Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Problem liczby skoków w zbiorach częściowo uporządkowanych

Kombinatoryczne algorytmy aproksymacyjne, przeszukiwanie wyczerpujące i złożoność obliczeniowa

rozprawa doktorska

Przemysław Krysztowiak
Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika w Toruniu

Promotor rozprawy
prof. dr hab. Maciej M. Sysło
Uniwersytet Mikołaja Kopernika w Toruniu
Uniwersytet Wrocławski

Toruń, kwiecień 2014

Oświadczenie autora rozprawy:

oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

.....	.....
18 kwietnia 2014 roku	*Przemysław Krysztowiak*

Oświadczenie promotora rozprawy:

niniejsza rozprawa jest gotowa do oceny przez recenzentów.

.....	.....
18 kwietnia 2014 roku	*prof. dr hab. Maciej M. Sysło*

Spis rzeczy

Streszczenie

Głównym problemem rozprawy doktorskiej jest minimalizacja liczby skoków posetu. Problem skoków dla danego posetu P polega na znalezieniu rozszerzenia liniowego, które minimalizuje liczbę sąsiadujących par elementów, nieporównywalnych w P. NP-trudność tego problemu została najpierw wykazana przez Pulleyblanka [56], a następnie na posetach przedziałowych przez Mitas [52].

Proponujemy kilka nowych algorytmów dla tego problemu. Najwięcej uwagi poświęcamy posetom przedziałowym. Dla posetów przedziałowych w latach 90-tych zaproponowano trzy wielomianowe algorytmy aproksymacyjne o współczynniku 3/2. Głównym rezultatem rozprawy jest przełamanie tego współczynnika. Poprawiamy algorytm podany przez Mitas i otrzymujemy aproksymację ze współczynnikiem 1.484. Ponadto przedstawiamy algorytm genetyczny dla problemu skoków na posetach przedziałowych, a także szybki algorytm dokładny dla tej klasy, działający w czasie $\mathcal{O}(1.47^n \cdot \mathrm{poly}(n))$.

W przypadku ogólnym, prezentujemy adaptację algorytmu przeszukiwania z zakazami działającą w oparciu o półsilnie zachłanne rozszerzenia liniowe, sformułowane przez Sysłę. Podejmujemy również temat posetów dwuwymiarowych. W klasie dwuwymiarowych posetów przedziałowych otrzymujemy algorytm aproksymacyjny dla problemu skoków ze współczynnikiem 4/3.

Praktyczna część pracy obejmuje eksperymentalną analizę wydajności algorytmów przybliżających liczbę skoków.

Słowa kluczowe: teoria grafów, poset, poset przedziałowy, poset dwuwymiarowy, liczba skoków, rozszerzenie liniowe, optymalizacja kombinatoryczna, algorytm aproksymacyjny, pakowanie podgrafów, pokrycie zbioru, problem komiwojażera, szeregowanie zadań.

Klasyfikacja tematyczna AMS: 06A07, 68R05, 68R10, 90C10, 90C27, 90C57, 90C59, 94C15.

Abstract

The main problem considered in this thesis is to minimize the jump number of a poset. The jump number problem for a given poset P is to find a linear extension minimizing the number of adjacent pairs which are incomparable in P. NP-hardness of this problem was first established by Pulleyblank [56], and later for interval orders by Mitas [52].

In the thesis, some new algorithms for this problem are proposed. We focus mainly on interval orders. In the 1990's, three polynomial-time approximation algorithms have been given for interval orders with approximation ratio of 3/2. The main result of this thesis is an improvement of this approximation ratio. We enhance the algorithm given by Mitas and we obtain a 1.484-approximation algorithm. Moreover, we present a genetic algorithm for the jump number problem on interval orders, and a fast exact algorithm for this class, whose time complexity is bounded by $\mathcal{O}(1.47^n \cdot \mathrm{poly}(n))$.

In the general case, we present an adaptation of the tabu search technique, based on semi-strongly greedy linear extensions, defined by Sysło. We also undertake the jump number of two-dimensional orders. We obtain a 4/3-approximation algorithm for the class of two-dimensional interval orders.

In addition, the thesis contains an experimental analysis of efficiency of algorithms to approximate the jump number.

Key words: graph theory, poset, interval order, two-dimensional poset, linear extension, combinatorial optimization, approximation algorithm, subgraph packing, set cover, traveling salesman problem, task scheduling.

AMS subject classification: 06A07, 68R05, 68R10, 90C10, 90C27, 90C57, 90C59, 94C15.

Wprowadzenie

Tematem tej rozprawy są algorytmy dla problemu minimalizacji liczby skoków posetu (zbioru częściowo uporządkowanego). Problem ten był badany już w latach 70-tych: rozpowszechnili go Chein i Martin [13] artykułem z 1972 roku, w którym przypisali sformułowanie problemu Kuntzmannowi i Verdillonowi [49] w 1971 roku. Pierwszy dowód NP-trudności dla tego problemu przedstawił Pulleyblank [56] w 1981 roku.

Problem liczby skoków dla danego posetu P polega na znalezieniu rozszerzenia liniowego, które minimalizuje liczbę sąsiadujących par elementów nieporównywalnych w P. Klasycznym zastosowaniem problemu jest szeregowanie zadań na jednej maszynie. Załóżmy, że oprócz zadań mamy ograniczenia orzekające, że pewne zadania nie mogą się rozpocząć, zanim nie skończą się inne (poprzednicy w posecie). Przyjmijmy ponadto, że wykonanie zadania bezpośrednio po zadaniu, które nie było jego poprzednikiem, wymaga ponownego przygotowania maszyny. Oznacza to dodatkowy koszt (skok). Celem jest znalezienie takiej kolejności wykonywania zadań, która minimalizuje liczbę skoków.

Do problemu skoków często stosuje się aparat teoriografowy. Na przykład można rozpatrywać problem na diagramie Hassego posetu. Każde rozszerzenie liniowe jest jakimś podziałem posetu na łańcuchy, więc w przypadku digrafu Hassego mamy szczególny podział na ścieżki. Według tej interpretacji, rozwiązanie problemu skoków jest równoważne minimalizacji liczby łuków, jakie trzeba dodać do digrafu, by uzupełnić go do digrafu półhamiltonowskiego. Problem skoków można też zredukować do asymetrycznego problemu komiwojażera. Z kolei w podejściu zapoczątkowanym przez Sysłę, z posetami stowarzysza się diagramy łukowe, a więc elementy posetu są reprezentowane łukami, a nie wierzchołkami digrafu. Aby znaleźć liczbę skoków, należy zminimalizować liczbę łuków, które trzeba dodać do digrafu, by utworzyć w nim drogę eulerowską ze źródła do ujścia.

Jeśli ograniczymy rozważania do węższych klas posetów, otrzymamy dalsze redukcje do znanych problemów optymalizacji kombinatorycznej. W rozprawie skupiamy się głównie na posetach przedziałowych. Jak wykazała Mitas [52], problem skoków na tych posetach redukuje się do pakowania wierzchołkowo rozłącznych krawędzi i nieparzystych cykli w pewnym grafie. Ceroi [9], badając posety dwuwymiarowe, zauważył, że problem skoków w tej klasie jest równoważny z wyznaczeniem zbioru niezależnego w grafie przecięć prostokątów na płaszczyźnie, o największej wadze, gdzie prostokąty odpowiadają

łańcuchom posetu. Na posetach dwudzielnych problem redukuje się do problemu najliczniejszego skojarzenia z dodatkowymi warunkami [11].

Problem ma bogatą literaturę, najistotniejsze rezultaty cytujemy w podrozdziale 2.2. Wiele wyników uzyskano dla posetów dwudzielnych, dla których problem pozostaje NP-trudny. W latach 90-tych intensywnie badano posety przedziałowe. Pojawiły się trzy różne algorytmy aproksymacyjne, wszystkie ze współczynnikiem $\frac{3}{2}$, opublikowane odpowiednio przez Felsnera [27], Sysłę [69] i Mitas [52]. Udowodniono też NP-trudność problemu w tej klasie [52]. Otwartym problemem badawczym pozostaje złożoność i aproksymacja liczby skoków na posetach dwuwymiarowych.

Rozprawa poświęcona jest w głównej mierze posetom przedziałowym. Wobec faktu, że trzy algorytmy mają ten sam współczynnik aproksymacji, można było sądzić, że nie da się go poprawić. Najważniejszym wynikiem rozprawy jest algorytm o współczynniku aproksymacji 1.484 dla posetów przedziałowych, omówiony w podrozdziale 3.7. Algorytm jest konsekwencją poprawy rezultatów Mitas. Ponadto, podano algorytm obliczający liczbę skoków posetu przedziałowego w czasie $\mathcal{O}(1.47^n \cdot \mathrm{poly}(n))$ (podrozdział 3.9). Zaproponowano też metaheurystyki: dla posetów przedziałowych, w oparciu o fakty wykorzystywane w algorytmach kombinatorycznych, sformułowano algorytm genetyczny (podrozdział 3.8). Dla ogólnego przypadku posetów zaprezentowano w podrozdziale 2.4 algorytm przeszukiwania z zakazami (tabu search), który bazuje na ścieżkach silnie- i półsilnie zachłannych, wprowadzonych przez Sysłę i z sukcesem wykorzystanych w dotychczasowych algorytmach dla problemu skoków.

Zanim poprawiono algorytm aproksymacyjny, znane wcześniej algorytmy zostały gruntownie przetestowane i porównane. Udało się zidentyfikować trudne instancje dla wszystkich trzech algorytmów, wymuszające największe możliwe odchylenie od optimum. Otrzymano liczną bazę nietrywialnych posetów, na których można testować nowe algorytmy. Rozprawa ma charakter eksperymentalno-teoretyczny. Doświadczenia z algorytmami aproksymacyjnymi są opisane w podrozdziałach 3.2, 3.3, 3.4, 3.5, 3.6, a baza trudnych posetów – w podrozdziale 3.10.

Skoro problem skoków jest otwarty na posetach dwuwymiarowych, podjęto próbę zastosowania niektórych algorytmów do tej klasy posetów. W przypadku, gdy posety dwuwymiarowe są zarazem przedziałowe, otrzymano aproksymację ze współczynnikiem $\frac{4}{3}$ (podrozdział 4.1). W podrozdziale 4.3 zawarto ponadto wyniki doświadczeń

z algorytmem przeszukiwania z zakazami na posetach dwuwymiarowych.

Rezultaty uzyskane w toku prac nad rozprawą doktorską były prezentowane podczas kilku konferencji, m.in. *EuroComb'11* w Budapeszcie i *Combinatorics 2012* w Perugii [45]. Poprawiony współczynnik aproksymacji dla posetów przedziałowych został opublikowany w artykule [44]. Baza trudnych posetów została omówiona w pracy [46] i na stronie internetowej [47], przeszukiwanie z zakazami opisano w [48], a doświadczenia z algorytmami na posetach przedziałowych w [43].

Omówienie wyników rozprawy

Pierwszy rozdział rozprawy zawiera pojęcia z dziedziny złożoności obliczeniowej oraz z teorii grafów, które są używane w dalszej części pracy.

Drugi rozdział rozpoczyna się od sformułowania problemu skoków. Następnie zarysowano w nim aktualny stan wiedzy o tym problemie. Obszerną część tego rozdziału stanowi przegląd algorytmu Sysły (podrozdział 2.3), który był jednym z pierwszych algorytmów dla problemu skoków. Czwarty podrozdział tego rozdziału jest prezentacją nowego algorytmu przeszukiwania z zakazami (tabu search) dla minimalizacji liczby skoków na dowolnych posetach. Algorytm ten jest rozwinięciem podejścia Sysły i eksploatuje tę samą przestrzeń poszukiwań, złożoną z półsilnie zachłannych rozszerzeń liniowych. Proponowany algorytm przeszukiwania z zakazami opisano również w artykule [48].

Trzeci rozdział rozprawy poświęcony jest posetom przedziałowym. W latach 90-tych zaproponowano trzy różne algorytmy $\frac{3}{2}$-aproksymacyjne dla problemu skoków na posetach przedziałowych. Po przytoczeniu charakteryzacji tych posetów, omówiono kolejno znane wcześniej algorytmy, a także przedstawiono wyniki eksperymentów obliczeniowych. W oparciu o przygotowane implementacje algorytmów Felsnera, Sysły i Mitas, porównano doświadczalnie te algorytmy, a także znaleziono trudne instancje wejściowe (podrozdziały 3.2 – 3.6). Doświadczenia te opisano również w artykule [43]. Ponadto, w podrozdziale 3.3 przetestowano też algorytm przeszukiwania z zakazami, przedstawiony w rozdziale drugim.

W dalszej części rozprawy zawarto główny teoretyczny rezultat, czyli algorytm 1.484-aproksymacyjny dla problemu skoków na posetach przedziałowych (podrozdział

3.7). Algorytm ten, będący rozwinięciem algorytmu Mitas, był omówiony na konferencji *Combinatorics 2012* w Perugii [45] i jest również przedstawiony w artykule [44]. Podrozdział 3.8 zawiera opis nowego algorytmu genetycznego dla problemu skoków na posetach przedziałowych (zawarty również w [43]), zaś w podrozdziale 3.9 sformułowano algorytm dokładny wyznaczający liczbę skoków w czasie $\mathcal{O}(1.47^n \cdot \mathrm{poly}(n))$ na posetach przedziałowych. W podrozdziale 3.10 przybliżono bazę trudnych posetów, na których testowano algorytmy, omówioną też w pracy [46].

Czwarty rozdział dotyczy posetów dwuwymiarowych. W podrozdziale 4.1 rozważono dwuwymiarowe posety przedziałowe i zaobserwowano, że istnieje algorytm $\frac{4}{3}$-aproksymacyjny dla liczby skoków w tej klasie posetów. Następnie przetestowano algorytm przeszukiwania z zakazami, zaproponowany w drugim rozdziale rozprawy i w pracy [48].

1. Wiadomości ze złożoności obliczeniowej i z teorii grafów

Przedmiotem badań w tej rozprawie są algorytmy dla problemów sformułowanych w odniesieniu do skończonych obiektów matematycznych. Problemy te zwykle mają charakter optymalizacyjny i często są modelowane za pomocą grafów. Przytaczamy podstawowe fakty ze złożoności obliczeniowej, dotyczące problemów NP-trudnych oraz algorytmów aproksymacyjnych, jak również pojęcia z teorii grafów, wykorzystywane w dalszych rozdziałach.

