, że nowy zbiór będzie nadal drzewem (np. należy do zbioru  $Q(T^*(\lambda_1), e_0)$ , gdzie  $e_0$  to krawędź usuwana, gdzie w takim przypadku na pewno otrzymalibyśmy nowe drzewo). Aby zapewnić sobie, że nowy zbiór będzie nadal drzewem, powinniśmy wartość parametru  $\lambda_1$  (i każdy następny) podnieść jeszcze wyżej o  $\epsilon$ , a następnie poprosić algorytm dla problemu MST o nowe rozwiązanie.

<sup>2</sup> Graf może zawierać już tylko łuki  $E = T_{s'}^* \cup T_s^*$ .

<sup>3</sup> Jeśli dla zadanych danych problem minimalnego drzewa rozpinającego w wersji INCREMENTAL ma rozwiązanie, najprawdopodobniej przerwiemy obliczenia poniżej wartości  $\lambda_{\max}$ , jednakże w przeciwnym przypadku musimy takie górne ograniczenie przyjąć.

Wtedy będziemy generować kolejne drzewa:  $T(\lambda_{\min}), T(\lambda_{\min} + \Delta), \dots, T(\lambda_{\min} + i \cdot \Delta) = T^k$ , gdzie po  $i$  iteracjach możemy natknąć się na pierwsze optymalne rozwiązanie  $T^k$  dla badanego problemu. Problemy, które natychmiastowo pojawiają się przy takim rozwiązyaniu są dwa. Nie potrafimy określić jakiego rzędu wielkości powinien być parametr  $\Delta$ , by przypadkiem nie „przeskoczyć” odpowiednich wartości  $\lambda$ . Z drugiej strony, zmniejszanie tego pierwszego bardzo szybko zwiększa nam liczbę potencjalnych minimalnych drzew rozpinających, które musielibyśmy policzyć, co z kolei przekłada się negatywnie na czas działania takiego algorytmu.

### 5.3 Binarne poszukiwanie rozwiązania

Pierwszą próbą udoskonalenia przedstawionego pomysłu jest skorzystanie z własności funkcji  $f$ , która zwraca liczbę krawędzi będących częścią jednego z drzew, będącego parametrem tej funkcji, zaś nie należących do drzewa przekazanego jako jej drugi parametr. W dowodzie twierdzenia 5.2.1 pokazaliśmy, że wspomniana funkcja jest funkcją monotoniczną — dla dwóch różnych parametrów  $\lambda_i, \lambda_j$ , gdzie  $i < j$  ( $\lambda_i < \lambda_j$ ) zachodzi  $f(T(\lambda_i), T_s^*) \geq f(T(\lambda_j), T_s^*)$ . W naszym przypadku  $f(T(\lambda_i), T_s^*) \geq f(T(\lambda_i + \Delta), T_s^*)$ , zaś w przypadku przytoczonych dowodów zakładaliśmy, że każdy kolejny parametr  $\lambda$  spełniał  $f(T(\lambda_i), T_s^*) < f(T(\lambda_j), T_s^*)$ . Fakt takiego zachowania się funkcji  $f$  prowadzi nas do pomysłu zastosowania dla badanego problemu wyszukiwania binarnego — mając ciąg parametrów  $\lambda_0 < \lambda_1 < \dots < \lambda_{i-1} < \lambda_i < \lambda_{i+1} < \dots < \lambda_{\max}$  (i odpowiadający im ciąg wartości funkcji  $f(T(\lambda_0), T_s^*) \geq f(T(\lambda_1), T_s^*) \geq \dots \geq f(T(\lambda_{i-1}), T_s^*) \geq f(T(\lambda_i), T_s^*) \geq f(T(\lambda_{i+1}), T_s^*) \geq \dots \geq f(T(\lambda_{\max}), T_s^*)$ ) łatwo zauważać, że:

- Jeśli  $f(T(\lambda_i), T_s^*) > k$ , wtedy dla każdego drzewa  $T(\lambda_j)$ , gdzie  $j < i$  ( $\lambda_j < \lambda_i$ ) zachodzić będzie  $f(T(\lambda_j), T_s^*) > k$ . Zatem z góry możemy powiedzieć, że minimalne drzewa rozpinające liczone dla wszystkich scenariuszy  $\mathcal{S} = \{\mathbf{s}'(\lambda_j) : j \leq i\}$  okażą się rozwiązaniami, które nie są nawet dopuszczalne.
- Jeśli  $f(T(\lambda_i), T_s^*) < k$ , wtedy dla każdego drzewa  $T(\lambda_j)$ , gdzie  $j > i$  ( $\lambda_j > \lambda_i$ ) zachodzić będzie  $f(T(\lambda_j), T_s^*) < k$ . Zatem ponownie możemy powiedzieć, że minimalne drzewa rozpinające liczone dla wszystkich scenariuszy  $\mathcal{S} = \{\mathbf{s}'(\lambda_j) : j \geq i\}$  będą rozwiązaniami dopuszczalnymi, lecz nie optymalnymi (ze względu na niespełnienie warunków twierdzenia 5.2.1).
- Jeśli  $f(T(\lambda_i), T_s^*) = k$ , to znaleźliśmy rozwiązanie optymalne.

Dzięki takiemu podejściu, natychmiastowo redukujemy liczbę wymaganych obliczeń do logarytmu z tej liczby (tak jak to pokazano w pseudokodzie 3). Nadal jednak nie mamy żadnej gwarancji, że dobrana wartość parametru pomocniczego  $\Delta$  zapewni nam prawidłowe działanie naszego algorytmu — może się okazać, że podążając ścieżką wytyconą przez algorytm 3 dojdziemy do takiego momentu, w którym w rozpatrywanym przedziale  $[\lambda_l, \lambda_l + \Delta, \dots, \lambda_h - \Delta, \lambda_h]$  nie ma takiej wartości parametru  $\lambda_i^*$ , dla której otrzymalibyśmy rozwiązanie optymalne (takiego, że  $f(T(\lambda_i^*), T_s^*) = k$ ), chociaż zgodnie z algorytmem są spełnione warunki:  $f(T(\lambda_l), T_s^*) \geq k$  oraz  $f(T(\lambda_h), T_s^*) \leq k$ . Częściowym rozwiązaniem tego problemu jest pozwolenie algorytmowi w takiej sytuacji na porzucenie dyskretnego zbioru parametrów i samodzielne rozpoczęcie generowania nowych, będących średnimi arytmetycznymi wartości skrajnych kolejnych przedziałów. Takie rozwiązanie gwarantuje nam zatrzymanie się algorytmu w chwili znalezienia optymalnego rozwiązania  $T(\lambda^*)$ , lecz w żaden sposób nie określa liczby kroków, jakie wykona nasz algorytm zanim na to rozwiązanie „wpadnie”.

Aby rozwiązać i ten problem, przedstawimy dodatkowy lemat będący uzupełnieniem poprzednich dwóch (5.4 i 5.5).

**Lemat 5.3.1** [6] Dla danych kosztów krawędzi  $c_{e_i}$  w grafie  $G = (V, E)$ , dla każdego  $i \in \{1, \dots, |E|\}$  dla zmodyfikowanych kosztów zdefiniowanych jako  $c'_{e_i} = c_{e_i} + \phi(i)$ , gdzie  $\phi(i) = \frac{m \cdot i^2 + i}{(m+1)^3}$ , zachodzi:

$$c_{e_i} < c'_{e_i} < c_{e_i} + 1 \text{ oraz} \quad (5.10)$$

$$\forall (i, j, k, l : i \neq j \wedge i \neq k) \quad c'_{e_i} - c'_{e_j} \neq c'_{e_k} - c'_{e_l} \quad (5.11)$$

gdzie  $m = |E|$ .





Nasz nowo zdefiniowany zbiór  $\Lambda$  oczywiście spełnia założenia własności 5.11 — zbiór  $\{e_i : d(i, j) \in \Lambda\}$  z definicji jest rozłączny ze zbiorem  $\{e_j^* : d(i, j) \in \Lambda\}$  ( $e_i \in T_s^* \setminus T_{s'}^* \wedge e_j^* \in T_{s'}^* \setminus T_s^*$ ), zatem dla wyrażenia z lematu:  $c'_{e_i} - c'_{e_j} \neq c'_{e_k} - c'_{e_l}$ , spełnione są własności:  $i \neq j$  oraz  $k \neq l$ . Zarówno na samym początku omawiania podejście bazującego na wyszukiwaniu binarnym do wskazanego problemu, jak i w 3 dla argumentu wejściowego  $\Lambda$  wskazaliśmy potrzebę uporządkowania dyskretnego zbioru parametrów  $\lambda$  w porządku rosnącym tak, aby wykorzystanie algorytmu wyszukiwania binarnego było w ogóle możliwe. Aby spełnić i tę własność, musimy przyznać się sposobi generowania kolejnych wartości  $d(i, j) = c'_{e_i} - c'_{e_j}$ . Przypomnijmy, że  $\Lambda = \{c'_{e_i} - c'_{e_j} : e_i \in T_s^* \setminus T_{s'}^* \wedge e_j \in T_{s'}^* \setminus T_s^*\}$ . Rozpatrzmy następujący sposób konstrukcji tego zbioru:

- dla wszystkich krawędzi  $e_i \in T_s^* \setminus T_{s'}^*$  stworzymy zbiór  $E_{\geq}$ , do którego należeć będą wszystkie wspomniane krawędzie, a dodatkowo będą one posortowane względem ich kosztów w kolejności od największego do najmniejszego.
- Analogicznie, dla wszystkich krawędzi  $e_j^* \in T_{s'}^* \setminus T_s^*$  stworzymy zbiór  $E_{\leq}$ , do którego będą należeć wymienione krawędzie, a dodatkowo posortujemy je rosnąco względem ich kosztów tak, że krawędź o największym koszcie będzie ostatnim elementem zbioru  $E_{\leq}$ .
- Stworzymy, równoważny ze zbiorem  $\Lambda$ , zbiór  $\Lambda' = \{c'_{e_i} - c'_{e_j} : e_i \in E_{\geq} \wedge e_j \in E_{\leq}\}$ .