1.1. Złożoność obliczeniowa, algorytmy aproksymacyjne

W tym podrozdziale precyzujemy pojęcia problemu oraz algorytmu rozwiązującego problem, a także definiujemy złożoność obliczeniową problemu. Następnie przytaczamy pojęcie algorytmu aproksymacyjnego. Definicje tu zawarte pochodzą z literatury poświęconej problemom NP-zupełnym i algorytmom aproksymacyjnym: [30], [74], [17].

Niech Σ oznacza skończony alfabet, czyli skończony zbiór elementów. Słowem nad alfabetem Σ nazywamy skończony ciąg elementów zbioru Σ. Zbiór wszystkich słów nad alfabetem Σ oznaczamy przez $\Sigma^\star$. Językiem nad alfabetem Σ nazywamy dowolny podzbiór zbioru $\Sigma^\star$. Rozmiarem (długością) słowa jest liczba elementów w słowie.

Definicja 1.1.1. Problemem nazywamy podzbiór $P \subseteq I \times S$, gdzie $I \subseteq \Sigma^\star$, $S \subseteq \Sigma^\star$ są językami nad pewnym alfabetem Σ. $I \subseteq \Sigma^\star$ jest zbiorem **instancji** (lub **egzemplarzy**) problemu P, zaś $S \subseteq \Sigma^\star$jest zbiorem **rozwiązań** problemu P.

Definicja 1.1.2. Problem jest **decyzyjny**, jeśli jego rozwiązaniem jest odpowiedź TAK albo NIE.

W celu ścisłego sformalizowania procesów obliczeniowych, towarzyszących rozwiązywaniu problemów, przypominamy pojęcie maszyny Turinga.

Definicja 1.1.3. Niedeterministyczną maszyną Turinga nad alfabetem Λ nazywamy układ

$$M = (\Lambda, \Sigma, B, Q, \delta, q_0, A, R),$$

gdzie Σ jest skończonym alfabetem, takim, że $\Lambda \subseteq \Sigma$, $B \in \Sigma - \Lambda$ jest wyróżnionym symbolem, Q jest zbiorem stanów, rozłącznym z Σ, $q_0 \in Q$ jest stanem początkowym, $A, R \subseteq Q$ są zbiorami odpowiednio stanów akceptujących i stanów odrzucających (których sumę $F = A \cup R$ nazywamy stanami końcowymi), zaś

$$\delta \subseteq (Q - F) \times (\Lambda \cup \{B\}) \times \Sigma^k \times \Sigma^k \times Q \times \{-1, 0, 1\}^{k+1}$$

jest relacją przejścia. Jeśli δ jest funkcją:

$$\delta : ((Q - F) \times (\Lambda \cup \{B\}) \times \Sigma^k) \to (\Sigma^k \times Q \times \{-1, 0, 1\}^{k+1}),$$

to M nazywamy **deterministyczną maszyną Turinga**.

Definicja 1.1.4. Przez **algorytm** A dla problemu $P \subseteq I \times S$ rozumiemy listę instrukcji obliczalnych przez pewną maszynę Turinga. Algorytm A **rozwiązuje** P, jeśli dla dowolnej instancji $x \in I$ maszyna Turinga dla (A, x) na wejściu po skończonej liczbie kroków wyznacza rozwiązanie $y \in S$ takie, że $(x, y) \in I \times S$ albo kończy działanie nie dostarczając rozwiązania, jeśli takie y nie istnieje.

Utożsamiamy języki i problemy. Jeśli istnieje algorytm dla problemu $P \subseteq I \times S$, to zarazem istnieje maszyna Turinga akceptująca język I.

Definicja 1.1.5. Czasem działania (lub **złożonością czasową**) algorytmu A dla problemu $P \subseteq I \times S$ nazywamy funkcję $f : \mathbb{N} \to \mathbb{N}$, gdzie $f(n)$ jest maksymalną liczbą kroków wykonywanych przez maszynę Turinga rozwiązującą A na każdym słowie $x \in I$ długości n.

Definicja 1.1.6. Jeśli f, $g_1, g_2, \ldots, g_n$ są wielomianami rzeczywistymi, to f jest **ograniczona wielomianowo** przez $g_1, g_2, \ldots, g_n$, o ile istnieje rzeczywista funkcja u taka, że $f \leqslant u$ oraz u jest ciągiem złożeń funkcji $g_1, g_2, \ldots, g_n$.

Definicja 1.1.7. Algorytm jest **wielomianowy**, a odpowiadający mu język **wielomianowo rozstrzygalny**, jeśli jego funkcja złożoności czasowej jest ograniczona wielomianowo.

Dla dowolnej funkcji f, przez DTIME$(f(n))$ oznaczamy zbiór problemów decyzyjnych, które można rozwiązać algorytmem deterministycznym o złożoności czasowej

$f(n)$. Dla dowolnej funkcji f, przez NTIME($f(n)$) oznaczamy zbiór problemów decyzyjnych, które można rozwiązać algorytmem niedeterministycznym o złożoności czasowej $f(n)$.

Niech **P** oznacza klasę wszystkich problemów rozwiązywalnych w czasie proporcjonalnym do wielomianu rozmiaru wejścia przez algorytm deterministyczny:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} \mathrm{DTIME}(n^k).$$

Niech **NP** oznacza klasę wszystkich problemów rozwiązywalnych w czasie proporcjonalnym do wielomianu rozmiaru wejścia przez algorytm niedeterministyczny:

$$\mathbf{NP} = \bigcup_{k=0}^{\infty} \mathrm{NTIME}(n^k).$$

Definicja 1.1.8. Niech L_1, L_2 będą językami w **NP**. Język L_1 **redukuje się wielomianowo** do L_2, jeżeli istnieje wielomianowa maszyna Turinga M, która dla wejścia x generuje rozwiązanie y takie, że $x \in L_1$ wtedy i tylko wtedy, gdy $y \in L_2$.

Definicja 1.1.9. Język L jest **NP**-trudny, jeżeli dowolny język $L' \in \mathbf{NP}$ redukuje się wielomianowo do L. Język L jest **NP**-zupełny, jeżeli $L \in \mathbf{NP}$ i L jest **NP**-trudny.

Definicja 1.1.10. Na **problem optymalizacyjny** P składają się:

(a) zbiór instancji D_P, rozpoznawalny w czasie wielomianowym,

(b) zbiór rozwiązań dopuszczalnych $S_P(I)$ dla każdej instancji $I \in D_P$,

(c) obliczalna wielomianowo funkcja celu obj_P, przyporządkowująca każdej parze $(I, s) \in D_P \times S_P(I)$ liczbę wymierną,

(d) informacja, czy P jest problemem maksymalizacji, czy minimalizacji.

Rozwiązaniem optymalnym problemu minimalizacji (maksymalizacji) jest takie rozwiązanie dopuszczalne, dla którego funkcja celu przyjmuje najmniejszą (największą) wartość. Wartość funkcji celu dla rozwiązania optymalnego instancji I oznaczamy przez $OPT_P(I)$.

Z każdym problemem optymalizacyjnym P jest stowarzyszona wersja decyzyjna tego problemu. Wejściem takiego problemu jest para (I, K), gdzie I jest instancją problemu P, zaś K liczbą wymierną. Odpowiedzią jest TAK wtedy i tylko wtedy, gdy

dla I istnieje rozwiązanie dopuszczalne o wartości funkcji celu mniejszej lub równej K (o ile P jest problemem minimalizacji; w przeciwnym wypadku o wartości funkcji celu większej lub równej K) Mówimy, że P jest problemem **NP**-trudnym (**NP**-zupełnym), o ile **NP**-trudny (**NP**-zupełny) jest odpowiadający mu problem decyzyjny.

Definicja 1.1.11. Niech P będzie problemem minimalizacji (odpowiednio: maksymalizacji). Niech $\delta : Z^+ \to Q^+$ spełnia $\delta \geqslant 1$ ($\delta \leqslant 1$). Algorytm A jest **algorytmem δ-aproksymacyjnym** dla P, jeżeli dla dowolnej instancji I problemu P znajduje w czasie wielomianowym rozwiązanie dopuszczalne s spełniające

$$obj_P(I, s) \leqslant \delta(|I|) \cdot OPT(I) \text{ (odp. } obj_P(I, s) \geqslant \delta(|I|) \cdot OPT(I)).$$

1.2. Algorytmy metaheurystyczne

Zaprojektowano kilka klas algorytmów o charakterze stochastycznym, które można stosować do problemów optymalizacyjnych. Przeważnie są one inspirowane procesami obserwowanymi w przyrodzie. Rzadko się jednak zdarza, by dla takich algorytmów dało się udowodnić jakiś współczynnik aproksymacji. Niemniej, algorytmy metaheurystyczne bywają przydatne w praktyce. W sytuacjach, gdy dysponujemy kombinatorycznym algorytmem aproksymacyjnym, można z pomocą metaheurystyk dodatkowo poprawiać jakość znajdowanych rozwiązań. Jeśli z kolei dla jakiegoś problemu nie jest znany algorytm aproksymacyjny, to za pomocą metaheurystyk można znacznie szybciej wygenerować dość dobre rozwiązania, niż procedurami przeszukiwania wyczerpującego.

W tej pracy wykorzystuje się dwie metaheurystyki – algorytm genetyczny [51] i przeszukiwanie z zakazami [31].

W **algorytmie genetycznym** zarządza się populacją osobników (rozwiązań, zwanych też chromosomami) i w każdym kroku algorytmu jest generowana nowa populacja przez wybranie najlepiej dopasowanych osobników z poprzedniej populacji, czyli rozwiązań o największej wartości funkcji celu, gdy problem dotyczy maksymalizacji. Pewne z nich przechodzą mutacje (małe losowe zmiany) i podlegają krzyżowaniu (np. dwóch osobników składa się na dwóch potomków o podobnych fragmentach chromosomu, również w losowy sposób). Oczekuje się, że po pewnej liczbie iteracji wyprodukowane zostanie rozwiązanie bliskie optymalnemu (najlepiej dopasowane).

Aby sprecyzować algorytm genetyczny dla konkretnego problemu, trzeba zdefiniować reprezentację rozwiązań (chromosomy) oraz określić operatory krzyżowania i mutacji, a także schemat wyboru chromosomów do krzyżowania. Funkcja dopasowania z reguły jest określona w sformułowaniu problemu.

Obszernym wprowadzeniem do tematu algorytmów genetycznych jest monografia Michalewicza [51].

Przeszukiwanie z zakazami jest techniką algorytmiczną podążania krok po kroku w kierunku optymalnego rozwiązania problemu obliczeniowego. Jej cechą charakterystyczną jest zarządzanie listą ruchów nie dopuszczonych w danej iteracji, zwaną listą tabu. Ta lista jest wprowadzona po to, by w przebiegu algorytmu unikać powtórnego przeglądania tych samych rozwiązań. W ostatnich latach, przeszukiwanie z zakazami stało się jednym z głównych metaheurystycznych paradygmatów do aproksymacji trudnych problemów obliczeniowych.

W przeszukiwaniu z zakazami, każde rozwiązanie jest traktowane jako punkt w przestrzeni poszukiwań. Rozpoczynamy od rozwiązania sol i w każdym kroku przesuwamy się do innego rozwiązania sol', wybranego z sąsiedztwa sol. Dokładniej, ustalona liczba rozwiązań jest generowana na podstawie sol, a algorytm podąża do najlepszego z nich, pod warunkiem, że nie jest to ruch zakazany (tabu).

Aby sprecyzować algorytm przeszukiwania z zakazami dla konkretnego problemu, definiujemy sąsiedztwo rozwiązania oraz wybieramy sposób kodowania zabronionych ruchów na liście tabu.

Motywacja tej metody została przedstawiona w monografii Glovera i Laguny [31].

Innymi metaheurystykami są m.in. symulowane wyżarzanie [41] i optymalizacja mrówkowa [19].

1.3. Podstawowe pojęcia teorii grafów

W tym podrozdziale przytaczamy podstawowe pojęcia teorii grafów oraz sformułowania problemów najliczniejszego skojarzenia, pakowania podgrafów, zbioru niezależnego, pokrycia zbioru, dla których algorytmy są wykorzystywane przy rozwiązywaniu problemu liczby skoków posetu.

Definicja 1.3.1. Graf nieskierowany, graf prosty lub po prostu **graf** to para $G = (V, E)$, gdzie V i E są zbiorami, $V \cap E = \emptyset$, przy czym E jest zbiorem nieuporządkowanych par elementów zbioru V. Elementy zbioru V nazywamy **wierzchołkami**. Elementy zbioru E nazywamy **krawędziami**. Jeśli dla elementu $e \in E$ mamy $e = \{u, v\}$, to mówimy, że krawędź e jest **incydentna** z wierzchołkami u i v (lub że krawędź e **łączy** wierzchołki u i v), zaś wierzchołki u i v są **wierzchołkami sąsiednimi**.

Jeśli zachodzi potrzeba łączenia pary wierzchołków więcej niż jedną krawędzią, to rozszerzamy definicję 1.3.1 przyjmując, że E jest multizbiorem. Tak określone obiekty nazywamy **multigrafami**. W multigrafach para wierzchołków nie wyznacza jednoznacznie krawędzi, toteż istotne staje się wprowadzenie oznaczeń identyfikujących krawędzie.

Definicja 1.3.2. Niech $G' = (V', E')$ oraz $G = (V, E)$ będą grafami. Jeśli $V' \subseteq V$ i $E' \subseteq E$, to graf G' jest **podgrafem** grafu G, co oznaczamy $G' \subseteq G$. Jeśli $G' \subseteq G$ i G' zawiera wszystkie krawędzie $\{x, y\} \in E$ takie, że $x, y \in V'$, to G' jest **podgrafem indukowanym** grafu G.

Definicja 1.3.3. Ścieżką lub **drogą** w grafie $G = (V, E)$ nazywamy skończony ciąg krawędzi $P = \{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{m-1}, v_m\}$, o ile wszystkie wierzchołki incydentne z tymi krawędziami są parami różne. Jeśli P jest ścieżką i $m \geqslant 2$, to graf złożony z krawędzi $P \cup \{v_m, v_0\}$ nazywamy **cyklem**. Graf G nazywamy **grafem acyklicznym**, jeśli w G nie ma cyklu. Jeśli nie prowadzi to do nieporozumień, ścieżkę (cykl) oznaczamy wskazując incydentne wierzchołki: $v_0 - v_1 - \ldots - v_m$. Liczba m krawędzi w ścieżce (cyklu) jest **długością** ścieżki (cyklu).

Definicja 1.3.4. Graf skierowany (lub **digraf**) to układ $D = (V, A, t, h)$, gdzie V i A są zbiorami, $V \cap A = \emptyset$, zaś $t, h : A \to V$ są odwzorowaniami incydencji. Elementy zbioru V nazywamy **wierzchołkami**, a zbioru A — **łukami**. Jeśli dla elementu $a \in A$ mamy $t(a) = u$ i $h(a) = v$, gdzie $u, v \in V$, to mówimy, że łuk a jest łukiem **incydentnym** z wierzchołkami u i v (a także, że a jest łukiem z wierzchołka u do wierzchołka v), $t(a) = u$ jest **ogonem** łuku a, zaś $h(a) = v$ jest **grotem** łuku a. Przez $\mathrm{indeg}(v)$ oznaczamy liczbę łuków wchodzących do wierzchołka v, zaś przez $\mathrm{outdeg}(v)$ – liczbę łuków wychodzących z wierzchołka v. Łuk a oznaczamy przez $a = (t(a), h(a))$, a czasem także (u, v), chociaż dla łuków wielokrotnych podanie pary wierzchołków incydentnych nie

identyfikuje takich łuków, gdyż w digrafie mogą istnieć wielokrotne łuki z wierzchołka u do wierzchołka v. Łuk $a \in A$ w digrafie $D = (V, A, t, h)$ taki, że $t(a) = h(a) = v$, nazywamy **pętlą** w wierzchołku v.

Definicja 1.3.5. Ścieżką lub **drogą** w digrafie nazywamy ciąg łuków $\pi = (a_1, a_2, \ldots, a_l)$ takich, że $h(a_i) = t(a_{i+1})$ dla $i = 1, 2, \ldots, l-1$, zaś wszystkie wierzchołki incydentne z tymi łukami są parami różne. Liczbę l nazywamy **długością** ścieżki π. Podobnie jak w przypadku pojedynczych łuków, mówimy, że ścieżka π ma **ogon** $t(\pi) = t(a_1)$ oraz **grot** $h(\pi) = h(a_l)$. Jeśli π jest ścieżką o długości przynajmniej 1, to digraf złożony z łuków $\pi \cup (h(a_l), t(a_1))$ nazywamy **cyklem**.

Domknięcie przechodnie digrafu $D = (V, A, t, h)$ oznaczamy przez $\mathrm{tc}(D) = (V, \mathrm{tc}A, t^*, h^*)$, gdzie $b \in \mathrm{tc}A$ wtedy i tylko wtedy, gdy w D istnieje ścieżka $(a_1, a_2, \ldots, a_l)$, $l \geqslant 1$ taka, że $t^*(a_1) = t^*(b)$ i $h^*(a_l) = h^*(b)$.

Problem najliczniejszego skojarzenia

W tym punkcie przytaczamy sformułowanie problemu najliczniejszego skojarzenia w grafie. Pierwszy wielomianowy algorytm dla tego problemu podał Edmonds [21], patrz [71]. W następnym punkcie przytaczamy jedno z uogólnień problemu skojarzenia, które również ma rozwiązanie wielomianowe.

Definicja 1.3.6. Skojarzenie $M \subseteq E$ jest podzbiorem rozłącznych krawędzi grafu $G = (V, E)$, tzn. każdy wierzchołek $v \in V$ jest incydentny z co najwyżej jedną krawędzią z M. **Najliczniejszym skojarzeniem** nazywamy skojarzenie o możliwie największej liczbie krawędzi. Jeśli każdy wierzchołek grafu jest incydentny z jakąś krawędzią M, to M jest **skojarzeniem doskonałym**.

Mamy zatem następujący problem optymalizacyjny:

Problem 1.3.7. Problem znalezienia najliczniejszego skojarzenia w grafie $G = (V, E)$ polega na wyznaczeniu takiego skojarzenia $M_{opt} \subseteq E$, dla którego

$$|M_{opt}| = \max\{|M| : M \text{ jest skojarzeniem w grafie } G\}.$$

Definicja 1.3.8. Dla skojarzenia M mówimy, że krawędź $e \in M$ jest **skojarzona**, natomiast krawędź $e \notin M$ jest **wolna**. Podobnie wierzchołek jest **skojarzony**, jeśli jest incydentny ze skojarzoną krawędzią oraz jest **wolny**, jeśli nie jest incydentny z żadną skojarzoną krawędzią.

Definicja 1.3.9. Ścieżka $P = v_0, e_1, v_1, e_2, \ldots, e_p, v_p$ między dwoma wierzchołkami v_0, v_p jest **ścieżką naprzemienną** względem skojarzenia M, jeśli jej krawędzie są na przemian wolne i skojarzone: $e_1, e_3, \ldots \in E - M$, $e_2, e_4, \ldots \in M$ lub na odwrót. Ścieżka naprzemienna P względem skojarzenia M między dwoma wierzchołkami v_0, v_p jest **ścieżką powiększającą** M, jeśli wierzchołki v_0, v_p są wolne.

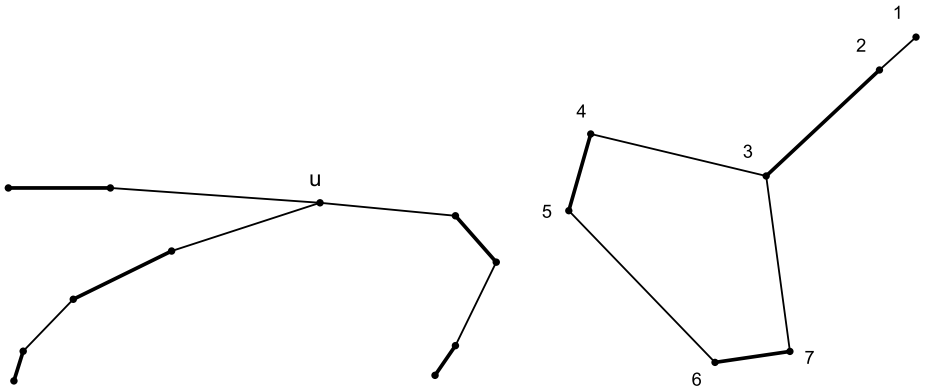
Twierdzenie 1.3.10 (Berge [5])**.** *Skojarzenie M w grafie G jest najliczniejsze wtedy i tylko wtedy, gdy G nie zawiera ścieżki powiększającej M.*

Wobec twierdzenia 1.3.10 można powiększyć skojarzenie M, jeśli znajdziemy ścieżkę powiększającą. Ogólna idea algorytmu znajdującego najliczniejsze skojarzenie jest zatem następująca: należy w danym grafie G ze skojarzeniem M (początkowo pustym) wyszukać ścieżkę P powiększającą M. Następnie tworzymy skojarzenie $M' = M \oplus P$, które ma o jedną krawędź więcej. Znak $\oplus$ oznacza różnicę symetryczną: w nowym skojarzeniu M' krawędzie skojarzone ze ścieżki P stają się wolne, natomiast krawędzie wolne ze ścieżki P stają się skojarzone (rysunek 1.3.11). Powtarzając ten proces, uzyskamy wreszcie skojarzenie, którego nie można powiększyć, a więc będzie to skojarzenie najliczniejsze.

Rysunek 1.3.11. Powiększanie skojarzenia wzdłuż ścieżki powiększającej

W celu znalezienia ścieżki powiększającej stosujemy przeszukiwanie wszerz, rozpoczynając od wierzchołka wolnego. Podczas poszukiwania powstaje drzewo, które nazywamy **drzewem naprzemiennym**, ponieważ krawędzie na każdej ścieżce tego drzewa są na przemian wolne i skojarzone. Poszukiwanie może skończyć się na trzy sposoby. Po pierwsze, algorytm może znaleźć ścieżkę powiększającą. Po drugie może się okazać, że takiej ścieżki nie ma. Uzyskujemy wówczas tzw. **drzewo węgierskie** (rysunek 1.3.12): każda ścieżka wychodząca z wierzchołka u kończy się wierzchołkiem skojarzonym. Trzecią możliwość napotykamy przy próbie dodania do drzewa naprzemiennego

krawędzi, której oba końce już są w tym drzewie. Jeśli domknięty w ten sposób cykl nieparzysty o $2k+1$ krawędziach zawiera k skojarzonych krawędzi, to nazywamy go **kielichem**. W przypadku napotkania kielicha nie można jednoznacznie zidentyfikować ścieżki naprzemiennej. Przykładowy kielich jest pokazany na rysunku 1.3.12. Powiększanie skojarzenia wzdłuż ścieżki $1-2-3-4-5-6-7-3-2-1$ powodowałoby, że krawędzie $\{3,4\}$ oraz $\{3,7\}$ stałyby się skojarzone, co dla incydentnych krawędzi jest niedopuszczalne.



Rysunek 1.3.12. Przykładowe: drzewo węgierskie (po lewej) i kielich (po prawej)

Po znalezieniu ścieżki powiększającej wykorzystujemy ją do powiększenia bieżącego skojarzenia. Jeśli natomiast wykryty zostanie kielich B, to należy wszystkie wierzchołki zawarte w B **ściągnąć** w jeden pseudowierzchołek v_B. Definiujemy wówczas nowy graf $G/B = (V/B, E/B)$, gdzie $V/B = (V-B) \cup \{v_B\}$, a krawędzie określamy następująco:

1. Każda krawędź w G łącząca dwa wierzchołki w $V-B$ pozostaje krawędzią w V/B.
2. Dla krawędzi łączącej $u \in V-B$ z wierzchołkiem $v \in B$ utworzyć krawędź łączącą wierzchołki u, v_B.
3. Skojarzone krawędzie w G pozostają skojarzone w G/B.

Skojarzenie w G/B oznaczamy M/B. Następujące twierdzenie 1.3.13 pozwala kontynuować poszukiwanie najliczniejszego skojarzenia po ściągnięciu kielicha:

Twierdzenie 1.3.13 (Edmonds [21])**.** *Niech B będzie kielichem względem skojarzenia M w grafie G. Skojarzenie M jest najliczniejsze wtedy i tylko wtedy, gdy M/B jest najliczniejszym skojarzeniem w grafie G/B.*

Wobec cytowanych wyżej faktów przebieg algorytmu można streścić następująco: należy szukać ścieżki powiększającej. Jeśli uda się ją znaleźć, powiększamy skojarzenie. Jeśli zamiast tego napotkamy kielich, ściągamy go i postępujemy jak do tej pory, tyle że na nowym grafie. Jeśli w nowym grafie znajdziemy ścieżkę powiększającą zawierającą pseudowierzchołek, to rozwijamy go i włączamy jego fragment do znalezionej ścieżki powiększającej, a następnie powiększamy skojarzenie w wyjściowym grafie. Może się też zdarzyć, że poszukiwanie ścieżki powiększającej w nowym grafie wykryje kolejny kielich. Wówczas ponownie tworzymy pseudowierzchołek i konstruujemy kolejny graf. Kompletny opis algorytmu dostępny jest w literaturze, np. [71, Rozdział 3.7], [4, Chapter 5].

Pakowanie krawędzi i cykli nieparzystych

W tym punkcie omawiamy uogólnienie problemu najliczniejszego skojarzenia. Cytujemy rezultat Cornuéjolsa, Hartvigsena i Pulleyblanka [15] orzekający, iż w wielu sytuacjach problem pakowania krawędzi i podskojarzalnych podgrafów (definicja 1.3.14) w danym grafie ma rozwiązanie wielomianowe. Wynik ten jest o tyle ciekawy, że pakowanie samych trójkątów jest NP-trudne [36]. Algorytm Mitas [52] dla problemu skoków, streszczony w podrozdziale 3.5 i poprawiony w podrodziale 3.7, zawiera poniższy algorytm 1.3.20 jako podprocedurę do rozwiązywania szczególnego przypadku, jakim jest pakowanie krawędzi i danej rodziny cykli nieparzystych wejściowego grafu.

Definicja 1.3.14. Graf $G = (V, E)$ jest **podskojarzalny**, jeśli ma nieparzystą liczbę wierzchołków i dla każdego wierzchołka v, graf $G - v$ ma skojarzenie doskonałe.

Definicja 1.3.15. Niech $G = (V, E)$ będzie grafem, zaś F rodziną podgrafów grafu G. Zbiór $P \subseteq F$ nazywamy **upakowaniem**, jeśli każde dwa grafy w P są wierzchołkowo rozłączne. Mówimy, że wierzchołek G jest **pokrywany** przez upakowanie P, jeśli należy do któregoś grafu w upakowaniu P. Upakowanie pokrywające każdy wierzchołek grafu G nazywamy **upakowaniem doskonałym**.

Problem 1.3.16. Problem F-upakowania polega na znalezieniu upakowania $P \subseteq F$, które pokrywa możliwie największą liczbę wierzchołków G.

Omawiamy rozwiązanie problemu F-upakowania w sytuacji, gdy F składa się ze wszystkich krawędzi grafu G i pewnej rodziny grafów podskojarzalnych. W kontekście tego problemu ważnym pojęciem jest graf P-krytyczny.

Definicja 1.3.17. Niech $G[S]$ będzie podgrafem G indukowanym przez zbiór wierzchołków $S \subseteq V$, zaś F_S zbiorem tych grafów spośród F, których wszystkie wierzchołki należą do S. Podgraf $G[S]$ jest P-krytyczny, jeśli nie ma upakowania doskonałego $P \subseteq F_S$, ale dla każdego $v \in S$, graf $G[S - \{v\}]$ ma upakowanie doskonałe $P \subseteq F_{S-\{v\}}$

Cornuéjols, Hartvigsen i Pulleyblank [15] udowodnili następujące twierdzenia 1.3.18 i 1.3.19.