Zauważmy, że elementy tak zdefiniowanego zbioru spełniają poniższe nierówności (innymi słowy funkcja zwracająca ich wartości —  $d : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$  — jest monotoniczna i rosnąca względem któregokolwiek z parametrów):

$$\forall i, i' \in \{1, \dots, |E^*|\} \quad d(i, j) < d(i', j) \Leftrightarrow i < i' \text{ oraz} \quad (5.12)$$

$$\forall j, j' \in \{1, \dots, |E^*|\} \quad d(i, j) < d(i, j') \Leftrightarrow j < j'. \quad (5.13)$$

## 5.4 Pełny algorytm przeszukiwania binarnego

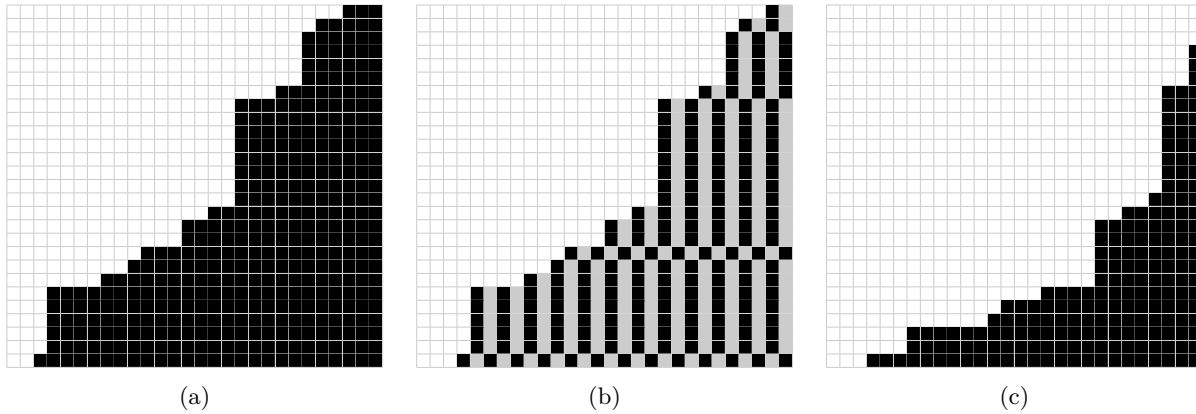
Pełen pseudokod naszego algorytmu rozwiązującego problem minimalnego drzewa rozpinającego w wersji INCREMENTAL, bazujący na algorytmie wyszukiwania binarnego prezentujemy poniżej (zobacz Pseudokod 4). Większą jego część zdążyliśmy już omówić, gdyż wszystkie do tej pory poznane przez nas problemy oraz sposoby na ich rozwiązanie zostały w nim wykorzystane — zamiast więc jeszcze raz je przytaczać, w tej części skupimy się na implementacyjnych szczegółach podanego pseudokodu. Na początku warto zwrócić uwagę na schemat omawianego algorytmu, który swoją konstrukcją mocno przypomina poprzednio przedstawiany algorytm. Bardzo łatwo zauważać, że tak samo jak w przypadku IMST-BINARY-SEARCH  $(G^*, T_s^*, s', \Lambda, k)$ , także tutaj główną osią algorytmu jest podejmowanie decyzji dotyczących zawężania dolnego i górnego parametru ograniczającego przestrzeń rozwiązań, które należy sprawdzić w celu wyłonienia tego optymalnego (linijki 20–25 dla INCREMENTAL-MST  $(G^*, T_s^*, s', k, L, U)$  odpowiadają liniom 9–15 w IMST-BINARY-SEARCH  $(G^*, T_s^*, s', \Lambda, k)$ ). Jedyną różnicą w tej części kodu jest sposób wybierania wartości nowego górnego bądź dolnego ograniczenia dla następnej iteracji oraz sposób jej wywołania (choć nic nie stoi na przeszkodzie by, na wzór drugiego z wymienionych algorytmów, przekształcić ten pierwszy, usuwając wywołania rekurencyjne, tak jak to pokazano na pseudokodzie 5). Sam zaś algorytm ma głównie na celu generowanie kolejnych zbiorów parametrów  $\lambda$ , które są dopuszczalne w myśl aktualnie wybranych ograniczeń (takimi wartościami parametru  $\lambda$  będą zatem wszystkie wartości  $d(i, j)$  spełniające  $L \leq d(i, j) \leq U$ ). Same zaś ograniczenia  $L$  i  $U$  będziemy oczywiście chcieli tak dobierać, aby z każdą następną iteracją zbiór  $F = \{(i, j) : L \leq d(i, j) \leq U\}$  był coraz mniejszy — naszym celem bowiem jest doprowadzenie do sytuacji, w której liczba elementów tego zbioru jest na tyle niewielka, że wykonanie procedury IMST-BINARY-SEARCH  $(G^*, T_s^*, s', \Lambda, k)$  nie będzie nas wiele kosztować (linie 11–13).

Na początek jednak przyjrzyjmy się fragmentowi, który odpowiada za zliczanie liczby elementów  $(i, j)$ , które spełniają zadane ograniczenia (2–9). Dla każdej pierwszej krawędzi z pary łuków  $(e_i, e_j^*)$ , których różnica kosztów jest wyliczana wyrażeniem  $d(i, j)$ , chcemy znaleźć parę indeksów  $(j_L, j_U)$ , taką że zachodzi warunek  $L \leq d(i, j_L) \leq \dots \leq d(i, j_U) \leq U$ . Nasze poszukiwania tych parametrów możemy zacząć w



dowolny sposób, jednak już w tym kroku napotykamy pierwszy problem — co jeśli dowolne z wyrażeń  $\min\{j : L \leq d(i, j)\}$  lub  $\max\{j : d(i, j) \leq U\}$  nie zwróci żadnego wyniku? Może bowiem się zdarzyć, że dla zadanego parametru  $i$ , tak zadana wartość (koszt krawędzi  $e_i \in T_s^* \setminus T_{s'}^*$ ) dla wyrażenia  $d(i, j)$  dla dowolnego  $j \in \{j : e_j^* \in T_{s'}^* \setminus T_s^*\}$  nie będą spełniać zadanych nierówności. Zwróćmy uwagę, że w takim przypadku, jeżeli byśmy rozpoczęli wyszukiwanie indeksu  $i$  od najniżej do najwyższej wartości dla  $\text{MinIndex}[i]$  (oczekujemy, że szukając najmniejszego indeksu spełniającego zadany warunek, najszybciej na niego natrafimy, rozpoczynając przeszukiwanie w takiej kolejności) i dla żadnego  $j \in \{1, \dots, m'\}$  ( $m' = |T_s^* \setminus T_{s'}^*| = |T_{s'}^* \setminus T_s^*|$ ) warunek  $L \leq d(i, j)$  nie byłby spełniony, aby zachować poprawność algorytmu, powinniśmy zwrócić wartość większą niż  $m'$ , tak aby w linii 6 zapewnić sobie niespełnienie znajdującego się w niej warunku (wartość  $\text{MaxIndex}[i]$  nigdy nie będzie większa od  $m'$ , o czym się zaraz przekonamy). Analogicznie do  $\text{MinIndex}[i]$ , wartość wyrażenia  $\text{MaxIndex}[i]$  możemy ustalić rozpoczynając wyszukiwanie indeksu  $j_U = \max\{j : d(i, j) \leq U\}$  od  $j = m'$  (znowu oczekujemy, że skoro szukamy maksymalnej jego wartości, rozsądnie będzie zacząć od największego indeksu, dla którego wyrażenie  $d(i, j)$  ma sens). Tutaj także musimy obsłużyć przypadek, w którym  $\forall j \in \{m', \dots, 1\} d(i, j) > U$  — naturalnym wyborem indeksu krawędzi  $j$  takiej sytuacji jest 0 (jako że od samego początku indeksujemy krawędzie począwszy od jedynki).

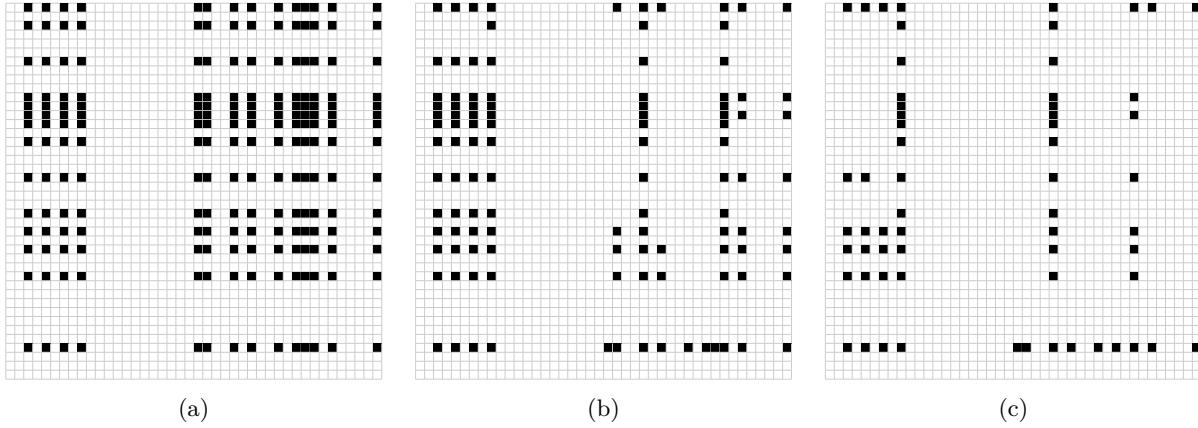
Tak poprawnie wyliczona wielkość zbioru, w przypadku jej liczebności przekraczającej zadaną granicę (tutaj jest to  $12 \cdot n$ ), powinna zostać w następnym kroku zmniejszona, co dzieje się w liniach od 15 do 19 pseudokodu 4, w których to „przesiewamy” dopuszczalny zbiór parametrów na podstawie wybranej wartości  $K$ . Doskonałą ilustrację schematu działania znajdującego się w tym fragmencie kodu mechanizmu, przedstawiają rysunki od 5.2a do 5.2c, gdzie w wyniku tylko dwóch iteracji, zredukowaliśmy przestrzeń przeszukiwania o ponad 75%. Na uwagę zasługuje też fakt, że w przedstawionym przypadku ani razu nie ograniczyliśmy wartości  $d(i, j)$  z góry — widzimy to po specyficzny kształcie prezentowanym na wszystkich trzech prezentowanych rysunkach.



Rysunek 5.2: Tablice przedstawiające przykładowy, rzeczywisty rozkład dopuszczalnych wartości funkcji  $d(i, j)$  dla par  $(i, j)$  i grafu  $G$  z 15 wierzchołkami o gęstości na poziomie 40% grafu pełnego (z  $\binom{15}{2}$  krawędziami). Z 42 krawędzi grafu, wygenerowane drzewa rozpinające dla dwóch scenariuszy  $s$  i  $s'$  zostały wygenerowane minimalne drzewa rozpinające  $T_s^*$  oraz  $T_{s'}^*$ , których część wspólną stanowi 28 krawędzi grafu  $G$ . Kolumny odpowiadają pierwszemu indeksowi z pary  $(i, j)$ . (a) Zbór  $\Lambda = \{c'_{e_i} - c'_{e_j} : e_i \in T_s^* \setminus T_{s'}^* \wedge e_j \in T_{s'}^* \setminus T_s^*\}$  (cała przestrzeń) oraz dopuszczalne pary  $F = \{(i, j) : L' \leq d(i, j) \leq U\}$  (zaznaczone kolorem czarnym) po pierwszej iteracji algorytmu INCREMENTAL-MST( $G^*, T_s^*, s', k, L, U$ ) (dla ograniczeń  $L'$  i  $U$ , gdzie  $L < L' \leq U$ ). Wstępna liczba dopuszczalnych par została zredukowana z 1764 do 784 (dzięki ograniczeniu się do par wartości ze zbioru  $\Lambda$ ), a następnie dalej, do liczby 380 (po zastosowaniu nowego dolnego ograniczenia). (b) „Przesiany” zbiór dopuszczalnych rozwiązań, na podstawie którego wyliczona zostanie mediana będąca nową wartością  $\lambda$  (która w następnej iteracji będzie albo nowym górnym, albo nowym dolnym ograniczeniem na dopuszczalność par  $(i, j)$ ). (c) Pary  $F = \{(i, j) : L'' \leq d(i, j) \leq U\}$  dla nowego dolnego ograniczenia. Liczba par została zredukowana z 380 do 189.