Twierdzenie 1.3.18. *Każdy P-krytyczny graf jest podskojarzalny.*

Twierdzenie 1.3.19. *Jeśli wierzchołkowo indukowany podskojarzalny podgraf $G[S]$ nie jest P-krytyczny, to ma upakowanie doskonałe krawędzi i dokładnie jednego z podskojarzalnych grafów z F_S.*

Jeśli w instancji problemu F-upakowania F zawiera wszystkie krawędzie G oraz rodzinę grafów podskojarzalnych o rozmiarze wielomianowym względem G, to na mocy twierdzenia 1.3.19 można w czasie wielomianowym wykonać następujące dwie operacje:

- $(O1)$ sprawdzić, czy dany podskojarzalny podgraf $G[S]$ jest P-krytyczny,
- $(O2)$ wygenerować upakowanie doskonałe w podskojarzalnym podgrafie $G[S]$, który nie jest P-krytyczny.

Rozważając graf podskojarzalny $G[S]$ wystarczy bowiem dla każdego grafu podskojarzalnego $T \in F$ sprawdzić, czy $G[S - T]$ ma skojarzenie doskonałe. Jeśli tak, to znaleźliśmy upakowanie doskonałe w $G[S]$, a jeśli to się nie udało dla żadnego T, to znaczy, że $G[S]$ jest P-krytyczny.

Przedstawiony poniżej algorytm 1.3.20 jest modyfikacją algorytmu Edmondsa dla problemu najliczniejszego skojarzenia. W przebiegu algorytmu buduje się las naprzemienny. Wierzchołki lasu naprzemiennego A mogą być dwojakiego typu. Wierzchołek rzeczywisty lasu A jest po prostu wierzchołkiem grafu G. Pseudowierzchołek w A jest wierzchołkowo indukowanym podgrafem G, który jest P-krytyczny. Krawędzie A są krawędziami G, zaś krawędź j jest incydentna z pseudowierzchołkiem, gdy dokładnie jeden koniec j jest w tym pseudowierzchołku. Każde drzewo w lesie jest zakorzenione

w jakimś wierzchołku (który może być wierzchołkiem rzeczywistym lub pseudowierzchołkiem). Wierzchołek w drzewie A jest nieparzysty (parzysty), jeśli liczba krawędzi w ścieżce w A do korzenia jest nieparzysta (parzysta). Nieparzyste wierzchołki A będą zawsze rzeczywistymi wierzchołkami, i będą incydentne z dwiema krawędziami A. Las naprzemienny A jest zawsze zdefiniowany względem upakowania $P \subseteq F$ (które nie jest doskonałe). Korzenie drzew A są dokładnie wierzchołkami, które nie są pokrywane przez P. W każdej ścieżce A zaczynającej się w korzeniu krawędzie muszą być naprzemiennie poza upakowaniem i w upakowaniu P.

Algorytm 1.3.20. Pakowanie krawędzi i grafów podskojarzalnych.

Wejście: Graf G i zbiór F podskojarzalnych podgrafów G.

Wyjście: Upakowanie rozłącznych krawędzi i podgrafów z F o największej liczbie wierzchołków.

0 (inicjalizacja). Niech $P \subseteq F$ będzie dowolnym upakowaniem (np. $P = \emptyset$).

1 (test optymalności). Jeśli P pokrywa każdy wierzchołek, zakończ. W przeciwnym razie niech las A składa się z tych wierzchołków G, które nie są pokrywane przez P. (Czyli A początkowo jest zbiorem korzeni drzew).

2 (wybór krawędzi). Znajdź, jeśli istnieje, krawędź j łączącą parzysty wierzchołek u lasu A z wierzchołkiem v nie będącym nieparzystym wierzchołkiem lasu A. Jeśli nie ma takiego wierzchołka, zakończ. W przeciwnym razie są 4 możliwości.

(a) v nie jest wierzchołkiem lasu A i jest incydentny z krawędzią k w upakowaniu P. Idź do kroku **3**.

(b) v nie jest wierzchołkiem lasu A i jest pokrywany przez graf podskojarzalny H z upakowania P. Idź do kroku **4**.

(c) v jest parzystym wierzchołkiem lasu A w innym drzewie, niż u. Idź do kroku **5**.

(d) v jest parzystym wierzchołkiem lasu A w tym samym drzewie, co u. Idź do kroku **6**.

3 (rozrost lasu). Niech $k = \{v, w\}$. Dodaj do A krawędzie j i k oraz wierzchołki v i w. Zatem v staje się wierzchołkiem nieparzystym lasu A, zaś w staje się wierzchołkiem parzystym. Idź do kroku **2**.

4 (poszerzenie podskojarzalne). Usuń graf H z upakowania P i zastąp go przez krawędzie skojarzenia doskonałego w $H[S - \{v\}]$, gdzie S jest zbiorem wierzchołków H. Idź do kroku **5**.

5 (proste poszerzenie). Dodaj krawędź j do upakowania P. W ścieżce lasu A od wierzchołka u do korzenia, krawędzie, które były w P, usuń z P, podczas gdy te, które nie były w P, zawrzyj w P. Zmień P analogicznie w ścieżce A od v do jego korzenia (o ile ma miejsce przypadek (c)). Po tej zmianie każdy pseudowierzchołek A jest incydentny z dokładnie jedną krawędzią upakowania P. Skoro pseudowierzchołki są podskojarzalne, P można odpowiednio zmodyfikować wewnątrz każdego pseudowierzchołka tak, by wyznaczyć w nim skojarzenie doskonałe. Zlikwiduj las A i wszelkie znalezione do tej pory pseudowierzchołki. Idź do kroku **1**.

6 (krytyczny podgraf). Krawędź j dodana do A tworzy nieparzysty cykl C. Niech H będzie podgrafem G indukowanym przez rzeczywiste wierzchołki cyklu C wraz z wierzchołkami wewnątrz pseudowierzchołków cyklu C. Sprawdź, czy H jest P-krytyczny (operacja $O1$). Jeśli tak, idź do kroku **6a**. W przeciwnym razie idź do kroku **6b**.

6a (ściąganie). Utwórz nowy pseudowierzchołek zawierający H. Teraz ten pseudowierzchołek jest parzystym wierzchołkiem w A. Idź do kroku **2**.

6b (poszerzenie). Znajdź upakowanie doskonałe w H (operacja $O2$). Uzupełnij poszerzenie wzdłuż ścieżki lasu A od grafu H do korzenia tak, jak w kroku **5**. Zlikwiduj las A i wszelkie znalezione do tej pory pseudowierzchołki. Idź do kroku **1**.

Dyskusję dotyczącą poprawności algorytmu można znaleźć w artykule autorów [15], a także w rozprawie doktorskiej Hartvigsena [34, Section 4].

Wielomianowy algorytm dla omawianego tu problemu został równocześnie podany przez Hella i Kirkpatricka [37].

Problemy pokrycia zbioru

Dla kompletności formułujemy kilka ogólnych problemów rozważanych w dziedzinach złożoności obliczeniowej oraz algorytmiki. Są to problemy, do których w pewnych sytuacjach redukuje się problem liczby skoków posetu. W szczególności, w podrozdziale 3.7 wykorzystujemy algorytm aproksymacyjny dla problemu 3-pokrycia zbioru.

Problem 1.3.21. Problem pokrycia zbioru. Dane są: uniwersum $U = \{u_1, u_2, \ldots, u_n\}$ oraz rodzina $\mathcal{F}$ podzbiorów U, taka, że $\sum_{S \in \mathcal{F}} S = U$. Dla każdego podzbioru

$S \in \mathcal{F}$ określony jest nieujemny koszt c_S. Należy wyznaczyć podrodzinę $\mathcal{F}' \subseteq \mathcal{F}$ taką, że $\sum_{S \in \mathcal{F}'} S = U$ i łączny koszt $\sum_{S \in \mathcal{F}'} c_S$ jest możliwie najmniejszy.

Problem 1.3.22. Problem k-pokrycia zbioru zdefiniowany jest dla dowolnej ustalonej liczby naturalnej k jako problem pokrycia zbioru, w którym dla każdego podzbioru $S \subseteq \mathcal{F}$, $|S| \leqslant k$.

Jak wykazał Karp [39], problem k-pokrycia zbioru, dla każdego $k \geqslant 3$, jest NP-trudny.

Twierdzenie 1.3.23 (Duh, Fürer [25])**.** *Istnieje algorytm $\frac{4}{3}$-aproksymacyjny dla problemu 3-pokrycia zbioru.*

Algorytm Duha-Fürera rozpoczyna się od dowolnego pokrycia (na przykład zachłannego). Następnie w każdym kroku usuwa się co najwyżej jedną trójkę z bieżącego pokrycia, a wstawia jedną lub dwie nowe trójki do pokrycia (co nazywamy (2,1)-wymianą), pod warunkiem, że ruch ten poprawia ocenę rozwiązania. W sytuacji, gdy trójki są wybrane, resztę zbioru U pokrywa się dwójkami i jedynkami optymalnie, za pomocą najliczniejszego skojarzenia. Dana (2,1)-wymiana poprawia ocenę rozwiązania, jeśli zmniejsza się koszt pokrycia, a także gdy koszt pokrycia pozostaje bez zmian, a zmniejsza się liczba użytych zbiorów jednoelementowych. Algorytm kończy się, gdy nie ma (2,1)-wymiany poprawiającej bieżące rozwiązanie.

Twierdzenie 1.3.23 jest wykorzystywane w podrozdziale 3.7, gdzie podano nowy współczynnik aproksymacji algorytmu dla problemu skoków na posetach przedziałowych.

Innym ogólnym problemem NP-trudnym jest pytanie o najcięższy zbiór niezależny w danym grafie.

Problem 1.3.24. Załóżmy, że na wierzchołkach grafu $G = (V, E)$ określona jest wymierna, nieujemna funkcja wagowa w. Problem zbioru niezależnego polega na znalezieniu zbioru wierzchołków $S \subseteq V$ takiego, że żadne dwa wierzchołki w S nie są połączone krawędzią, a ich łączna waga $\sum_{v \in S} w(v)$ jest możliwie największa.

2. Problem skoków i podstawowe algorytmy

Rozpoczynamy ten rozdział od zdefiniowania liczby skoków posetu, której obliczanie jest głównym problemem rozważanym w tej rozprawie. Po wprowadzeniu problemu skoków, streszczamy aktualny stan wiedzy o tym problemie. Cytujemy podejście algorytmiczne Sysły, oparte na grafowej reprezentacji posetów. Po pierwsze dlatego, że był to jeden z pierwszych algorytmów, dających się zastosować do dowolnych posetów. Po drugie dlatego, że w oparciu o charakteryzację rozwiązań optymalnych podaną przez Sysłę, zaprojektowano nowy algorytm metaheurystyczny dla problemu skoków. Algorytm ten, będący realizacją przeszukiwania z zakazami, kończy ten rozdział. Jego opis został omówiony w artykule [48].

2.1. Sformułowanie problemu skoków

Wprowadzamy pojęcia, których będziemy używać formułując problem skoków oraz przy omawianiu algorytmów dla tego problemu.

Definicja 2.1.1. Relacją dwuargumentową na zbiorach A i B nazywamy dowolny podzbiór iloczynu kartezjańskiego $R \subseteq A \times B$.

Definicja 2.1.2. Relację dwuargumentową $R \subseteq A \times A$ nazywamy **częściowym porządkiem**, jeżeli są spełnione następujące warunki:

(a) R jest relacją zwrotną: $\forall_{a \in A}\ aRa$,

(b) R jest relacją przechodnią: $\forall_{a,b,c \in A}\ (aRb) \wedge (bRc) \rightarrow (aRc)$,

(c) R jest relacją słabo antysymetryczną: $\forall_{a,b \in A}\ (aRb) \wedge (bRa) \rightarrow (a = b)$.

Jeśli dodatkowo spełniony jest warunek: $\forall_{a,b \in A}\ (aRb) \vee (bRa)$, to relację R nazywamy **porządkiem liniowym**.

Definicja 2.1.3. Zbiorem częściowo uporządkowanym lub **posetem** nazywamy zbiór P wraz z zadaną na nim relacją częściowego porządku $\leqslant_P$ i oznaczamy jako $(P,\leqslant_P)$ lub P. Jeśli relacja $\leqslant_P$ jest porządkiem liniowym, to P nazywamy **zbiorem liniowo uporządkowanym** lub **łańcuchem**.

W tej pracy rozważamy tylko posety skończone. Ponadto pomijamy warunek zwrotności, czyli badamy **ostre porządki częściowe**, oznaczane przez $(P, <_P)$, lub po prostu przez P. To znaczy, $<_P$ jest relacją przechodnią i niezwrotną na zbiorze P. Liczbę elementów posetu P oznaczamy przez $|P| = n$.

Definicja 2.1.4. Mówimy, że dwa elementy p, q posetu $(P, <_P)$ są **porównywalne**, jeśli $p <_P q$ lub $q <_P p$. W przeciwnym wypadku elementy p, q są **nieporównywalne**, co oznaczamy $p \parallel_P q$. Podzbiór C posetu P jest łańcuchem, jeśli każde dwa elementy $p, q \in C$ są porównywalne. Podzbiór A posetu P nazywamy **antyłańcuchem**, jeśli żadne dwa elementy $p, q \in A$ nie są porównywalne.

Dla dowolnego $p \in P$, $Succ_P(p) = \{q \in P : p <_P q\}$ jest **zbiorem następników** p, zaś $Pred_P(P) = \{q \in P : q <_P p\}$ jest **zbiorem poprzedników** p. $\mathcal{S}_P = \{Succ_P(p) : p \in P\}$ jest **rodziną różnych zbiorów następników**, a $\mathcal{P}_P = \{Pred_P(p) : p \in P\}$ jest **rodziną różnych zbiorów poprzedników** posetu P. Mówimy, że p jest **pokrywany** przez q, jeśli $p <_P q$ i dla żadnego $r \notin \{p, q\}$ nie zachodzi $p <_P r <_P q$.

Definicja 2.1.5. Element p posetu $(P, <_P)$ nazywamy **elementem minimalnym** w P, jeśli nie istnieje $q \in P$ taki, że $q <_P p$. Zbiór elementów minimalnych posetu P oznaczamy przez $Min(P)$. Analogicznie, $Max(P) = \{p \in P :$ nie istnieje $q \in P$ taki, że $p <_P q\}$ jest zbiorem **elementów maksymalnych** posetu P.

Definiujemy teraz główny obiekt badań tej rozprawy.

Definicja 2.1.6. Całkowite uporządkowanie $L = p_1, p_2, \ldots, p_n$ elementów zbioru P nazywamy **rozszerzeniem liniowym** posetu $(P, <_P)$, jeśli zachowane są wszystkie relacje, tzn. $p_i <_P p_j$ pociąga za sobą $i < j$. Dwa sąsiadujące elementy p_i, p_{i+1} w L są oddzielone **skokiem**, jeśli $p_i \not<_P p_{i+1}$, a w przeciwnym razie są oddzielone **progiem**. Skoro skoki dzielą L na łańcuchy posetu, możemy napisać: $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$.

Na mocy twierdzenia 2.1.7 każdy (również nieskończony) poset ma rozszerzenie liniowe.

Twierdzenie 2.1.7 (Szpilrajn [72])**.** *Rodzina $\mathcal{L}$ rozszerzeń liniowych posetu P jest niepusta i $P = \bigcap_{L \in \mathcal{L}} L$.*

Co więcej, twierdzenie Szpilrajna implikuje, że każdy poset jest w pełni wyznaczony przez rodzinę jego rozszerzeń liniowych. Ciekawym parametrem posetu jest najmniejsza liczba rozszerzeń liniowych potrzebnych, by go scharakteryzować.

Definicja 2.1.8. Rodzina rozszerzeń liniowych $\{L_1, \ldots, L_d\}$ posetu P nazywa się jego **realizatorem**, jeśli ich przekrojem jest $<_P$. **Wymiar** posetu to najmniejsza moc realizatora $(P, <_P)$. Poset jest d-wymiarowy, jeśli jego wymiar jest równy co najwyżej d.

W tym miejscu formułujemy główny problem rozważany w niniejszej rozprawie.

Problem 2.1.9. Niech $s_L(P)$ oznacza liczbę skoków w rozszerzeniu liniowym L posetu P. Problem liczby skoków polega na znalezieniu

$$s(P) = \min\{s_L(P) : L \text{ jest rozszerzeniem liniowym } P\}.$$

Problem 2.1.9 jest równoważny maksymalizacji liczby progów $b_L(P)$ pośród rozszerzeń liniowych P, ponieważ dla każdego L mamy $s_L(P) + b_L(P) = n - 1$. Oznaczamy przez $b(P)$ największą liczbę progów pośród rozszerzeń liniowych P. Jeśli

$$s_L(P) = s(P) = n - 1 - b_L(P),$$

to L jest nazywane **optymalnym rozszerzeniem liniowym** posetu P.

Definicja 2.1.10. Graf mający P jako zbiór wierzchołków, w którym $\{u, v\}$ jest krawędzią wtedy i tylko wtedy, gdy u i v są porównywalne, $u \neq v$, nazywamy **grafem porównywalności** posetu P. Digraf mający P jako zbiór wierzchołków, w którym (u, v) jest łukiem, gdy u pokrywa v lub v pokrywa u, jest **digrafem pokryć** posetu P. **Diagramem Hassego** posetu P nazywamy digraf pokryć narysowany na płaszczyźnie w taki sposób, że jeśli $p <_P q$, to punkt odpowiadający q jest wyżej niż punkt odpowiadający p. Na takich rysunkach pomijamy groty strzałek.

Klasyczne twierdzenie 2.1.11, sformułowane poniżej, daje ograniczenie dolne na liczbę skoków.

Twierdzenie 2.1.11 (Dilworth [18])**.** *Rozmiar $\omega(P)$ najliczniejszego antyłańcucha w posecie $(P, <_P)$ jest równy najmniejszej liczbie rozłącznych łańcuchów, na które można podzielić P.*

Stąd $s(P) \geqslant \omega(P) - 1$. Ta nierówność motywuje następującą definicję.

Definicja 2.1.12. P nazywamy **posetem Dilwortha**, gdy $s(P) = \omega(P) - 1$.

W tej rozprawie koncentrujemy się na poniższych dwóch klasach posetów.

Definicja 2.1.13. Poset $(P, <_P)$ jest **posetem przedziałowym**, jeśli istnieje bijekcja pomiędzy jego elementami i domkniętymi przedziałami na osi rzeczywistej, $P \leftrightarrow \{I_p = [l(p), r(p)], l(p) \leqslant r(p)\}_{p \in P}$, taka, że $p <_P q$ wtedy i tylko wtedy, gdy $r(p) < l(q)$. Poset $(P, <_P)$ jest **dwuwymiarowy** (lub 2D), jeśli $<_P$ jest przekrojem dwóch rozszerzeń liniowych $\{L_1, L_2\}$, zwanych realizatorem P.

Motywacja do badania problemu skoków

Problem liczby skoków polega na znalezieniu rozszerzenia liniowego danego posetu, minimalizującego liczbę skoków, tj. nieporównywalnych sąsiadujących par. Jedną z motywacji do jego badania jest następujący problem szeregowania. Przypuśćmy, że dany zbiór zadań ma być wykonany przez jedną maszynę, jedno zadanie po drugim, względem pewnych technologicznych ograniczeń kolejnościowych. Każde zadanie przetwarzane bezpośrednio po zadaniu, które nie było jego poprzednikiem, wymaga ponownego przygotowania maszyny (tutaj nazywanego skokiem), co przekłada się na koszt równy jednej jednostce. Celem jest uszeregowanie, które minimalizuje łączny koszt przygotowań maszyny, a więc liczbę skoków.

Czysto teoretyczne zainteresowanie problemem jest związane z faktem udowodnionym przez Habiba [32], że posety o izomorficznych grafach porównywalności mają równą liczbę skoków. Konsekwentnie, liczba skoków wpasowuje się w zakres badań niezmienników porównywalności, razem z wymiarem posetu, liczbą wszystkich rozszerzeń liniowych, najmniejszym podziałem na ścieżki, i innymi właściwościami. Prowadzi to do pytań charakteryzacyjnych o grafy porównywalności spełniające dane własności i o możliwe interpretacje tych niezmienników.

Pokrewnym tematem jest klasyfikacja posetów skokowo krytycznych. $(P, <_P)$ jest skokowo krytyczny, jeśli dla dowolnego $p \in P$, $P - \{p\}$ ma mniej skoków niż P. Pewne rezultaty charakteryzacyjne zostały ustanowione przez El-Zahara i innych [22–24].

2.2. Aktualny stan wiedzy o problemie liczby skoków

Trudność problemu skoków rozważano już na początku lat 80-tych na posetach dwudzielnych, czyli na posetach wysokości 1. Chaty i Chein [11] wykazali, że wyznaczenie liczby skoków dla posetów dwudzielnych jest równoważne ze znalezieniem najliczniejszego skojarzenia na grafie porównywalności, z dodatkowym warunkiem: nie może to być skojarzenie, którego co druga krawędź należy do (tego samego) cyklu. W oparciu o tę równoważność, Pulleyblank [56] udowodnił NP-trudność problemu skoków na posetach dwudzielnych. Następnie Müller [54] uszczegółowił ten rezultat wykazując, że problem skoków pozostaje NP-trudny na posetach dwudzielnych, których graf porównywalności nie zawiera ani jednego cyklu C_k dla żadnego $k \geqslant 6$ jako podgrafu indukowanego; (innymi słowy, każdy cykl długości większej bądź równej 6 ma cięciwę).

Dla kontrastu, Steiner i Stewart [61] podali wielomianowy algorytm do obliczania liczby skoków posetu dwudzielnego, jeśli zarazem jest on dwuwymiarowy. W grafie dwudzielnym $G = (X \cup Y, E)$ uporządkowanie klasy wierzchołków X jest wypukłe, jeśli dla każdego wierzchołka $y \in Y$, wszystkie wierzchołki sąsiednie do y występują kolejno w uporządkowaniu X. Graf dwudzielny jest wypukły, jeśli jedna klasa wierzchołków ma takie uporządkowanie, oraz biwypukły, gdy obie klasy mają takie uporządkowania. Wielomianowe algorytmy dla szerszych klas posetów dwudzielnych podawali kolejno: Brandstädt [8] (gdy graf porównywalności jest biwypukły), Dahlhaus [16] (gdy graf porównywalności jest wypukły), a ostatnio Soto i Telha [62] (gdy wierzchołki grafu porównywalności są dwupokolorowane, leżą na płaszczyźnie i krawędź kładziemy między a i b, gdy ich rzuty na osie spełniają $a_x \leqslant b_x$, $a_y \leqslant b_y$ oraz a i b mają różne kolory).

Algorytmy wielomianowe zaprojektowano też dla kilku innych klas posetów. W szczególności są to posety N-wolne (Rival [57], Sysło [65]) oraz podklasa porządków przedziałowych, w której wszystkie elementy posetu są reprezentowane przedziałami jednej długości (Arnim i Higuera [2]). Colbourn i Pulleyblank [14] podali wielomianowy algorytm dla posetów o stałej szerokości.

Mitas [52] wykazała, że problem skoków jest NP-trudny w klasie posetów przedziałowych. Niemniej, dla tej klasy posetów zaprojektowano aż trzy różne wielomianowe algorytmy aproksymacyjne. Podejścia Felsnera [27] i Sysły [69] stanowią realizację strategii zachłannej. Algorytm Mitas opiera się na redukcji do problemu pakowania podgrafów. Wszystkie trzy algorytmy mają stały współczynnik aproksymacji, równy $\frac{3}{2}$. Autor

rozprawy poprawił [44] współczynnik aproksymacji w tej klasie posetów do 1.484. Omawiamy ten rezultat w podrozdziale 3.7, po wprowadzeniu pracy Mitas.

Problem skoków był też badany z punktu widzenia algorytmów dokładnych. Liczbę skoków dowolnego posetu można wyznaczyć algorytmem przeszukiwania wyczerpującego, podanym przez Sysłę [68], omówionym w podrozdziale 2.3. Pesymistyczna złożoność czasowa szacowana jest przez funkcję silnia od liczby łuków pozornych w diagramie reprezentującym poset. Jeśli pytamy, czy $s(P) \leqslant k$, gdzie k jest stałą, to odpowiedź uzyskamy w czasie $\mathcal{O}(k!k^2n)$ algorytmem McCartin [50]. Ostatnio Kratsch i Kratsch [42] podali algorytm dokładny działający w czasie $\mathcal{O}(1.8638^n)$, a przy ograniczeniu do posetów przedziałowych otrzymali złożoność $\mathcal{O}(1.7593^n)$. Pokazujemy w podrozdziale 3.9, że na posetach przedziałowych można otrzymać algorytm o niższej złożoności, stosując technikę rozgałęziania.

Pomimo, że na posetach wysokości 1 problem skoków jest NP-trudny, Sysło, Koh i Chia [66] podali wielomianowy algorytm rozstrzygający, czy dany dwudzielny poset jest posetem Dilwortha. Z tym wynikiem kontrastuje rezultat Bouchitté i Habiba [6], że rozpoznawanie posetów Dilwortha wysokości 2 jest NP-trudne. Duffus, Rival i Winkler [20] wykazali, że posety bezcyklowe są posetami Dilwortha.

Otwartym problemem pozostaje złożoność problemu skoków, lub choćby problemu rozpoznawania posetów Dilwortha, w klasie posetów dwuwymiarowych. Ceroi [9] sformułował dowód NP-trudności w rozszerzonym przypadku, gdy na porównywalnościach określona jest całkowitoliczbowa funkcja wagowa, a celem jest maksymalizacja sumy wag na progach rozszerzenia liniowego. Nie podano też do tej pory algorytmu aproksymacyjnego dla posetów dwuwymiarowych. W podrozdziale 4.1 pokazujemy, jak aproksymacja liczby skoków na posetach przedziałowych przenosi się na posety przedziałowe, będące zarazem dwuwymiarowymi.

Algorytm genetyczny dla liczby skoków

Jedną z najpopularniejszych metaheurystyk – algorytm genetyczny (naszkicowany w podrozdziale 1.2) – zastosował do rozwiązywania problemu liczby skoków Ngom [55].

W adaptacji dla problemu liczby skoków, odpowiedniej dla dowolnych posetów, rozwiązanie jest reprezentowane jako permutacja L elementów posetu. Funkcja dopa-

sowania bierze pod uwagę następujące trzy wartości:

- **liniowość** permutacji L względem posetu P, tj. łączną liczbę porównywalności w L, które jednocześnie są porównywalnościami w P,
- łączną liczbę porównywalności w P,
- liczbę progów w L, tj. $b_L(P)$,

przy czym $b_L(P)$ uwzględnia się dopiero, gdy liniowość permutacji jest równa liczbie wszystkich porównywalności posetu (czyli gdy L jest poprawnym rozszerzeniem liniowym).

Mutacja losuje dla pozycji i w L inną pozycję j i zamienia miejscami dwa elementy na tych pozycjach, z prawdopodobieństwem malejącym, gdy $|i - j|$ rośnie. Operator krzyżowania dla dwóch osobników L_1 i L_2, generuje losową maskę binarną M długości n i kopiuje elementy na kolejnych pozycjach $i = 1, \ldots, n$ z L_1 i L_2 do dwóch permutacji C_1 i C_2 odpowiednio, zgodnie z maską binarną (tj. jeśli $M[i] = 0$, to $C_1[i] := L_1[i]$, w przeciwnym razie $C_2[i] := L_2[i]$). Następnie puste miejsca w C_1 i w C_2 są wypełniane brakującymi elementami posetu, wziętymi odpowiednio z L_2 (dla C_1) i z L_1 (dla C_2), w tej samej kolejności, w której występują w osobnikach źródłowych.

Redukcja problemu skoków do TSP

Zauważmy, że problem skoków można zredukować do problemu komiwojażera.