Aby uzyskać jeszcze lepszy obraz sytuacji, do której doprowadził nas wykorzystywany algorytm, spójrzmy na rysunki 5.3 — przedstawiono na nich wszystkie możliwe wartości funkcji  $d(i, j)$ , dla każdego  $i, j \in$

$\{1, \dots, |E|\}$  (nie tylko dla ograniczonego zbioru  $\{(i, j) : e_i \in T_s^* \setminus T_{s'}^* \wedge e_j \in T_{s'}^* \setminus T_s^*\}$ ). Pierwszy rysunek (5.3a) rozpoczyna się tam, gdzie skończyliśmy poprzednio, analizując ilustrację 5.2c — na drugiej iteracji algorytmu INCREMENTAL-MST ( $G^*, T_s^*, s', k, L, U$ ), dla grafu z 15 wierzchołkami i 42 krawędziami, których liczebność wynika z żądanego gęstości generowanego losowo grafu (40%). Należy podkreślić, że obydwa grafy, których tyczą się ilustracje 5.2a oraz 5.3a, choć posiadają te same parametry, są różne względem siebie (procedura wykorzystana do ich generowania jest losowa). Rozmiar danych jednak jest na tyle mały, że liczby dopuszczalnych par  $(i, j)$  po drugiej iteracji algorytmu w obu przypadkach są zbliżone do siebie (189 w pierwszym przypadku, 196 w drugim).



Rysunek 5.3: Macierz, której komórki przedstawiają wartości wyrażeń  $c_i - c_j$  dla dowolnej kombinacji krawędzi  $e_i$  oraz  $e_j$  w grafie  $G = (V, E, q)$ , gdzie  $|V| = 15$ ,  $|E| = 42$ ,  $q$  jest gęstością grafu ( $q = 0.4$ ). Zaznaczone na czarno elementy należą do zbioru  $F \{(i, j) : L \leq c_i - c_j \leq U\}$ , gdzie wartości  $L$  i  $U$  odpowiadają aktualnym w drugiej iteracji algorytmu INCREMENTAL-MST ( $G^*, T_s^*, s', k, L, U$ ) ograniczeniom. (a) Dopuszczalne pary w liczbie 196. (b) Zredukowany zbiór dopuszczalny par  $(i, j)$ , dla których wartość  $c_i - c_j$  jest nie mniejsza niż dolne i nie większa niż górne ograniczenie, wybrane przez z algorytmem w kolejnej iteracji. Liczba zaznaczonych elementów zmalała do 110. (c) Pomniejszony w wyniku kolejnej iteracji algorytmu INCREMENTAL-MST ( $G^*, T_s^*, s', k, L, U$ ) zbiór, który liczy już tylko 62 elementy.

To co od razu możemy dostrzec, patrząc zwłaszcza na pierwszy z rysunków (5.3a), to wyraźna regularność zaznaczonych elementów, spowodowana wykorzystaniem wcześniej udowadnianych własności (5.4, 5.5, 5.3.1), które pozwoliły nam na skonstruowanie zbioru  $\Lambda$  — na wspomnianym rysunku widać wyraźnie, że dla niektórych par krawędzi (dla wierszy oraz kolumn symbolizujących krawędzie będące wspólną częścią drzew  $T_s^*$  oraz  $T_{s'}^*$ ) nie istnieje żadna komórka, która byłaby wliczana w zbiór par dopuszczalnych rozwiązań.

#### 5.4.1 Lepiej, szybciej

Tak jak to zostało podkreślone na samym początku poprzedniego rozdziału, niewielkim nakładem pracy jesteśmy w stanie przekształcić pseudokod 4 do postaci nie korzystającej z rekurencji, tak aby do złudzenia swoją konstrukcją przypominał tradycyjny algorytm przeszukiwania binarnego (3). Jedną z niewątpliwych zalet takiego rozwiązania jest nie tylko poprawiona czytelność prezentowanego kodu, co niejednokrotnie dużo szybszy czas jego działania, idącego w parze z oszczędnością pamięci, której algorytm wymaga coraz więcej za każdym wywołaniem rekurencyjnym. W tej części skupimy się na rozwiązaniach jakie możemy zastosować w celu poprawienia ogólnej wydajności wcześniejszej omówionego algorytmu, krok po kroku go analizując.

Do tej pory nic nie wspominaliśmy o złożoności prezentowanego rozwiązania — jak pamiętamy, podstawową koncepcją algorytmu jest zmniejszanie liczby elementów  $(i, j)$  o zadanych własnościach do momentu, w którym jest ich na tyle niewielu, że decydujemy się na binarne wyszukanie rozwiązania przy wykorzystaniu IMST-BINARY-SEARCH ( $G^*, T_s^*, s', F, k$ ). Owocuje to natychmiastowo złożonością na poziomie  $O(I + K + S + \log(n) \cdot B)$ , gdzie:

- $I$  oznacza czas potrzebny na znalezienie rozwiązań  $T_s^*$  i  $T_{s'}^*$  dla problemu minimalnego drzewa rozpinającego dla początkowego scenariusza  $s$  oraz nowego —  $s'$ , i jest równoznaczne z czasem wykonania się

**Pseudokod 4:** INCREMENTAL-MST ( $G^*, T_s^*, s', k, L, U$ )

**Wejście:**  $G^* = (V, E^*)$  — graf ze zbiorem krawędzi  $T_{s'}^* \cup T_s^*$ ,  
 $T_s^*$  — początkowe minimalne drzewo rozpinające dla scenariusza  $s$ ,  
 $s'$  — nowe koszty krawędzi grafu,  
 $k$  — parametr problemu IMST,  
 $L$  — dolne ograniczenie na wartość parametru  $\lambda^*$ ,  
 $U$  — górne ograniczenie na wartość parametru  $\lambda^*$ ,

**Wyjście:**  $T(\lambda^*)$  — optymalne rozwiązanie problemu IMST.

```

1 begin
2   for  $i \in \{1, \dots, m'\}$                                 //  $m' = |T_s^* \setminus T^*|$ 
3     do
4       MinIndex[i]  $\leftarrow \min \{j : L \leq d(i, j)\}$ 
5       MaxIndex[i]  $\leftarrow \max \{j : d(i, j) \leq U\}$ 
6       if  $MinIndex[i] \leq MaxIndex[i]$  then
7         Count[i]  $\leftarrow MaxIndex[i] - MinIndex[i] + 1$ 
8       else
9         Count[i]  $\leftarrow 0$ 
10      TotalCount  $\leftarrow \sum_{i=1}^{m'} Count[i]$ 
11      if  $TotalCount \leq 12 \cdot n$  then
12        F  $\leftarrow \{(i, j) : L \leq d(i, j) \leq U\}$ 
13        return IMST-BINNARY-SEARCH( $G^*, T_s^*, s', F, k$ )
14      else
15        K  $\leftarrow \lfloor \frac{TotalCount}{6 \cdot n} \rfloor$ 
16        for  $i \in \{1, \dots, m'\}$  do
17          H[i]  $\leftarrow \{(i, j) : L \leq d(i, j) \leq U \wedge j \leftarrow MinIndex[i] + r \cdot K \wedge r \in \mathbb{Z}^+\}$ 
18         $\mathcal{H} \leftarrow \bigcup_{i=1}^{m'} H[i]$ 
19         $\lambda \leftarrow \text{MEDIAN}(\mathcal{H})$ 
20        if  $f(T(\lambda), T_s^*) = k$  then
21          return  $T(\lambda)$ 
22        else if  $f(T(\lambda), T_s^*) > k$  then
23          return INCREMENTAL-MST( $G^*, T_s^*, s', k, \lambda, U$ )
24        else if  $f(T(\lambda), T_s^*) < k$  then
25          return INCREMENTAL-MST( $G^*, T_s^*, s', k, L, \lambda$ )

```

podstawowego algorytmu rozwiązyującego ten problem ( $Om(\alpha(m, n))$  [4]),

- $K$  czasu musimy przeznaczyć na skonstruowanie parametrów, których wartości do tej pory oznaczaliśmy poprzez  $d(i, j)$ ,
- $S$  oznacza czas potrzebny algorytmowi IMST-BINNARY-SEARCH ( $G^*, T_s^*, s', F, k$ ) na wstępne posortowanie otrzymanego zbioru  $F$  ( $O(n \cdot \log(n))$ ), zaś
- $B = I$ , gdyż wspomniany wcześniej algorytm w najgorszym przypadku będzie musiał obliczyć  $O(\log(n))$  minimalnych drzew rozpinających zanim zwróci rozwiązanie.

Podsumowując, otrzymujemy złożoność na poziomie  $O(m \cdot \alpha(m, n) + K + n \cdot \log(n) \cdot \alpha(m, n))$ . Wartym uświadomienia jest fakt, że przywoływana tutaj **odwrócona funkcja Ackermann'a** jest funkcją ekstremalnie wolno rosnącą, w związku z czym  $O(m \cdot \alpha(m, n) + K + n \cdot \log(n) \cdot \alpha(m, n)) \approx O(m + K + n \cdot \log(n))$  (dla dowolnie dużych, lecz jednocześnie sensownie małych w kontekście rozpatrywanego problemu, funkcja zwraca wartości na tyle małe, że w notacji dużego  $O$  możemy uznać je za stałe). To zaś wskazuje na istotność w jaki sposób wyliczane są parametry  $\lambda$  — możemy przedstawić dwa podejścia do tego problemu.



- Naiwne policzenie wszystkich możliwych wartości — nawet dla bardzo okrojonego zbioru jakim jest  $\{(x, y) : e_i \in T_s^* \setminus T_{s'}^* \wedge e_j \in T_{s'}^* \setminus T_s^*\}$ , chęć wyliczenia wszystkich możliwych parametrów natychmiast ograniczy naszą złożoność do poziomu  $\Omega(n^2)$ <sup>4</sup>.
- Wyliczanie wartości  $d(i, j)$  na bieżąco (tak jak to właśnie zakładamy w przedstawianych pseudokodach). Skutkuje to koniecznością wygenerowania dwóch zbiorów ( $E_0 = \{e \in T_s^* \setminus T_{s'}^*\}$  i  $E_1 = \{e \in T_{s'}^* \setminus T_s^*\}$ ) oraz ich posortowania, co wykonamy w czasie co najwyżej  $O(m + n \cdot \log(n))$  ( $O(m)$  zajmie nam wybór odpowiednich krawędzi,  $O(n \cdot \log(n))$  — posortowanie ich, pierwszego rosnąco, drugiego malejąco). Od tej pory, odwołanie się do funkcji  $d : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$  zwracać nam będzie pożądaną wartość w czasie stałym.

Zajmijmy się teraz sposobem w jaki zminimalizować możemy liczbę wywołań wspomnianej funkcji. Spoglądając na linijki od 1 do 9 w pseudokodzie 4 widzimy, że dla każdej wartości od 1 do  $m'$  wyliczamy dwie wartości na podstawie funkcji  $\min(\bullet)$  oraz  $\max(\bullet)$ . Powiedzieliśmy sobie, że dla każdego  $i \in \{1, \dots, m'\}$  najrozsądniej z naszej strony będzie rozpoczęć wyszukiwanie tej pierwszej od  $j = 1$ , zaś tej drugiej od  $j = m'$  — w najgorszym przypadku (gdy dla każdego  $i$ ,  $\min\{j : L \leq d(i, j)\} = m'$  a  $\max\{j : d(i, j) \leq U\} = 1$ ) daje nam to złożoność ma poziomie  $O(m' \cdot m')$  (pamiętamy, że  $m'$  jest liczbą krawędzi dowolnego ze zbiorów  $T_s^* \setminus T_{s'}^*$  lub  $T_{s'}^* \setminus T_s^*$ ,  $m' \leq n - 1$ ). Widzimy zatem, że taka strategia może zaowocować nawet złożonością na poziomie  $O(n^2)$ , gdzie podobny rezultat uzyskaliśmy stosując naiwne podejście.

Pokażemy teraz jak w prosty sposób, wykorzystując własności funkcji  $d : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$  (monotoniczność względem dowolnego z jej parametrów), możemy przyspieszyć ten proces, ograniczając go do co najwyżej  $m'$  wywołań powyższej funkcji. W 5.12 pokazaliśmy, że wartość funkcji  $d(\bullet, \bullet)$  rośnie wraz z dowolnym jej parametrem to znaczy zachodzą:

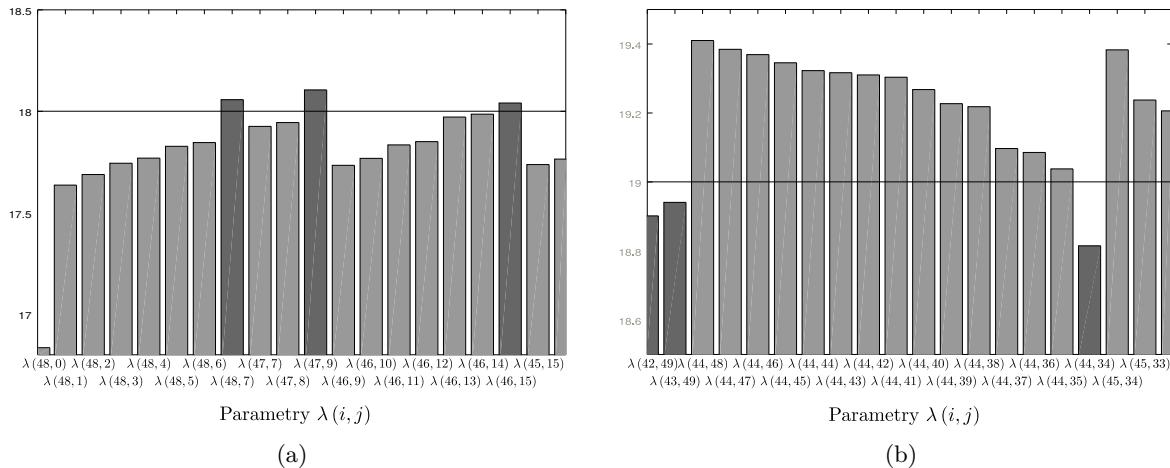
- $\forall i, j \ d(i, j) < d(i, j + 1)$ <sup>5</sup>,
- $\forall j. j \ d(i, j) < d(i + 1, j)$
- $\forall i, j \ d(i, j) < d(i + 1, j + 1)$ .

Przyjrzyjmy się teraz wyrażeniom  $\min\{j : L \leq d(i, j)\}$  oraz  $\max\{j : d(i, j) \leq U\}$ . Przypomnijmy jeszcze raz, że w celu znalezienia indeksu  $j$  o najmniejszej wartości, postanowiliśmy rozpoczęć jego poszukiwania od  $j = 1$ . Niech teraz  $i = m'$ . Niech  $j = j_1$  będzie takim indeksem, że  $j_1 = \min\{j : L \leq d(m', j)\}$ . Zgodnie z powyższymi własnościami zachodzi:  $d(m' - 1, j_1) < d(m', j_1)$  — mamy teraz prawo podejrzewać, że skoro  $j = j_1$  było najmniejszym indeksem, dla którego wyrażenie  $d(m', j)$  było większe niż  $L$ , to wartość  $d(m' - 1, j)$  może nie spełniać już tej własności. W takim przypadku, by znaleźć kolejny indeks, tym razem dla  $i = m' - 1$ , musimy wybrać taki parametr  $j$ , aby zwiększyć poprzednią wartość funkcji (by na nowo  $d(i, j)$  przewyższało wartością dolne ograniczenie  $L$ ). Z własności przedstawionych wyżej wiemy, że jedynym sposobem na dokonanie tego jest zwiększenie drugiego z parametrów ( $d(m' - 1, j_1) < d(m' - 1, j_1 + 1) < \dots$ ). Co także wiemy, w pewnym momencie dla pewnego  $j = j_2 > j_1$  zajdzie  $L \leq d(m' - 1, j)$ , tak więc indeks  $j_2$  jest tym najmniejszym, który dla kolejnej wartości indeksu  $i$  spełnia wszystkie założenia. Proces ten możemy kontynuować aż do momentu, w którym  $j = m' + 1$ , bądź policzyliśmy wartości  $\text{MinIndex}[i]$  dla wszystkich  $i \in \{m', \dots, 1\}$ . W tym pierwszym przypadku (linie 13–14 algorytmu 5) możemy zauważyć, że nie ma potrzeby kontynuować procesu dla wszystkich takich  $i'$ , których wartość jest mniejsza od ostatniego  $i$ , dla którego zachodziła nierówność  $L \leq d(i, m')$  — ponownie, opierając się na własnościach monotoniczności funkcji  $d(\bullet, \bullet)$  wiemy, że dla wszystkich pozostałych wartości  $i' < i$ , aby utrzymać wartość wyrażenia  $d(\bullet, \bullet)$  powyżej dolnego ograniczenia  $L$ , jesteśmy zmuszeni do sukcesywnego podwyższania drugiego z jej parametrów, ten zaś nie może być dalej zwiększany, gdyż osiągnął już swoją graniczą wartość, dla której  $d(\bullet, \bullet)$  ma dla nas sens. Może się oczywiście zdarzyć, że dla pewnego ciągu  $i - k, i - (k - 1), \dots, i$  będzie zachodzić  $L \leq d(i - k, j) < d(i - (k - 1), j) < \dots < d(i, j)$  — w takim przypadku po prostu wstrzymujemy się na pewien czas z podwyższaniem wartości parametru  $j$ .

Opisaną powyższą strategię doskonale ilustruje rysunek 5.4a, gdzie wyraźnie widzimy w których momencach wartość indeksu  $j$  dla wyrażenia  $d(i, j)$  została na tyle podwyższona, aby przekroczyła wartość dolnego

<sup>4</sup>Chcąc być dokładniejszym — do  $\Omega(|T_s^* \setminus T_{s'}^*|^2)$ , czyli w najgorszym przypadku właśnie do  $\Omega((|V| - 1)^2) = \Omega(n^2)$ .

<sup>5</sup>Na mocy lematu 5.3.1 wiemy także, że we wszystkich trzech przypadkach zachodzi ostra nierówność.



Rysunek 5.4: Graficzne zaprezentowanie wartości funkcji  $d : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$  dla wycinka algorytmu 5, obliczającego  $\min\{j : L \leq d(i, j)\}$  dla wybranego zakresu wartości indeksów. (a) Początek procesu obliczania kolejnych zmiennych  $\text{MinIndex}[i]$  dla  $i \in \{48, \dots, 45\}$  ( $m' = 49$ ), gdzie jasnym szarym kolorem oznaczono te wartości  $d(i, j)$ , dla których nie zachodził warunek  $L \leq d(i, j)$ . Miejsca oznaczone kolorem ciemniejszym stanowią punkty, w których dla danego  $i$  odnaleziono takie  $j$ , że  $j = \min\{j : L \leq d(i, j)\}$  ( $L = 18$ ). (b) Graficzna reprezentacja fragmentu algorytmu obliczającego wyrażenie  $\max\{j : d(i, j) \leq U\}$  (od  $i = 42$  do  $i = 45$ ). Ciemniejszym kolorem oznaczono wartości funkcji  $d(i, j)$ , która dla tak dobranych parametrów, spełnia powyższą nierówność ( $U = 19$ ).

ograniczenia  $L$ , co pociąga za sobą obniżenie pierwszego parametru i rozpoczęcie procesu od nowa. Analogicznie do schematu wyszukiwania wszystkich wartości  $\text{MinIndex}[i]$  zachowuje się proces obliczania wyrażeń  $\text{MaxIndex}[i]$  (przedstawiony na rysunku 5.4b) — opierając o to samo rozumowanie oraz te same własności co powyżej, zaczynając od  $i = 1$  oraz  $j = m'$ , możemy tak manipulować obydwoema wartościami, aby całkowity czas operacji nie przekroczył czasu liniowego, zależnego od  $m'$ , podobnie jak miało to miejsce wcześniej. W przypadku rozpoczęcia od  $j = m'$ , możemy poczynić niewielką obserwację, że gdy pętla 17-21 w algorytmie 5 zostanie przerwana zanim policzymy  $\text{MaxIndex}[i]$  dla wszystkich  $i \in \{1, \dots, m'\}$ , do pozostałych zmiennych możemy przypisać stałą 0, gdyż taką wartość przyjmie wtedy indeks  $j$ . Razem z następnymi liniijkami, fragment 5–26 stanowi eleganckie usprawnienie bloku 2–10 poprzedniego algorytmu, już znacznie go usprawniając.

Dalsze różnice pomiędzy algorytmami 4 a 5 mają charakter bardziej kosmetyczny (usunięcie wywołań rekurencyjnych, zasugerowanie sposobu konstrukcji zbiorów dopuszczalnych par  $F$ , przerzedzonego zbioru  $\mathcal{H}$ ) i nie wymagają poświęcania im większej uwagi.