Twierdzenie 2.2.1. *Dla dowolnego posetu P można wyznaczyć liczbę skoków w czasie $\mathcal{O}(2^n \cdot \mathrm{poly}(n))$.*

Dowód. Z posetem P stowarzyszamy instancję problemu komiwojażera, w której wierzchołki odpowiadają punktom P, zaś odległości są następujące:

- jeśli $p <_P q$, to $c_{pq} = 0$,
- jeśli $p \not<_P q$, to $c_{pq} = 1$,
- w przeciwnym razie $c_{pq} = \infty$.

Ponadto dokładamy wierzchołek d taki, że odległości od d do elementów minimalnych posetu są równe 0 i odległości od elementów maksymalnych do d są równe 0. Wówczas każdemu rozszerzeniu liniowemu L odpowiada cykl Hamiltona od d do d. Łączny koszt każdego takiego cyklu jest równy $s_L(P)$, zaś każda pozostała permutacja wierzchołków zawiera połączenie o koszcie ∞. Można wobec tego zastosować algorytm programowania dynamicznego Helda i Karpa [35] dla problemu komiwojażera, by policzyć $s(P)$ w czasie $\mathcal{O}(2^n \cdot \text{poly}(n))$. □

Zachłanne rozszerzenia liniowe

Istotną rolę w projektowaniu algorytmów dla problemu skoków odegrały zachłanne rozszerzenia liniowe, zdefiniowane w tym punkcie.

Definicja 2.2.2. Łańcuch C w posecie P jest **zachłanny**, jeśli $Pred_P(p) \cup \{p\} = C$, gdzie $p = \sup C$, i nie ma elementu q pokrywającego p takiego, że łańcuch $C \cup \{q\}$ miałby tę własność. Rozszerzenie liniowe $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$ nazywamy **zachłannym rozszerzeniem liniowym**, jeśli C_i jest zachłannym łańcuchem w $P - \cup_{j<i} C_j$.

Prosty algorytm zachłanny przytoczony jako algorytm 2.2.4 nie ma żadnej gwarancji aproksymacji. Można go stosować do dowolnych posetów. W podrozdziale 2.3 prezentujemy algorytm Sysły, eksploatujący pewne szczególne łańcuchy zachłanne, a w dalszej części pracy omawiamy specjalizacje tego algorytmu dla posetów przedziałowych (podrozdziały 3.2 i 3.3). Motywacją do badań strategii zachłannej jest twierdzenie 2.2.3.

Twierdzenie 2.2.3. *Każdy poset ma optymalne rozszerzenie liniowe, które jest zachłanne.*

Dowód znajduje się w artykule Sysły [67].

Algorytm 2.2.4. Algorytm zachłanny dla problemu liczby skoków.

Wejście: Poset $(P, <_P)$

Wyjście: Zachłanne rozszerzenie liniowe L posetu P.

1. L jest pustą listą, do której dodajemy kolejne elementy generowanego rozszerzenia liniowego. H jest aktualizowanym co krok zbiorem następników poprzednio dodanego elementu, należących zarazem do $Min(P)$; początkowo $H = \emptyset$.

2. Powtarzaj, dopóki $P \neq \emptyset$:

(a) Jeśli $H = \emptyset$, to wybierz dowolny $p \in Min(P)$. W przeciwnym razie wybierz dowolny $p \in H$.

(b) Dodaj wybrany element p na koniec listy L.

(c) Usuń element p z posetu P.

(d) $H := Succ_P(p) \cap Min(P)$.

3. Zwróć rozszerzenie liniowe L.

Istnieje nietrywialna klasa posetów, dla których algorytm 2.2.4 gwarantuje wygenerowanie optymalnego rozszerzenia liniowego. Niech N oznacza następujący poset: $N = \{a, b, c, d\}, a < c, b < c, b < d$ i nie ma innych porównywalności. Poset P nazywa się **N-wolny**, jeśli nie zawiera czterech elementów, pomiędzy którymi są dokładnie te porównywalności (i nie ma innych).

Twierdzenie 2.2.5 (Rival)**.** *Jeśli $(P, <_P)$ jest posetem N-wolnym, to dowolne zachłanne rozszerzenie liniowe jest optymalne.*

Twierdzenie 2.2.5 jako pierwszy udowodnił Rival [57]. Później Sysło [65] podał alternatywny dowód, wykazując, że jego algorytm, prezentowany w podrozdziale 2.3 (algorytm 2.3.12) zawsze generuje rozwiązanie optymalne dla posetów N-wolnych.

2.3. Algorytm Sysły

Omawiamy teraz podejście Sysły do problemu skoków. Posety są tutaj reprezentowane specjalnymi digrafami, zdefiniowanymi na początku tego podrozdziału (definicja 2.3.1). Algorytm jest realizacją strategii zachłannej, i jednocześnie próbą przeniesienia własności optymalnych rozszerzeń liniowych z posetów N-wolnych na dowolne posety. W porównaniu ze zwykłym algorytmem zachłannym (algorytm 2.2.4), przestrzeń poszukiwań jest znacznie ograniczona.

Diagram łukowy posetu, ścieżki zachłanne

Rozpoczynamy od zdefiniowania diagramów łukowych, używanych do reprezentacji posetów w omawianych dalej algorytmach.

Definicja 2.3.1. Digraf acykliczny $D(P) = (V, R, t, h)$ wraz z odwzorowaniem $\phi : P \to R$ nazywamy **diagramem łukowym** posetu $(P, <_P)$, jeśli dla dowolnych $p, q \in P$, $p \neq q$, mamy

$$p <_P q \text{ wtedy i tylko wtedy, gdy } (h^*(\phi(p)), t^*(\phi(q))) \in R^*,$$

gdzie t^*, h^* są odwzorowaniami incydencji w $\mathrm{tc}(D(P))$, zaś $R^* = \mathrm{tc}(R) \cup \{(v, v) : v \in V\}$. Łuki $a \in \phi(P)$ nazywamy **łukami posetu**, natomiast pozostałe łuki to **łuki pozorne**.

Nieformalnie, pewne łuki (należące do $\phi(P)$) reprezentują elementy posetu P, zaś pozostałe łuki dodaje się w celu zachowania porównywalności wzdłuż ścieżek diagramu $D(P)$.

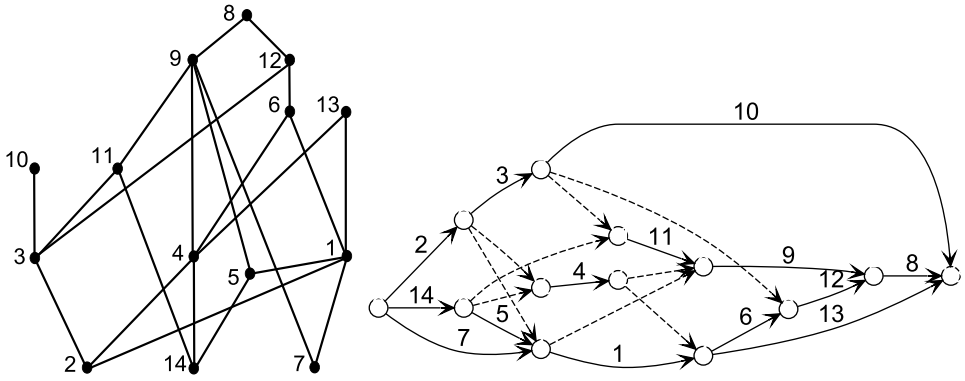
Definicja 2.3.2. Diagram łukowy $D(P) = ((V, R, t, h); \phi)$ jest **diagramem zwartym**, jeśli spełnia następujące warunki:

(a) $D(P)$ ma dokładnie jedno źródło i dokładnie jedno ujście,

(b) dla każdego łuku pozornego a, $\mathrm{indeg}(h(a)) > 1$ oraz $\mathrm{outdeg}(t(a)) > 1$.

Ogólny algorytm budowy zwartego diagramu łukowego podał Sysło [63]. Procedura ta sformułowana jest jako algorytm 2.3.9.

Przykład diagramu łukowego jest pokazany na rysunku 2.3.3. Elementy 2, 3, 12, 8 stanowią jeden z łańcuchów w tym posecie, więc cztery odpowiadające łuki są ułożone w jedną ze ścieżek prowadzących ze źródła do ujścia przedstawionego diagramu. W dowolnym rozszerzeniu liniowym te (i inne) ograniczenia posetowe muszą być respektowane, np. $(2, 3, 10) \oplus (7) \oplus (14, 5, 1) \oplus (4, 13) \oplus (6, 12) \oplus (11, 9, 8)$.



Rysunek 2.3.3. Diagram Hassego oraz zwarty diagram łukowy posetu

Definicja 2.3.4. W diagramie łukowym $D(P)$, naturalnym odpowiednikiem zachłannego łańcucha C (definicja 2.2.2) jest **ścieżka zachłanna** $\pi(C) = (a_1, a_2, \ldots, a_l)$, $l \geqslant 1$, tj. ścieżka spełniająca następujące warunki:

(a) Żaden wierzchołek $\pi(C)$ za wyjątkiem $h(a_l)$ nie jest grotem łuku innego niż a_j, dla każdego $j = 0, 1, \ldots, l-1$.

(b) a_j jest łukiem posetu, dla każdego $j = 1, \ldots, l$.

(c) $\pi(C)$ nie można rozszerzyć do dłuższej ścieżki spełniającej powyższe dwa warunki.

W przeciwną stronę, każda ścieżka zachłanna π indukuje zachłanny łańcuch C_π w $D(P)$. Zatem algorytm generujący zachłanne rozszerzenie liniowe można sformułować w terminach diagramu łukowego dla P, czyli jako algorytm 2.3.5.

Algorytm 2.3.5. Zachłanne rozszerzenie liniowe posetu w postaci diagramu łukowego.

Wejście: $D(P)$, diagram łukowy dla posetu P.

Wyjście: $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$, zachłanne rozszerzenie liniowe posetu P.

1. { Inicjalizacja }

$D := D(P)$

$L := \emptyset$

2. **while** $D \neq \emptyset$

($\star$) znajdź ścieżkę zachłanną π w D

$L := L \oplus C_\pi$

$D := D(P - L)$, diagram łukowy dla pozostałego posetu

3. **return** L

Definiujemy teraz specjalne ścieżki zachłanne, do których ograniczają się prezentowane dalej algorytmy.

Definicja 2.3.6. Ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$ w diagramie łukowym $D(P)$ jest **ścieżką silnie zachłanną**, jeśli spełnia dodatkowo jeden z warunków:

(a) albo $h(\pi)$ jest ujściem $D(P)$, albo

(b) $h(\pi)$ jest grotem łuku posetu $b \neq a_l$ takiego, że żadna ścieżka zakończona łukiem b nie zawiera wierzchołka incydentnego z łukiem pozornym.

Definicja 2.3.7. Ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$ w diagramie łukowym $D(P)$ jest **ścieżką półsilnie zachłanną**, jeśli nie jest ścieżką silnie zachłanną i przechodzi przez wierzchołek będący ogonem łuku pozornego, ale nie będący grotem łuku pozornego.

Przykład 2.3.8. Diagram na rysunku 2.3.3 ma dwie silnie zachłanne ścieżki. Ścieżka $(2, 3, 10)$ przechodzi przez ogon łuku pozornego, więc klasyfikowałaby się jako półsilnie zachłanna, ale dodatkowo kończy się ona w ujściu, więc jest silnie zachłanna. Ponadto, wierzchołek kończący ścieżkę $(14, 5)$ kończy również ścieżkę (7), która nie ma wierzchołka incydentnego z łukiem pozornym. Toteż $(14, 5)$ jest silnie zachłanna.

Konstrukcja diagramów łukowych

Cytujemy teraz algorytm budowy diagramu łukowego dla dowolnego skończonego posetu. Algortym podał Sysło [63].

Algorytm 2.3.9. Uniwersalna konstrukcja diagramu łukowego.

Wejście: Poset P.

Wyjście: Zwarty diagram łukowy reprezentujący poset P.

1. Przetwarzanie wstępne posetu P:

(a) Wygeneruj różne zbiory poprzedników $Pred_1, Pred_2, \ldots, Pred_k$.

(b) Wygeneruj różne zbiory następników $Succ_1, Succ_2, \ldots, Succ_l$.

(c) Dla każdego $Pred_i$ wyznacz zbiór $U_i = \bigcap_{p \in Pred_i} Succ(p)$.

2. Określ zbiór wierzchołków diagramu:

(a) $x_1, x_2, \ldots, x_h$ odpowiadają tym $Pred_i$, dla których istnieje $Succ_j = U_i$.

(b) $y_{h+1}, y_{h+2}, \ldots, y_k$ odpowiadają pozostałym zbiorom $Pred_i$.

(c) $z_{h+1}, z_{h+2}, \ldots, z_l$ odpowiadają pozostałym zbiorom $Succ_j$.

Niech $y_i = z_i = x_i$ dla $i = 1, 2, \ldots, h$.

3. Określ zbiór łuków diagramu:

(a) Dla każdego $p \in P$ dodaj łuk posetu (y_i, z_j), gdzie $P_i = Pred(p)$, $Succ_j = Succ(p)$.

(b) Dla każdych $p, q \in P$, $p \neq q$, jeśli p jest pokrywany przez q, zaś $z_j \neq y_i$, gdzie $Succ_j = Succ(p)$, $Pred_i = Pred(q)$, dodaj łuk pozorny (z_j, y_i).

(c) Usuń łuki przechodnie, pod warunkiem, że nie są łukami posetu.

Półsilnie zachłanne rozszerzenia liniowe

Na podstawie twierdzenia 2.2.3 można zaprojektować procedurę przeliczającą wszystkie zachłanne rozszerzenia liniowe, by wyznaczać w ten sposób liczbę skoków posetu. Eksperymenty wykazują, że jest to bardzo czasochłonny i niewydajny proces. Niemniej, zostało udowodnione w serii prac [63, 65, 67, 68, 70], że przestrzeń poszukiwań istotnie się redukuje, skoro dla każdego posetu klasa zachłannych rozszerzeń liniowych może być dalej ograniczona do klasy bardzo szczególnych zachłannych rozszerzeń liniowych, zawierającej optymalne rozwiązanie. Te rozszerzenia liniowe są złożone z dwóch specjalnych typów zachłannych łańcuchów, opisanych powyżej w definicjach 2.3.6 i 2.3.7.