## 5.5 Analiza poprawności

```

fsfdf
d
fdfd
fdf
d
gg
fg
fg
fsfdf
d
fdfd

```

---

**Pseudokod 5:** INCREMENTAL-MST' ( $G^*, T_s^*, \mathbf{s}', k, L, U$ )

---

```
1 begin
2   MaxIndex [0] ←  $m'$ 
3   MinIndex [ $m'$ ] ← 0
4   while  $L \leq U$  do
5     TotalCount ← 0
6      $i \leftarrow m'$ 
7      $j \leftarrow \text{MinIndex}[i]$ 
8     while  $j \leq m'$  do
9       while  $L > d(i, j) \wedge j \leq m'$  do
10         $j \leftarrow j + 1$ 
11        MinIndex [ $i$ ] ←  $j$ 
12         $i \leftarrow i - 1$ 
13      for  $i' \in \{i, \dots, 1\}$  do
14        MinIndex [ $i$ ] ←  $j$ 
15       $i \leftarrow 0$ 
16       $j \leftarrow \text{MaxIndex}[i]$ 
17      while  $j > 0$  do
18        while  $d(i, j) > U \wedge j > 0$  do
19           $j \leftarrow j - 1$ 
20          MaxIndex [ $i$ ] ←  $j$ 
21           $i \leftarrow i + 1$ 
22      for  $i' \in \{i, \dots, 1\}$  do
23        MaxIndex [ $i$ ] ← 0
24      for  $i \in \{1, \dots, m'\}$  do
25        if  $\text{MinIndex}[i] \leq \text{MaxIndex}[i]$  then
26          TotalCount ←  $\text{MaxIndex}[i] - \text{MinIndex}[i] + 1$ 
27      if  $\text{TotalCount} \leq 12 \cdot n$  then
28         $F \leftarrow \emptyset$ 
29        for  $i \in \{1, \dots, m'\}$  do
30          for  $j \in \{\text{MinIndex}[i], \dots, \text{MaxIndex}[i]\}$  do
31             $F \leftarrow F \cup (i, j)$ 
32        return IMST-BINNARY-SEARCH ( $G^*, T_s^*, \mathbf{s}', F, k$ )
33      else
34         $K \leftarrow \lfloor \frac{\text{TotalCount}}{6 \cdot n} \rfloor$ 
35         $\mathcal{H} \leftarrow \emptyset$ 
36        for  $i \in \{1, \dots, m'\}$  do
37           $j_{\max} \leftarrow \text{MaxIndex}[i]$ 
38          for  $j \leftarrow \text{MinIndex}[i] ; j < j_{\max} ; j \leftarrow j + K$  do
39             $\mathcal{H} \leftarrow \mathcal{H} \cup (i, j)$ 
40         $\lambda \leftarrow \text{MEDIAN}(\mathcal{H})$ 
41         $kDiff \leftarrow \text{GET-DIFF}(T(\lambda), T_s^*)$  // Zwróć liczbę krawędzi  $e \in T(\lambda) \setminus T_s^*$ .
42        if  $kDiff = k$  then
43          return  $T(\lambda)$ 
44        else if  $kDiff > k$  then
45           $L \leftarrow \lambda$ 
46        else
47           $U \leftarrow \lambda$ 
```

---



fdf  
d  
gg  
fg  
fg  
fsfdf  
d  
fdfd  
fdf  
d  
gg  
fg  
fg

---

## 5.6 Podsumowanie rozdziału

Wychodząc od funkcji celu, którą otrzymaliśmy w części poświęconej programowaniu liniowemu, w tym rozdziale bardzo dokładnie omówiliśmy proces konstrukcji algorytmu rozwiązującego problem minimalnego drzewa rozpinającego w wersji INCREMENTAL, pochylając się nad wszystkimi problemami jakie w tym czasie napotkaliśmy i sugerując sposoby ich rozwiązania. Pozwoliło nam to otrzymać bardzo wydajny algorytm, z którego to będziemy korzystać przy próbie przyjrzenia się następnemu problemowi: optymalizacji odpornej z możliwością poprawy rozwiązania, który zdefiniowaliśmy w 3.18. O ile zagadnienie, któremu poświęciliśmy cały ten rozdział, znajduje się jeszcze w zasięgu naszych możliwości obliczeniowych (należy do problemów klasy  $P$  — ma wielomianowy algorytm go rozwiązymy), to któremu będziemy się chcieli przyjrzeć wykracza daleko poza nie [13], w związku z czym nasze dalsze rozważania nie skupią się na konstrukcji algorytmu dającego dokładne jego rozwiązania — będziemy chcieli zaprezentować nieco odmienny punkt widzenia, przedstawiając algorytmy lokalnego przeszukiwania.





# Odporna optymalizacja regeneracyjna w sąsiedztwie

---

Jedną z podstawowych i bardzo wydajnych metod konstrukcji rozwiązań problemów optymalizacyjnych jest metoda oparta na przeszukiwaniu bezpośrednich **sąsiadów** aktualnie posiadanego rozwiązania z nadzieją, że takie krokowe zachowanie się algorytmu będzie skutkowało szybkim zbliżaniem się do optymalnego rozwiązania. W poniższym rozdziale wskażemy trzy takie metody, przy czym jedną z nich — tą najefektywniejszą — omówimy bardziej szczegółowo. Choć wydawać by się mogło, że taka metoda rozwiązywania problemów optymalizacyjnych nie ma prawa się dobrze zachowywać ze względu na bardzo silne założenia dotyczące sposobu zachowywania się rozwiązań sąsiednich do obecnie badanego, to właśnie takie algorytmy — szczególnie **Tabu Search** — są najogólniejszym sposobem na rozwiązywanie problemów optymalizacyjnych.

## 6.1 Algorytm zachłanny

**Algorytm zachłanny** jest najbardziej podstawową i naturalną strategią poszukiwania rozwiązania dla problemów optymalizacyjnych w myśl zasady „jeśli wartość mojego rozwiązania jest niewiele gorsza od wartości optymalnej, za pomocą niewielkich zmian powiniensem tą wartość osiągnąć”. Fundament, na którym zbudowane jest to zdanie, zawiera bardzo ważne założenie określające charakter zachowania się poszczególnych rozwiązań problemu optymalizacyjnego. Zakładamy bowiem, że proporcjonalnie do zmian wprowadzonych w znalezionym rozwiązańiu, rośnie (bądź maleje) koszt takiego rozwiązania. Takim problemem jest na przykład rozpatrywany przez nas problem minimalnego drzewa rozpinającego, który leży u podstaw problemu, na którego rozwiązanie (aproksymację rozwiązań) poświęcimy cały poniższy rozdział:

$$\min_{\mathbf{x} \in X} \left( v(\mathbf{x}, \mathbf{s}) + \max_{\mathbf{s}' \in S} \min_{\mathbf{y} \in X_{\mathbf{x}}^k} v(\mathbf{y}, \mathbf{s}') \right) \quad (6.1)$$

### 6.1.1 Sąsiedztwo

Rozpoczniemy wyjątkowo od przedstawienia pseudokodu algorytmu zachłannego (zobacz Pseudokod 6)<sup>1</sup>, by następnie na jego podstawie wytłumaczyć pojęcie **sąsiedztwa**. Przypomnijmy, że funkcja  $v(\bullet, \mathbf{s})$ , zdefiniowana w drugim rozdziale, reprezentuje koszt rozwiązania problemu dla scenariusza  $\mathbf{s}$  (gdzie jako pierwszy parametr przekazywaliśmy jej binarny wektor  $\mathbf{x}$  reprezentujący to rozwiązanie). W odniesieniu do problemu minimalnego drzewa rozpinającego, zamiast wektora, będziemy przekazywać do niej drzewo — wartością za tej funkcji będzie suma kosztów krawędzi należących do tego drzewa. Dla rozróżnienia kosztu rozwiązania problemu minimalnego drzewa rozpinającego w wersji INCREMENTAL od tego dla wyrażenia 6.1, wprowadźmy oddzielne oznaczenia:  $v_{IMST}(\bullet, \mathbf{s})$  dla tego pierwszego oraz  $v_{RRMST}(\bullet, S)$  dla problemu odpornej optymalizacji przyrostowej z możliwością poprawy (ang. RRMST — *Robust Recoverable Incremental Minimum Spanning Tree*). Należy wziąć pod uwagę, że policzenie wartości tego drugiego wyrażenia wymaga od nas w między

<sup>1</sup>Ten i pozostałe pseudokody, chociaż dotyczące uniwersalnych algorytmów, będą przedstawiane z perspektywy problemu minimalnego drzewa rozpinającego (tak więc np. zamiast punktu startowego  $x$  z dziedziny dopuszczalnych rozwiązań  $X$ , będziemy mieli początkowe drzewo rozpinające  $T_s^*$ ). Dodatkowo, gdy będzie to uzasadnione, będziemy korzystać z wcześniej udowodnionego lematu 5.2.1 (zamiast grafu  $G = (V, E)$ , będziemy odwoływać się do  $G^* = (V, E^*)$ ).



czasie rozwiązywania szeregu problemów IMST oraz problemu adwersarza — drugi z nich oraz sposób jego rozwiązywania przedstawiliśmy w 3.4.2. Dlatego też zamiast przekazywać do funkcji pojedynczy scenariusz, tak jak to robiliśmy do tej pory, do wyrażenia  $v_{\text{RRMST}}(\bullet, \bullet)$  będziemy przekazywać zbiór scenariuszy dla problemu adwersarza. Mając na uwadze dotychczas omówione zagadnienia, nie będziemy więcej wracać do problemu wyliczania wartości tej funkcji, skupimy się zaś na algorytmach zwracających aproksymację wyrażenia  $\min_{\mathbf{x} \in X} v_{\text{RRMST}}(\mathbf{x}, S) = \min_{\mathbf{x} \in X} (v(\mathbf{x}, \mathbf{s}) + \max_{\mathbf{s}' \in S} \min_{\mathbf{y} \in X_{\mathbf{x}}^k} v(\mathbf{y}, \mathbf{s}'))$ .

---

**Pseudokod 6:** LOCAL-SEARCH ( $G^*, T_s^*, S, k$ )

---

**Wejście:**  $G^* = (V, E^*)$  — graf ze zbiorem krawędzi  $T_{s'}^* \cup T_s^*$ ,  
 $T_s^*$  — początkowe minimalne drzewo rozpinające dla scenariusza  $s$ ,  
 $S$  — zbiór scenariuszy adwersarza,  
 $k$  — parametr problemu IMST.

**Wyjście:**  $T^*$  — lokalnie optymalne rozwiązanie problemu IMST.

```

1 begin
2   Wybierz dowolne drzewo  $T$ , będące dopuszczalnym rozwiązaniem problemu IMST.
3   while znaleziono lepsze rozwiązanie do
4      $T \leftarrow \min \{v_{\text{RRMST}}(T', S) : T' \in N(T, T_s^*, k)\}$ 
5     if  $v_{\text{RRMST}}(T, S) < v_{\text{RRMST}}(T_s^*, S)$  then
6        $T_s^* \leftarrow T$ 
7     else
8       return  $T_s^*$ 
```

---

W przedstawionym pseudokodzie, linii 4, jesteśmy zainteresowani pozyskaniem ze zbioru  $N(T, T_s^*, k)$  najlepszego rozwiązania dopuszczalnego dla problemu IMST. Jednocześnie, przekazując do niej drzewo  $T$ , chcemy aby zwrócone rozwiązanie było do niego w określony sposób podobne. Skalę tego podobieństwa dla różnych problemów definiujemy inaczej, dla problemu minimalnego drzewa rozpinającego jakim się zajmujemy, zbiór generowany przez powyższe wyrażenie przybierze formę:

$$N(T, T_s^*, k) = \{T' : f(T', T_s^*) \leq k \wedge f(T', T) = 1\}, \quad (6.2)$$

i oznacza zbiór wszystkich tych drzew będących dopuszczalnymi rozwiązaniami problemu IMST ( $f(T', T_s^*) \leq k$ ), które jednocześnie są **sąsiadami** drzewa  $T$ . W naszym przypadku sąsiadem  $T$  nazwiemy dowolne drzewo  $T'$ , które różni się od tego pierwszego dokładnie jedną krawędzią. Wybór takiego drzewa  $T'$  spośród zbioru drzew  $N(T, T_s^*, k)$  będziemy nazywać **ruchem** — przejściem z jednego rozwiązania dopuszczalnego dla danego problemu do drugiego. Główna pętla algorytmu zachłannego 6 będzie powtarzana do momentu, w którym żadne sąsiednie rozwiązanie nie okaże się lepsze od tego, na podstawie którego wygenerowaliśmy otoczenie (stąd sama nazwa algorytmu — zachłanny). Powstaje naturalne pytanie — jak generować kolejne drzewa rozpinające, które na dodatek są sąsiadami innego, wskazanego przez nas ( $T$ )? Jednym z pomysłów na realizację tego zadania może być konstrukcja nowych drzew poprzez dodawanie do nich krawędzi nie należących do  $T$  a usuwaniu tych, które razem z dodaną przed chwilą krawędzią tworzą cykl [10].