Twierdzenie 2.3.10 (Sysło [67])**.** *Niech P będzie posetem, zaś $D(P)$ jego zwartym diagramem łukowym. Następujące stwierdzenia są prawdziwe:*

(a) *jeśli π jest silnie zachłanną ścieżką w $D(P)$, to istnieje optymalne rozszerzenie liniowe posetu P rozpoczynające się łańcuchem C_π i $s(P) = s(P - C_\pi) + 1$,*

(b) *jeśli diagram łukowy $D(P)$ posetu P nie zawiera ścieżki silnie zachłannej, to zawiera ścieżkę półsilnie zachłanną,*

(c) *jeśli diagram łukowy $D(P)$ nie zawiera ścieżki silnie zachłannej, to poset P ma optymalne rozszerzenie liniowe rozpoczynające się jakąś ścieżką półsilnie zachłanną.*

Wniosek 2.3.11. *Każdy poset ma optymalne rozszerzenie liniowe $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$, zwane półsilnie zachłannym, takie, że każdy łańcuch C_i jest silnie zachłanny w $P_i = P - \cup_{j<i} C_j$ lub półsilnie zachłanny w P_i, jeśli P_i nie ma silnie zachłannych łańcuchów.*

Aby zaprojektować algorytm generujący jedno półsilnie zachłanne rozszerzenie liniowe, po prostu zastępujemy krok **2**. ($\star$) w algorytmie 2.3.5 wyszukaniem silnie zachłannej ścieżki, a jeśli jej nie ma, to półsilnie zachłannej ścieżki w diagramie łukowym.

Algorytm 2.3.12. Półsilnie zachłanne rozszerzenie liniowe.

... (jak w algortymie 2.3.5) ...

2. **while** $D \neq \emptyset$

($\star$) znajdź silnie zachłanną ścieżkę π w D; jeśli nie znaleziono takiej ścieżki, to wybierz π jako dowolną ścieżkę półsilnie zachłanną w D.

... (jak w algortymie 2.3.5) ...

Możemy teraz również podać dokładny algorytm dla liczby skoków, który poszukuje optymalnego rozwiązania pomiędzy wszystkimi półsilnie zachłannymi rozszerzeniami liniowymi za pomocą przeszukiwania z nawrotami. W tym celu w kroku **2**. ($\star$) algorytmu 2.3.12, jeśli nie ma ścieżek silnie zachłannych, to zamiast wybrać dowolną półsilnie zachłanną ścieżkę, weryfikujemy każdą z nich, i stosujemy procedurę rekurencyjnie na każdym odpowiadającym podposecie. Odnosimy się do tego algorytmu jako `OptLinExt` w dalszej części pracy, gdzie m.in. proponowany jest nowy algorytm przeszukiwania z zakazami, który również ogranicza poszukiwania do tych specjalnych rozszerzeń liniowych. Algorytm `OptLinExt` sformułowany jest w postaci algorytmu 2.3.13.

Algorytm 2.3.13. Algorytm dokładny Sysły dla problemu skoków.

---

optLinExt(D)

Wejście: Zwarty diagram łukowy D posetu P.

Wyjście: Optymalne rozszerzenie liniowe L posetu P.

1. Wyznacz ścieżki silnie i półsilnie zachłanne w diagramie D.

(a) Jeśli znaleziono ścieżkę silnie zachłanną, to usuń ją z diagramu, dodając elementy tej ścieżki do rozszerzenia liniowego L. Jeśli zredukowany diagram D jest pusty, to zakończ — L jest optymalnym rozszerzeniem liniowym P. W przeciwnym razie powtórz krok **1**.

(b) Jeśli diagram D nie zawiera ścieżki silnie zachłannej, to dla każdej ścieżki półsilnie zachłannej π utwórz kopię D_π diagramu D oraz kopię L_π rozszerzenia L, a następnie wykonaj $subLinExt(D_\pi, \pi, L_\pi)$.

2. Zwróć rozszerzenie liniowe o najmniejszej liczbie skoków, znalezione w którymś z wywołań *subLinExt*.

subLinExt(D, π, L)

Wejście: Zwarty diagram łukowy D, z którego być może usunięto pewne ścieżki zachłanne, składające się na częściowe rozszerzenie liniowe L, a także ścieżka półsilnie zachłanna π w diagramie D.

1. Usuń ścieżkę półsilnie zachłanną π z diagramu D, powiększając rozszerzenie liniowe L o elementy z tej ścieżki.

(a) Jeśli diagram D jest pusty, to sprawdź, czy rozszerzenie L ma mniej skoków niż najlepsze znalezione do tej pory i jeśli tak, zachowaj je. Koniec.

(b) Jeśli diagram D po wyeliminowaniu π nie jest pusty, to idź do kroku **2**.

2. Wyznacz ścieżki silnie i półsilnie zachłanne w diagramie D.

(a) Jeśli znaleziono ścieżkę silnie zachłanną, to usuń ją z diagramu, dodając elementy tej ścieżki do rozszerzenia liniowego L. Jeśli zredukowany diagram D jest pusty, to sprawdź, czy wygenerowane rozszerzenie liniowe ma mniej skoków niż najlepsze znalezione do tej pory i jeśli tak, zachowaj je. Koniec. W razie, gdy diagram D nie jest pusty, powtórz krok **2**.

(b) Jeśli diagram D nie zawiera ścieżki silnie zachłannej, to dla każdej ścieżki półsilnie zachłannej π utwórz kopię D_π diagramu D oraz kopię L_π rozszerzenia L, a następnie wykonaj $subLinExt(D_\pi, \pi, L_\pi)$.

Aby algorytm był kompletny, musimy umieć „naprawić" diagram łukowy po usunięciu ścieżki zachłannej. Najprostszym sposobem jest zbudowanie diagramu od nowa dla pozostałej części posetu. Druga możliwość to implementacja algorytmu 2.3.14.

Algorytm 2.3.14. Usunięcie ścieżki zachłannej z diagramu łukowego.

Wejście: Zwarty diagram łukowy D posetu P, ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$.

Wyjście: Zwarty diagram łukowy pozbawiony elementów ścieżki π.

1. Usuń wszystkie łuki $a_1, a_2, \ldots, a_l$ z diagramu D.

2. Usuń wszystkie łuki pozorne, których ogony są wierzchołkami ścieżki π, za wyjątkiem $h(\pi)$.

3. Usuń wszystkie łuki pozorne, których ogonem jest $h(\pi)$, pod warunkiem, że nie istnieją łuki wchodzące do $h(\pi)$.

4. Usuń wszystkie wierzchołki izolowane, powstałe w krokach **1-3**.

5. Zespól wszystkie wierzchołki o stopniu wchodzącym równym 0 w jeden wierzchołek (nowe źródło diagramu).

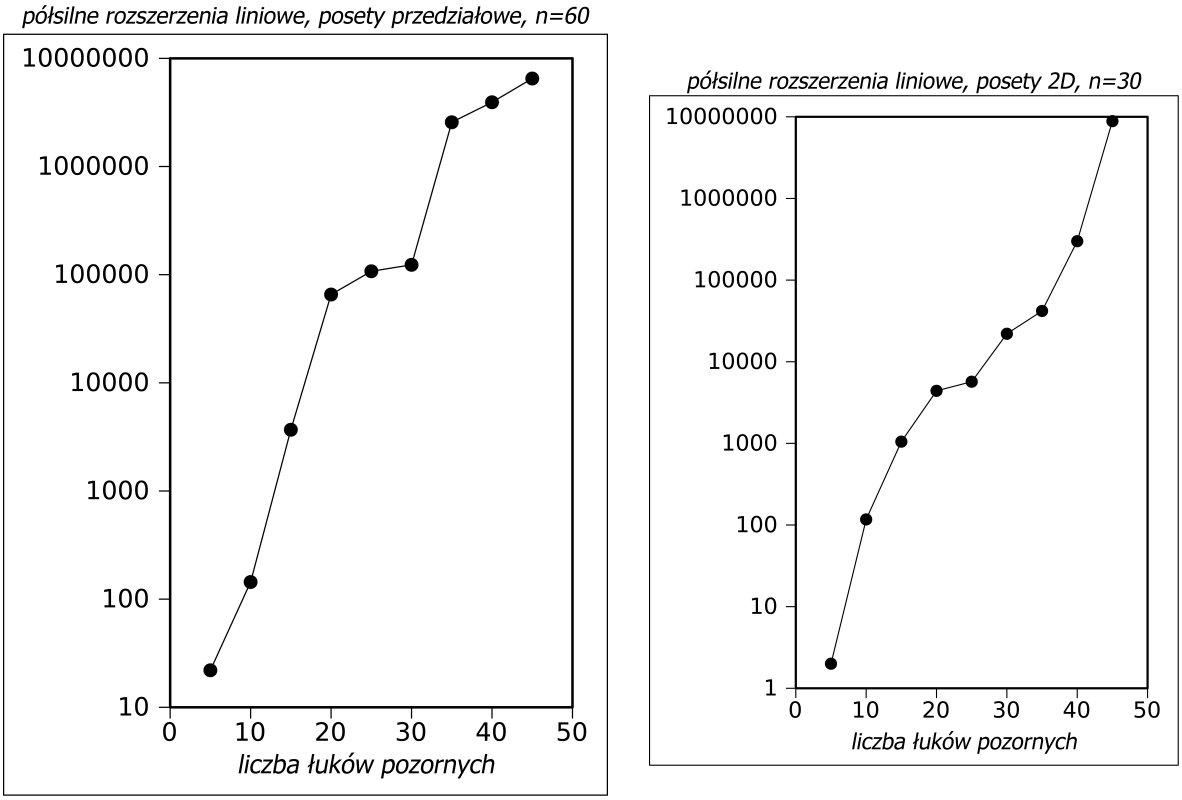
6. Usuń lub sklej nadmiarowe łuki pozorne. (Wierzchołki łuku a można skleić, gdy $t(a)$ nie jest ogonem innego łuku niż a lub $h(a)$ nie jest grotem innego łuku niż a). Łuk a można usunąć, gdy po jego usunięciu diagram w dalszym ciągu reprezentuje ten sam poset).

`OptLinExt` – doświadczenia obliczeniowe

Oznaczmy przez d liczbę łuków pozornych w $D(P)$. Jak uzasadniono w [68], pesymistyczna złożoność czasowa algorytmu `OptLinExt` jest rzędu $\mathcal{O}(d! \cdot \text{poly}(n, d))$, skoro

jest co najwyżej $d!$ półsilnie zachłannych rozszerzeń liniowych. To znaczy, d jest ważnym czynnikiem składającym się na złożoność problemu.

Przeprowadzono eksperyment, by sprawdzić, jak wiele rozwiązań jest generowanych w rzeczywistości na posetach o różnej liczbie łuków pozornych. Dla ustalonego rozmiaru posetu ($n = 120$ w przypadku posetów przedziałowych i $n = 30$ w przypadku posetów dwuwymiarowych) i ustalonej liczby łuków pozornych ($d \in \{5, 10, \ldots, 45\}$) wylosowana została setka posetów. Aby uzyskać posety o zadanej liczbie łuków pozornych w ich diagramach łukowych, użyto algorytmu genetycznego (którego założenia opisano w podrozdziale 1.2), gdzie odległość od zadanego d była czynnikiem optymalności. Największa liczba wygenerowanych rozszerzeń liniowych pośród grupy posetów o d łukach pozornych była zapisywana. Jest to pokazane na rysunku 2.3.15.



Rysunek 2.3.15. Liczba półsilnie zachłannych rozszerzeń liniowych w próbce posetów o rosnącej liczbie łuków pozornych

Te doświadczenia pomagają ocenić rząd wielkości przestrzeni poszukiwań, przeglądanej przez algorytm przeszukiwania z zakazami opisany w podrozdziale 2.4. Warto zauważyć, że przestrzeń poszukiwań jest znacznie większa w przypadku posetów dwu-

wymiarowych niż w przypadku posetów przedziałowych.

W grupie posetów przedziałowych na 120 elementach, zawierających 45 łuków pozornych, napotkany został poset mający 6 510 338 półsilnie zachłannych rozszerzeń liniowych, i algorytm dokładny potrzebował 5 godzin, by je wypisać. Na ogół liczba rozwiązań jest znacznie mniejsza. Z drugiej strony, znaleziony został 30-elementowy poset dwuwymiarowy, o 45 łukach pozornych w swoim diagramie, dla którego przestrzeń poszukiwań zawiera 8 857 068 półsilnie zachłannych rozszerzeń liniowych. W tym przypadku, wykonanie `OptLinExt` trwało 3 445 sekund, dzięki krótkiemu czasowi przebudowywania diagramów dla nie więcej niż 30 elementów.

Zauważamy, że liczba łuków pozornych jest mniejsza niż n na posetach przedziałowych. Tak nie jest w przypadku posetów dwuwymiarowych. W związku z tym posety dwuwymiarowe często mają istotnie większą przestrzeń półsilnie zachłnnych rozszerzeń liniowych niż posety przedziałowe.

2.4. Adaptacja przeszukiwania z zakazami

Genetyczna heurystyka Ngoma [55], omówiona w podrozdziale 2.2, została zaimplementowana i przetestowana. Okazało się, że nie stanowi ona konkurencji dla wyspecjalizowanych procedur, które w głęboki sposób wykorzystują charakteryzację optymalnych rozwiązań, przedstawionych w dalszej części pracy. Przeważnie algorytm proponowany przez Ngoma nie jest w stanie wygenerować poprawnego rozszerzenia liniowego dla wejściowego posetu w rozsądnym czasie. Taka słaba wydajność wynika z faktu, ze potencjalne rozwiązania poszukiwane są pośród wszystkich permutacji elementów posetu. Nawet jeśli wystartujemy ewolucję od poprawnego rozszerzenia liniowego (np. zachłannego), to ich jakość nie ulega znaczącej poprawie. Wobec takiej sytuacji zaproponowano adaptację przeszukiwania z zakazami. Algorytm opisano w artykule [48].

Ogólne założenia paradygmatu przeszukiwania z zakazami zostały wymienione w podrozdziale 1.2. Przechodzimy do opisu naszej adaptacji dla problemu liczby skoków.