---

**Pseudokod 7:** NEIGHBORHOOD ( $G, T$ )

---

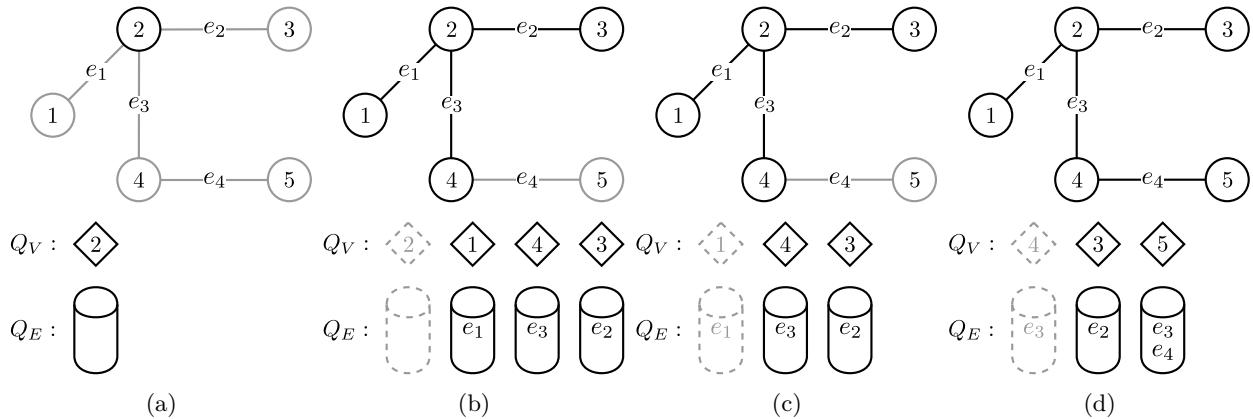
**Wejście:**  $G = (V, E)$ ,  
 $T$  — minimalne drzewo rozpinające dla grafu  $G$ .

**Wyjście:**  $N(T)$  — otoczenie drzewa  $T$ .

```

1 begin
2    $N(T) \leftarrow \emptyset$ 
3   foreach  $(i, j) \in E \setminus T$  do
4     foreach  $e \in v_i \xrightarrow{T} v_j$  do
5        $N(T) \leftarrow N(T) \cup (T \cup e_{ij} \setminus e)$ 
6   return  $N(T)$ 
```

---



Rysunek 6.1: Kolejne kroki zmodyfikowanego algorytmu BFS ( $T, v_s, v_t$ ) (ang. *Breadth-first search*), gdzie  $T$  jest drzewem rozpinającym graf  $G$ ,  $v_s$  oraz  $v_t$  są odpowiednio wierzchołkami początkowym i końcowym ścieżki  $v_s \rightsquigarrow v_t$ , dla której chcemy wyznaczyć zbiór krawędzi należących do  $T$  i do tej ścieżki. **(a)** Początkowa sytuacja algorytmu dla  $v_s = v_2$  i  $v_t = v_4$ . Na koniec kolejki  $Q_V$  został dodany wierzchołek początkowy. Odpowiadający mu element w kolejce  $V_E$  nie zawiera żadnych elementów. **(b)** W wyniku zdjęcia z kolejki pierwszego elementu  $v_2$ , algorytm dodał do  $Q_V$  wszystkie wierzchołki, do których prowadziły krawędzie ze zdążytego wierzchołka. Odpowiednio dla każdego takiego wierzchołka, do kolejki  $Q_E$  zostały dodane krawędzie, dzięki którym algorytm do nich dotarł. **(c)** Zgodnie z kolejnością ściągania elementów z kolejek, aktualnie badanymi elementami (zaznaczonymi w kolejce jasnym kolorem) są: wierzchołek  $v_1$  oraz zbiór krawędzi  $\{e_1\}$ . Z tego wierzchołka nie prowadzi jednak żadna nowa krawędź, badany wierzchołek nie jest też tym końcowym, toteż algorytm usuwa wskazane elementy. **(d)** W wyniku wykonania operacji na kolejnej parze elementów:  $(v_4, \{e_3\})$ , do kolejki priorytetowej  $Q_V$  został dodany<sup>2</sup> nowy jeszcze nieodwiedzony wierzchołek  $v_5 = v_t$ . Równolegle, do kolejki  $Q_E$ , dla dodanego wierzchołka, został dodany zbiór krawędzi będący sumą zbioru łuków, po których algorytm dotarł do analizowanego wierzchołka  $v_4$ , oraz łuku, po którym przeszedł on do nowego wierzchołka (zbiór  $\{e_3, e_4\}$ ). Fakt, że dodawany do kolejki  $Q_V$  element jest wierzchołkiem końcowym w ścieżce  $v_s \rightsquigarrow v_t$ , oznacza, że podany zbiór krawędzi jest jedną z możliwych ścieżek (w tym przypadku jedyną) pomiędzy tymi wierzchołkami. Algorytm kontynuujemy dopóki obie kolejki nie są puste, zapisując po drodze elementy z  $Q_E$ , które odpowiadają wierzchołkom  $v_t$ , dodawanym do kolejki  $Q_V$ .

Przez wyrażenie  $e \in v_i \xrightarrow{T} v_j$  (w linii 4 pseudokodu 7) rozumiemy wszystkie takie krawędzie  $e$ , które należą do ścieżki, której początek znajduje się w wierzchołku  $v_i$  a kończy w  $v_j$ . Dodatkowo jesteśmy zainteresowani tylko takimi krawędziami, które należą do wskazanego przez nas drzewa  $T$ . Tą własność możemy bardzo łatwo wymusić, szukając ścieżek tylko w obrębie danego drzewa rozpinającego (jako że z definicji łączy ono ze sobą wszystkie krawędzie tak więc na pewno istnieje choć jedna ścieżka pomiędzy wskazanymi wierzchołkami  $v_i$  oraz  $v_j$ ). Algorytmem służącym nam do generowania takich zbiorów może być na przykład zmodyfikowany algorytm przeszukiwania wszerz [5, 604–606]. Nasza modyfikacja polegałaby na tym, że zamiast jednej kolejki priorytetowej, posiadałabyśmy takich kolejek dwie, z czego pierwsza zachowałaby swoje oryginalne własności, elementami drugiej byłyby zbiory krawędzi reprezentujące ścieżki, które doprowadziły do konkretnego elementu z pierwszej listy, tak jak to pokazano na rysunkach od 6.1a do 6.1d.

### 6.1.2 Wady algorytmu zachłannego

Jak każdy algorytm zachłanny, i ten boryka się z tymi samymi problemami: lokalny charakter jego decyzji powoduje, że bardzo łatwo zatrzymać mu się w **lokalnym optimum**  $T^*$ , które charakteryzuje się tym, że dla dowolnego drzewa  $T' \in N(T, T_s^*, k)$  mamy  $v_{RRMST}(T^*, S) \leq v_{RRMST}(T', S)$ . W takiej sytuacji algorytm zachłanny oczywiście zakończy działanie z przekonaniem, że znalezione przez niego rozwiązanie jest optymalne globalnie, choć w rzeczywistości przeglądnięta przez niego przestrzeń rozwiązań dopuszczalnych jest bardzo

<sup>2</sup>Jako że dodawany wierzchołek jest wierzchołkiem końcowym ścieżki  $v_s \rightsquigarrow v_t$ , zamiast aktualizować obydwie kolejki, możemy odpowiadający mu (wierzchołkowi  $v_t$ ) zbiór krawędzi od razu przenieść do puli znalezionych ścieżek bez modyfikacji kolejek i przejść do następnego elementu.



mała i mogła „nie otrzeć” się nawet o prawidłowe rozwiązanie. Oczywistą konsekwencją powyższego jest w pełni deterministyczny schemat działania takiego algorytmu, tak więc jakość otrzymanego rozwiązania w pełni zależy od zdanego przez nas początkowego drzewa  $T_s^*$ . Dodatkowo algorytm nie uczy się — raz przeglądnięte przez niego rozwiązanie może być przeglądane dowolną liczbę razy, jeżeli zdarzy się, że dane rozwiązanie jest najlepsze w sąsiedztwie wielu innych drzew rozpinających. Jak zobaczymy w części poświęconej algorytmowi lokalnego przeszukiwania z listą ruchów zakazanych (ang. *Tabu Search*), mimo tak licznych i poważnych wad, algorytm ten jeszcze nam posłuży do konstrukcji dużo lepszego algorytmu.

## 6.2 Symulowane wyżarzanie

Algorytm **symulowanego wyżarzania** (ang. *Simulated annealing*) jest algorytmem mocno zbliżonym do poprzednio omawianego, jednak eliminuje pewne wady poprzedniego rozwiązania. Przede wszystkim, rezygnuje on z przeglądu wszystkich sąsiadów zadanego drzewa  $T$  (co wymuszało na nas wyrażenie  $\min \{v_{\text{RRMST}}(T', S) : T' \in N(T, T_s^*, k)\}$  w linii 4 pseudokodu 6). Zamiast tego, algorytm wprowadza pojęcie **temperatury**, która jest składową miary prawdopodobieństwa wybrania konkretnego sąsiada drzewa  $T$  z otoczenia  $N(T, T_s^*, k)$ . Na jej podstawie, zamiast przeglądać całość otoczenia  $N(T, T_s^*, k)$ , algorytm losuje z niego drzewo  $T'$ , oblicza wartość takiego rozwiązania ( $v_{\text{RRMST}}(T', s)$ ), oraz na podstawie tej danej oraz temperatury decyduje się na wybór takiego drzewa, bądź losuje następne, do momentu, w którym dane przekazane do funkcji wyliczającej prawdopodobieństwo wybrania danego rozwiązania  $T'$  jest dostatecznie wysokie. Algorytm rozpoczyna pracę od pewnej temperatury maksymalnej, a następnie ją sukcesywnie obniża, powodując spadek wartości obliczanych prawdopodobieństw [14] [8].

Zaletami takiego rozwiązania w porównaniu do poprzedniego algorytmu są niewątpliwie: brak konieczności przeglądu wszystkich sąsiednich rozwiązań (przeglądamy kolejne losowe rozwiązania do momentu, w którym zdefiniowane wcześniej reguły pozwolą nam na wybranie danego rozwiązania) oraz większa odporność na pozostanie w obszarze lokalnego optimum (ze względu na wprowadzoną losowość wyboru następnego rozwiązania z sąsiedztwa). Nadal jednak taka szansa istnieje, dlatego skupimy się teraz na trzecim i najbardziej złożonym rozwiązaniu, które w zamian nie jest obarczone wadami poprzedników — algorytmem lokalnego przeszukiwania z listą ruchów zakazanych.

## 6.3 Przeszukiwanie z listą Tabu

Algorytm **Tabu Search** [7] swoją nazwę zawdzięcza wprowadzanemu przez siebie dodatkowemu elementowi jakim jest **lista ruchów zakazanych** (ang *tabu list*). Taką listą będziemy nazywać zbiór ruchów, których z różnych względów bronimy algorytmowi wykonać w trakcie poszukiwania rozwiązania — w ten sposób, jeżeli **kadencja** takiego elementu na liście *tabu* będzie wystarczająco długa (przez pojęcie kadencji rozumiemy liczbę iteracji algorytmu jaką musi wykonać aby mieć możliwość ponownego wyboru zakazanego ruchu), zapewniamy sobie brak wpadania przez algorytm w cykle podczas poszukiwania rozwiązania<sup>3</sup>. Dodatkowo będziemy chcieli rozwiązać kwestię groźby trafienia i pozostańia w lokalnym minimum — aby temu zapobiegać, będziemy co jakiś czas resetować nasz algorytm, który w takiej formie (upraszczając) można przyrównać do wielokrotnego uruchamiania algorytmu zachłannego, za każdym razem dla innych drzew początkowych. Przy tym wszystkim będziemy oczywiście chcieli abyśmy w przypadku odnalezienia najlepszego rozwiązania do tej pory, mogli ruch do niego prowadzący, ignorując listę ruchów zakazanych, wykonać. O rozwiązaniu będącym efektem wykonania zabronionego ruchu będziemy mówić, że spełniało one **kryterium aspiracji**.

Aby zapewnić sobie przegląd jak największej przestrzeni rozwiązań dopuszczalnych problemu IMST, będziemy chcieli aby każde następne drzewo będące punktem wyjścia po restarcie algorytmu, było jak najbardziej oddalone od drzew do tej pory przeanalizowanych. Innymi słowy, będziemy chcieli zaradzić kolejnej wadzie, którą wykazaliśmy przy okazji analizy algorytmu zachłannego — stosunkowo niewielkiego obszaru

<sup>3</sup>Analogicznie jak w omawianym wcześniej zmodyfikowanym algorytmie BFS ( $T, v_s, v_t$ ) (patrz rysunek 6.1), tak w przypadku algorytmu TABU SEARCH, algorytm nie podejmie decyzji powtórzenia pewnych ruchów, gdyż będzie wiedział, że taki ruch już wykonał (można powiedzieć, że lista *tabu* jest częściowym zapisem historii wykonywanych ruchów).



rozwiązań, które ten algorytm badał, zanim zatrzymywał się w lokalnym optimum. Dla algorytmu TABU SEARCH nie jest inaczej — podczas jednego pełnego przebiegu jesteśmy w stanie przeanalizować niewielką tylko część dopuszczalnych rozwiązań, dlatego sztucznie będziemy wymuszać rozpoczęwanie wyszukiwania rozwiązania od nowa, zaczynając szukać w zupełnie innych miejscach, tak jak to zaprezentowano na pseudokodzie 8.

### 6.3.1 Sąsiedztwo

Analizując wspomniany przed chwilą pseudokod, nietrudno nie zauważyc, że w linijce 9 nagromadziliśmy bardzo dużą liczbę warunków — w tym miejscu założyliśmy również, że skoro jesteśmy zainteresowani tylko takim sąsiadem  $T'$  drzewa  $T$ , dla którego wartość  $v_{\text{RRMST}}(T', S)$  jest najmniejsza spośród wszystkich wartości  $v_{\text{RRMST}}(T'', S)$  dla drzew  $T'' \in N(T, T_s^*, k)$ , to nie potrzebujemy pamiętać jednocześnie pozostałych drzew w sąsiedztwie<sup>4</sup>. W naszym przypadku zatem drzewem  $T_N$  jest drzewo które:

- należy do bezpośredniego sąsiedztwa drzewa  $T$  (różni się od niego dokładnie jedną krawędzią) oraz ma nie więcej niż  $k$  krawędzi, które nie należą do drzewa początkowego  $T_s^*$  ( $T_N \in N(T, T_s^*, k)$ ),
  - ruch polegający na przejściu od  $T$  do innego dopuszczalnego rozwiązania  $T_N$  nie znajduje się na liście TABU ( $T' \notin \text{TABU}$ ),
  - lub wartość rozwiązania problemu RRMST dla drzewa  $T_N$  jest lepsza (mniejsza) niż najmniejsza taka wartość znaleziona do tej pory (kryterium aspiracji) —  $v_{\text{RRMST}}(T', S) < C^{\Delta*}$ ),
- $v_{\text{RRMST}}(T^N, S)$  zwraca najmniejszą wartość spośród wszystkich drzew, które spełniają wszystkie powyższe kryteria ( $T_N \leftarrow \min arg_{T'} \{v_{\text{RRMST}}(T', S) : \dots\}$ ).

W przypadku, gdy zbiór  $N(T, T_s^*, k)$  jest na tyle duży, że wykonanie powyższych obliczeń dla każdego drzewa należącego do tego zbioru w sposób nieakceptowalny wpływa na czas wykonywania się algorytmu, możemy zastosować dodatkowo **kryterium aspiracji plus**, które przyjmuje parametry:

- $\min$  — najmniejsza liczba drzew z sąsiedztwa, które musimy przeglądać ( $\min = \min \{ \min, |N(T, T_s^*, k)| \}$ ),
- $\max$  — liczba drzew ze zbioru  $N(T, T_s^*, k)$ , po którym przeglądnięciu musimy zwrócić drzewo, dla którego wartość wyrażenia  $v_{\text{RRMST}}(\bullet, S)$  była najmniejsza do tej pory — reszty drzew rozpinających nie przeglądamy ( $\max = \min \{ \max, |N(T, T_s^*, k)| \}$ ),
- $v_{\min}$  — próg aspiracji, po którego przekroczeniu przez wartość  $v_{\text{RRMST}}(\bullet, S)$  dla pierwszego drzewa (drzewo  $T^+$  i niech będzie ono i tym przeglądanym drzewem), będziemy przeglądać jeszcze  $p$  kolejnych w kolejności sąsiadów drzewa  $T$  ze zbioru  $N(T, T_s^*, k)$ , gdzie
- $p = \min \{i + \text{Plus}, \max\}$ .

W takiej sytuacji, zamiast wykonywać obliczenia dla wszystkich drzew w sąsiedztwie, przerywamy je w chwili, gdy zajdą następujące warunki:

- przeglądniemy  $\max$  drzew lub
  - wartość funkcji  $v_{\text{RRMST}}(\bullet, S)$  dla pewnego drzewa  $T^+$  jest mniejsza bądź równa niż ustalony z góry parametr  $v_{\min}$  oraz
  - wykonamy potrzebne obliczenia jeszcze dla  $p$  kolejnych drzew, należących do  $N(T, T_s^*, k)$ .

W tym przypadku, zwróconym drzewem  $T_N$  będzie albo najlepsze spośród pierwszych  $\max$  drzew, albo pierwsze takie drzewo, dla którego  $v_{\text{RRMST}}(\bullet, S) \leq v_{\min}(T^+)$ , gdzie  $v_{\min}$  jest parametrem stosowanego kryterium aspiracji, albo najlepsze znalezione po nim ( $T_N = \min arg_{T'} \{v_{\text{RRMST}}(T', S) : T' \in \{T^+, T^{+2}, \dots, T^{+p}\}\}$ ).

<sup>4</sup>Tak jak na przykład pseudokod algorytmu TABU SEARCH w [10], gdzie są rozdzielone kroki: wygenerowanie całego zbioru  $N(T, T_s^*, k)$  a następnie wybranie z niego drzewa o jak najmniejszej wartości rozwiązania dla problemu RRMST.

**Pseudokod 8:** TABU-SEARCH ( $G, T, S, k, it_{max}$ )

---

**Wejście:**  $G = (V, E)$ ,  
 $T_s^*$  — początkowe minimalne drzewo rozpinające dla scenariusza  $s$ ,  
 $S$  — zbiór scenariuszy adwersarza,  
 $k$  — parametr problemu IMST.  
 $it_{max}$  — liczba iteracji, po której następuje zresetowanie algorytmu.

**Wyjście:**  $T^{\Delta*}$  — najlepsze znalezione do tej pory rozwiązanie problemu RRMST.

```

1 begin
2    $T \leftarrow \text{RANDOM-MST}(G)$ 
3    $T^{\Delta*} \leftarrow T$ 
4    $C^{\Delta*} \leftarrow v_{\text{RRMST}}(T, S)$ 
5    $TABU \leftarrow \emptyset$ 
6    $E^c \leftarrow T^c$ 
7    $it \leftarrow 0$ 
8   while nie zaszło kryterium zatrzymania do
9      $T_N \leftarrow \min arg_{T'} \{v_{\text{RRMST}}(T', S) : T' \in N(T, T_s^*, k) \wedge (T' \notin TABU \vee v_{\text{RRMST}}(T', S) < C^{\Delta*})\}$ 
10     $C_N \leftarrow v_{\text{RRMST}}(T_N, S)$ 
11     $it \leftarrow it + 1$ 
12    if  $C_N < C^{\Delta*}$  then
13       $T^{\Delta*} \leftarrow T_N$ 
14       $C^{\Delta*} \leftarrow C_N$ 
15       $E^c \leftarrow E^c \cup T_N^c$ 
16    if  $it > it_{max}$  then
17       $T \leftarrow \text{RANDOM-MST}((V, E^c))$ 
18       $C \leftarrow v_{\text{RRMST}}(T, S)$ 
19      if  $C < C^{\Delta*}$  then
20         $T^{\Delta*} \leftarrow T$ 
21         $C^{\Delta*} \leftarrow C$ 
22       $TABU \leftarrow \emptyset$ 
23       $E^c \leftarrow T^c$ 
24       $it \leftarrow 0$ 
25    else
26       $T \leftarrow T_N$ 
27      UPDATE-TABU()
28  return  $T^{\Delta*}$ 
```

---

**6.3.2 Losowe drzewo rozpinające i strategia dywersyfikacji**

Omawiając cele jakie chcemy zrealizować implementując algorytm TABU SEARCH, wspomnialiśmy także o chęci resetowania algorytmu i rozpoczętania jego pracy na nowo lecz z rozwiązaniem początkowym  $T$  jak najdalej oddalonym od któregokolwiek drzewa, które pojawiło się w procesie wykonywania się iteracji poprzedniej. W tym celu wprowadziliśmy następujące oznaczenia:  $E^c$  oraz  $T^c$ , gdzie ten pierwszy jest konstruowany z sumy tych drugich.  $T^c$  zaś będziemy nazywali **alternatywą najgorszego przypadku** dla drzewa  $T$  — innymi słowy będzie to minimalne drzewo rozpinające (znalezione przez zwykły algorytm rozwiązujący problem MST) dla następującego scenariusza:

$$c_i^{S^-}(T) = \begin{cases} \max \{c_i^s : s \in S\} & e_i \in T, \\ \min \{c_i^s : s \in S\} & e_i \notin T \end{cases} \quad \forall i \in \{1, 2, \dots, m\}. \quad (6.3)$$

Celem takiego zabiegu jest wymuszenie zwrócenia przez algorytm minimalnego drzewa rozpinającego, którego część wspólna z drzewem  $T$  jest jak najmniejsza (czyli jednocześnie jest jak najbardziej oddalone od zbioru sąsiadów drzewa  $T$  —  $\{T' : f(T', T) = 1\}$ ). Widzimy, że ilekroć poprawiamy dotychczas odna-



lezione rozwiązanie (poza sytuacją, w której resetujemy algorytm i „przez przypadek” nowo wygenerowane drzewo okazuje się być tym najlepszym — linie 17–21), do zbioru  $E^c$  dołączamy krawędzie wygenerowanych alternatyw dla tych rozwiązań ( $E^c \leftarrow E^c \cup T_N^c$ ). Taka strategia skutkuje tym, że po  $it_{max}$  iteracjach rozpoczynamy poszukiwanie rozwiązania w zupełnie innym fragmencie przestrzeni rozwiązań dopuszczalnych, przez co mamy większą szansę „natknąć” się na lokalne optimum, które jest również optymalne globalnie.

Generowanie losowych drzew rozpinających dany graf (linie 2 oraz 17) jest zadaniem prostym — w tym przypadku również możemy skorzystać ze zmodyfikowanej wersji algorytmu BFS, bądź dowolnego innego, błądzącego po grafie w sposób losowy (choć istnieją algorytmy od nich efektywniejsze [18]). Najprostsza implementacja wraz z odwiedzaniem kolejnych krawędzi po prostu losowo wybiera wychodzącą z niego krawędź spośród dostępnych, dbając o to aby zakończyć ten proces w chwili wybrania ostatniej  $|V| - 1$ -ej krawędzi (odwiedzając przy tym każdy wierzchołek tylko raz).

### 6.3.3 Lista ruchów zakazanych i kryterium końca

Ostatnim elementem wymagającym omówienia jest lista TABU. Dzięki niej jesteśmy w stanie uniknąć zbyt częstego powtarzania tych samych ruchów czy wymusić na algorytmie przegląd coraz to dalszego sąsiedztwa danego drzewa początkowego w kolejnych iteracjach. Niestety, zastosowanie listy ruchów zakazanych ma swoje wady: fakt istnienia takiej listy może oznaczać, że w pewnych sytuacjach najbardziej optymalny ruch jest zakazany i nie będziemy mogli go wykonać (prawdopodobieństwo takiego zdarzenia jest tym większe im na dłuższe kadencje się zdecydujemy). Przeciwdziałać takim przypadkom ma, wprowadzone przez nas, kryterium aspiracji, które w sytuacjach takich jak opisana pozwala nam całkowicie zignorować listę TABU. Wokół samego wyboru długości kadencji poszczególnych elementów listy gromadzi się wiele problemów; z jednej strony powiedzieliśmy, że zbyt długie cykle mogą powodować pomijanie pewnych optymalnych rozwiązań, lecz dzięki temu, że niektóre ruchy są zakazane przez bardzo długi okres, mamy szansę przeglądać szerszy obszar rozwiązań dopuszczalnych wokół punktu startowego. Pociąga to za sobą również wady — przeszukując coraz to większy obszar przy niezmiennej liczbie iteracji tracimy na jakości uzyskiwanych wyników (badany zbiór rozwiązań jest rzadszy przez co większe jest prawdopodobieństwo pominięcia rozwiązania, które powinniśmy wybrać). Z kolei skłanianie się ku krótkim kadencjom podnosi co prawda naszą szansę na znalezienie optymalnego rozwiązania, lecz wraz z nią rośnie zagrożenie, że zwróconym rozwiązaniem będzie optimum lokalne, bądź wpadnięcie w cykl.

Z każdą następną iteracją, wraz z wywołaniem procedury UPDATE-TABU, pozostały czas kadencji każdego z elementów powinien ulec zmniejszeniu, zaś ruchy, których czas kadencji dobiegł końca — usunięte z listy, co czyni je ponownie możliwymi do wyboru. Dodatkowo, jako najprostsze kryterium końca algorytmu, możemy przyjąć warunek sprawdzający czy całkowita liczba iteracji nie została przekroczena. Jeżeli tak, zwróciemy najlepsze do tej pory odnalezione rozwiązanie.

### 6.3.4 Podsumowanie rozdziału

Algorytm TABU SEARCH jest tylko ideą — większa część pokazanego pseudokodu jest opcjonalna i tylko od tego do jakiego problemu się tę ideę zastosuje zależy finalny kształt tego algorytmu. Wszystkie podane parametry, takie jak okres kadencji, liczba iteracji, kryterium zatrzymania się algorytmu czy też parametry aspiracji plus — powinny być rozpatrywane dla każdego problemu oddzielnie i nie istnieje ogólna formuła wskazująca ich wartości.

ff df fffd fd



# **Wyniki eksperymentalne**

---

## **7.0.1 Środowisko testowe**

---

### **7.1 Gęstość grafu**

#### **7.1.1 Metodyka i dane**

#### **7.1.2 Wyniki i wykresy zależności**



# Zakończenie

---



# Bibliografia

---

- [1] Ahuja, Ravindra K., Magnanti, Thomas L., Orlin, and James B. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
- [2] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, Wrzesień 2009.
- [3] I. Averbakh and V. Lebedev. Interval data min–max regret network optimization problems. *Discrete Applied Mathematics*, 138(3):289–301, Kwiecień 2004.
- [4] Chazelle and Bernard. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, Listopad 2000.
- [5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, 7 edition, 2014.
- [6] Apostolos Dimitromanolakis. Analysis of the golomb ruler and the sidon set problems and determination of large near-optimal golomb rulers. Master’s thesis, University of Toronto, 2002.
- [7] Fred W. Glover and Manuel Laguna. *Tabu Search*. Springer, 1998.
- [8] Filip Jany. Równoległe implementacje algorytmów lokalnego przeszukiwania. Master’s thesis, Politechnika Wrocławskiego, 1 2015.
- [9] A. Kasperski and P. Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, Marzec 2006.
- [10] Adam Kasperski, Mariusz Makuchowski, and Paweł Zieliński. A tabu search algorithm for the minmax regret minimum spanning tree problem with interval data. *Journal of Heuristics*, 18(4):593–625, 2012.
- [11] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers., Boston, 1997.
- [12] Thomas L. Magnanti and Laurence A. Wolsey. Optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503–615. Elsevier, 1995.
- [13] Ebrahim Nasrabadi and James B. Orlin. Robust optimization with incremental recourse. *CoRR*, abs/1312.4075, 2013.
- [14] Nikulin and Yury. Simulated annealing algorithm for the robust spanning tree problem. *Journal of Heuristics*, 14(4):391–402, 2008.
- [15] Papadimitriou, Christos H., Steiglitz, and Kenneth. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.
- [16] Tomasz Strzałka. Algorytmy wyznaczania najkrótszych ścieżek w rzeczywistych sieciach drogowych. Master’s thesis, Politechnika Wrocławskiego, 1 2015.
- [17] Tarjan and R. E. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

- [18] Wilson and David Bruce. Generating random spanning trees more quickly than the cover time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 296–303, New York, NY, USA, 1996. ACM.
- [19] Hande Yaman, Oya Ekin Karasan, and Mustafa C. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, Sierpień 2001.
- [20] Onur Şeref, Ravindra K. Ahuja, and James B. Orlin. Incremental network optimization: Theory and algorithms. *Operations Research*, 53(3):586–594, Styczeń 2008.





# Biblioteka: Incremental MST

```
/  
├── PRESENTATIONS  
├── REFERENCES  
├── SCRIPTS  
├── THESIS  
├── TMH_EXAMPLES  
├── TMH_LIBRARY  
├── TMH_GRAPHHELPER  
├── clear.bash  
└── README.md  
/  
├── clear.bash ...  
└── SCRIPTS  
    ├── compileProject.bash ... Jak omawialiśmy  
    │           wcześniej --- służy do  
    │           komplikacji całego projektu.  
    ├── generatePlot.bash ... Generuje wykresy na  
    │           podstawie danych,  
    │           wygenerowanych przez  
    │           przykładowy program  
    │           TMH_EXAMPLES.  
    └── makeGraphViz.bash ... Generuje wizualizację  
        grafu zdefiniowanego  
        w pliku zgodnym z opisany  
        standardem.
```



```

/
  └─SCRIPTS
    └─randDistance.pl ... Pozwala na szybką
        modyfikację wszystkich
        kosztów ścieżek w pliku,
        definiującym graf.
    └─TMH_GRAPHHELPER... Pomocniczy projekt do
        generowania danych
        wejściowych (napisany
        w języku JAVA).
/Scripts/compileProject.bash
CRTL + ALT + T
R_USATEST_DKA
(ang. Directed acyclic graph)
pcname@user:/ Scripts$ bash compileProject.bash

```

Rysunek A.1: Wygenerowany wykres dwóch funkcji dla przykładowego wywołania skryptu GENERATEPLOT.BASH.

```

digraph G {
    rankdir=LR;
    node [shape = circle];
    1 --> 2 [ label = "2" ];
    1 --> 3 [ label = "6" ];
    2 --> 3 [ label = "3" ];
    2 --> 4 [ label = "4" ];
    2 --> 5 [ label = "5" ];
    3 --> 5 [ label = "1" ];
    5 --> 4 [ label = "2" ];
}

```

(a)

(b)

Rysunek A.2: Wygenerowana ilustracja grafu przez skrypt MAKEGRAPHVIZ.BASH. (a) Kod pośredni pomiejszy danymi wejściowymi w standardowym formacie DIMACS IMPLEMENTATION CHALLENGE a finalnym rysunkiem. Kod pośredni jest zapisany w języku DOT. (b) Rysunek grafu wygenerowany przez program GRAPHVIZ.

<http://algs4.cs.princeton.edu/home/> na licencji GNU GPLv3

$$\begin{aligned}
 f(x) + g(x) &= \cos(x) + \sin(x) + h(x) \\
 g(x) &= \sin(x) \\
 f(x) &= \cos(x) + h(x) \\
 s(x) &= \arcsin(x)
 \end{aligned}$$