W naszym podejściu, każde rozwiązanie jest jakimś półsilnie zachłannym rozszerzeniem liniowym L posetu P. Rozwiązanie sąsiedzkie jest generowane z L przez podział L pomiędzy sąsiadującymi łańcuchami, zachowanie pierwszej części rozszerzenia liniowego i uzupełnienie jej zgodnie z algorytmem półsilnie zachłannym (algorytm 2.3.12),

uruchomionym na pozostałym podposecie posetu P. Rozszerzenie liniowe L jest reprezentowane jako lista łańcuchów, które z kolei są listami elementów posetu. Jeśli więc zdecydujemy o podziale L po nc łańcuchach, to $L[0] \oplus \ldots \oplus L[nc-1]$ staje się początkową częścią sąsiada L'. Nasza adaptacja jest sformułowana jako algorytmy 2.4.1 i 2.4.2.

Przechowujemy dwie listy tabu. $TabuPositions$ jest cykliczną listą zawierającą $TabuSize$ ostatnich pozycji podziału. Dokładniej, jeśli L jest dzielona po nc łańcuchach, to para (nc, ne) może być dodana do $TabuPositions$, gdzie $ne = |L[0]| + \ldots + |L[nc-1]|$ jest łączną liczbą elementów posetu w zachowanych łańcuchach. Druga lista tabu, $TabuPaths$, zawiera zachłanne ścieżki przed i po pozycjach podziału. To znaczy, za każdym razem, gdy L jest dzielone po nc łańcuchach i uzupełniane, dodajemy do $TabuPaths$ czwórkę $(nc, ne, L[nc-1], L[nc])$. Wprowadzenie takich list tabu jest motywowane faktem, że są dwa główne punkty decyzyjne przy generowaniu sąsiada: po pierwsze, L jest dzielone w jakiejś pozycji nc; po drugie, wybierana jest jedna z dostępnych ścieżek zachłannych w $D(P - (L[0] \oplus \ldots \oplus L[nc-1]))$. $TabuPositions$ jest listą cykliczną, więc po dodaniu nowego wpisu, najstarszy jest usuwany. Natomiast $TabuPaths$ jest listą statyczną, z której nic nie jest usuwane w procesie algorytmu.

Gdy rozszerzenie liniowe jest generowane lub uzupełniane, mamy oczywistą zmianę w stosunku do oryginalnego algorytmu półsilnie zachłannego: ścieżka półsilnie zachłanna nie może zostać wybrana (i dodana do L), jeśli jest ona zawarta w liście $TabuPaths$ w miejscu obecnej pozycji podziału (krok **4** algorytmu 2.4.1). Co więcej, `OptLinExt` przegląda ścieżki zachłanne w sposób systematyczny, zaś tutaj ścieżki wybierane są losowo. Oczywiście, implementacja jest też poszerzona o aktualizowanie obu list tabu w każdej iteracji.

Pozycja podziału (nc, ne) jest dodawana do $TabuPositions$ wtedy, gdy pozostały podposet ma albo ścieżkę silnie zachłanną, albo tylko jedną ścieżkę półsilnie zachłanną, albo gdy jego liczba skoków jest oszacowana jako nieobiecująca (krok **2** algorytmu 2.4.1), albo jego liczba skoków została już policzona dokładnie wcześniej (krok **3** algorytmu 2.4.1). Innymi słowy, (nc, ne) nie jest tabu, gdy istnieją nadal niezbadane wybory zachłannych ścieżek w pozostałym podposecie. Każdy wybór ścieżki zachłannej jest zachowywany w $TabuPaths$.

Algorytm 2.4.1. `TS-CompleteLinExt`

Wejście: Poset P, początkowe łańcuchy jakiegoś zachłannego rozszerzenia liniowego $partialL = C_0 \oplus \ldots C_{c-1}$, i minimalna liczba skoków $s_\star$ znaleziona do tej pory.

Wyjście: $L = C_0 \oplus \ldots \oplus C_{c-1} \oplus C_c \oplus \ldots \oplus C_m$, rozszerzenie liniowe P.

1. { Inicjalizacja }

$D := D(P - partialL)$, diagram łukowy dla $P - partialL$

$c :=$ liczba łańcuchów w $partialL$

$e :=$ liczba elementów posetu w $partialL$

$s_{LB} := s(partialL) + 1 + s_{LB}(D)$

{ $s_{LB}(D)$ jest dolnym ograniczeniem na $s(P - partialL)$, tw. 2.4.3 }

2. **if** $s_{LB} \geqslant s_\star$ **then** dodaj (c, e) do $TabuPositions$, **return** $\emptyset$

else go to 3

3. **if** D ma nie więcej łuków pozornych niż $MaxDummies$ **then**

$remainingL :=$ `OptLinExt`$(P - partialL)$

dodaj (c, e) do $TabuPositions$

if $s(partialL) + 1 + s(remainingL) \geqslant s_\star$ **then return** $\emptyset$

else return $partialL \oplus remainingL$

else go to 4

4. $remainingL := \emptyset$

{ uzupełnij rozszerzenie liniowe łańcuchami półsilnie zachłannymi }

while $D \neq \emptyset$

$S :=$ ścieżki silnie zachłanne w D

$W :=$ ścieżki półsilnie zachłanne w D

if $|S| > 0$ **then**

$\pi :=$ dowolna ścieżka z S

$remainingL := remainingL \oplus C_\pi$

if C_π jest pierwszym łańcuchem w $remainingL$ **then**

dodaj (c, e) do $TabuPositions$

else { nie ma ścieżek silnie zachłannych }

if $|W| = 1$ **then**

$\pi :=$ jedyna ścieżka z W, $remainingL := remainingL \oplus C_\pi$

if C_π jest pierwszym łańcuchem w $remainingL$ **then**

dodaj (c, e) do $TabuPositions$

else { jest wiele ścieżek półsilnie zachłannych }

if $remainingL = \emptyset$ {tj., pierwszy wybór po podziale} **then**

$\pi :=$ losowa ścieżka z $W - TabuPaths$;

if nie znaleziono **then** dodaj (c, e) do $TabuPositions$, **return** $\emptyset$

{ π dodana do $TabuPaths$ w alg. 2.4.2}

else { nie pierwszy łańcuch po podziale } $\pi :=$ losowa ścieżka z W

$remainingL := remainingL \oplus C_\pi$

$D := D(P - (partialL \oplus remainingL))$

5. **return** $partialL \oplus remainingL$

Algorytm 2.4.2. `TS-OptimizeJumpNumber`

Wejście: Poset P, liczba iteracji T.

Wyjście: $L = C_0 \oplus \ldots \oplus C_m$, półsilnie zachłanne rozszerzenie liniowe P.

1. { Inicjalizacja }

$TabuPositions := \emptyset$

$TabuPaths := \emptyset$

$currentL :=$ `TS-CompleteLinExt`$(P, \emptyset, |P|)$

$bestL := currentL$

$t := 0$ {bieżąca iteracja}

2. **while** $t \leqslant T$

$bestNeighbour = \emptyset$

{ pozycja podziału dla najlepszego sąsiada }

$bestNC = 0$, $bestNE = 0$

$t := t + 1$

3. { wybierz najlepsze rozwiązanie sąsiedzkie }

for $n = 1$ **to** $CheckedNeighbours$

$nc :=$ losowa liczba mniejsza niż $|currentL|$

$ne := |L[0]| + \ldots + |L[nc - 1]|$

{ takie, że $(nc, ne) \notin TabuPositions$ }

$partialL :=$ pierwszych nc łańcuchów w $currentL$

$neighbourL :=$ `TS-CompleteLinExt`$(P, partialL, s(bestLinExt))$

if $s(neighbourL) < s(bestNeighbour)$ **then**

$bestNeighbour := neighbourL$

$bestNC := nc$, $bestNE := |neighbourL[0..bestNC - 1]|$ { $= ne$ }

4. { przejdź do sąsiada, zaktualizuj wynik }
if $bestNeighbour \neq \emptyset$ **then**
$currentL := bestNeighbour$
zaktualizuj $TabuPaths$ o $(bestNC, bestNE,$
$currentL[bestNC - 1], currentL[bestNC])$
if $s(currentL) < s(bestL)$ **then** $bestL := currentL$
5. **return** $bestL$

Najbardziej czasochłonną podprocedurą jest konstrukcja diagramu łukowego dla pozostałego podposetu za każdym razem, gdy ścieżka zachłanna jest wybierana i dodawana do L. W związku z tym, istotne jest szybkie odrzucanie nieobiecujących rozwiązań częściowych, w których liczba skoków nie będzie lepsza od poprzednio znalezionej (w dotychczasowym przebiegu algorytmu). Zatem, gdy wybrany zostaje punkt podziału, a diagram zostaje zrekonstruowany, oceniamy ten wybór obliczając dolne ograniczenie na liczbę skoków pozostałego posetu (krok **2** algorytmu 2.4.1). Może się natychmiast okazać, że trzeba wylosować inny punkt podziału. Używamy dolnego ograniczenia danego twierdzeniem 2.4.3.

Twierdzenie 2.4.3 (Sysło [68])**.** *Jeśli $D(P)$ jest diagramem łukowym posetu P, to* $\sum_{v \in V} \max\{0, indeg_P(v) - 1\} \leqslant s(P)$.

Co więcej, jeśli liczba łuków pozornych w diagramie jest mniejsza niż pewna ustalona liczba $MaxDummies$, stosujemy `OptLinExt` na pozostałym posecie.

Doświadczenia obliczeniowe pokazujące wydajność algorytmu 2.4.2 są opisane w podrozdziałach 3.3 (dla posetów przedziałowych) i 4.3 (dla posetów dwuwymiarowych).

3. Problem skoków na posetach przedziałowych

W tym rozdziale ograniczamy rozważania problemu skoków do klasy posetów przedziałowych. Okazało się, że nawet w tej klasie problem skoków jest NP-trudny [52]. Jednak w odróżnieniu od innych klas posetów, tutaj zaprojektowano aż trzy algorytmy o stałym współczynniku aproksymacji. Oprócz przypomnienia tych algorytmów, prezentujemy w podrozdziale 3.7 główny teoretyczny wynik rozprawy: poprawę współczynnika aproksymacji algorytmu podanego przez Mitas [52]. Wszystkie trzy algorytmy z lat 90-tych mają współczynnik aproksymacji równy $\frac{3}{2}$. Pokazujemy, że można usprawnić algorytm Mitas i uzyskać aproksymację liczby skoków ze współczynnikiem 1.484. Wynik ten został opublikowany w artykule autora [44] oraz omówiony na międzynarodowej konferencji *Combinatorics 2012* w Perugii [45]. Charakteryzacja optymalnych rozszerzeń liniowych podana przez Mitas prowadzi nie tylko do dobrej aproksymacji, ale również do obniżenia złożoności dokładnego wyznaczania liczby skoków. W podrozdziale 3.9 zaprezentowano więc nowy algorytm dokładny. Ponadto w podrozdziale 3.8 opisano nowy algorytm genetyczny, eksploatujący te same idee.

W celu zróżnicowania wydajności badanych algorytmów, przeprowadzono serię testów porównujących je. Dlatego opisy algorytmów przeplatane są wynikami doświadczeń obliczeniowych wykonanych w oparciu o implementacje tych algorytmów, zrealizowane przez autora rozprawy. Wzięto pod uwagę różne rozmiary posetu od 10 do 100 przedziałów i dla każdego rozmiaru posetu generowano rodzinę posetów przedziałowych, by sprawdzić, który algorytm zwróci mniejszą liczbę skoków. Posety były losowane z rozkładu jednostajnego i wykładniczego. Co więcej, zweryfikowano też wydajność algorytmów na posetach utworzonych przez złączenie szeregowe z trudnych instancji zgromadzonych w bazie danych (opisanej w podrozdziale 3.10). Próbka tych posetów jest pokazana na rysunkach 3.2.7, 3.3.7, 3.7.9 i 3.8.3. Większy zbiór tych posetów umieszczono na stronie internetowej [47].

3.1. Charakteryzacja posetów przedziałowych

Definicja 3.1.1. Poset $(P, <_P)$ jest **posetem przedziałowym** (lub **porządkiem przedziałowym**), jeśli istnieje bijekcja pomiędzy elementami P i domkniętymi prze-

działami w $\mathbb{R}$,

$$P \longleftrightarrow \{I_p = [l(p), r(p)], l(p) \leqslant r(p)\}_{p \in P}$$

taka, że $p <_P q$ wtedy i tylko wtedy, gdy $r(p) < l(q)$.

Charakteryzacja takich posetów znana jest jako twierdzenie 3.1.2 Fishburna, które przytaczamy wraz z dowodem Felsnera [28].

Twierdzenie 3.1.2 (Fishburn [29])**.** *Niech $(P, <_P)$ będzie posetem. Wówczas następujące warunki są równoważne:*

(1) P jest posetem przedziałowym.

(2) P nie zawiera podposetu izomorficznego z **2+2***, tzn. nie istnieją $\{p, q, r, s\} \subseteq P$ takie, że $p <_P q$, $r <_P s$ i nie ma innych porównywalności między p, q, r, s.*

(3) Dla każdych dwóch elementów $p, q \in P$ zachodzi $Pred(p) \subseteq Pred(q)$ albo $Pred(p) \supseteq Pred(q)$, to znaczy: rodzina zbiorów poprzedników $\mathcal{P}_P$ jest uporządkowana liniowo ze względu na zawieranie.

(4) Rodzina zbiorów następników $\mathcal{S}_P$ jest uporządkowana liniowo ze względu na zawieranie.

Dowód. (1) $\implies$ (2) Pokazujemy, że jeśli poset zawiera **2+2**, to nie może być przedziałowy.

Spróbujmy rozważyć przedziałową reprezentację posetu **2+2**. Niech pierwszym łańcuchem będzie $a <_P b$ (więc $r(a) < l(b)$), a drugim $c <_P d$ (więc $r(c) < l(d)$). Rozważmy możliwe ułożenia tych łańcuchów, reprezentowanych przedziałami. W każdym przypadku mamy albo $r(c) < l(b)$, albo $r(a) < l(d)$ (czyli $c <_P b$ albo $a <_P d$), a więc podposet $\{a, b, c, d\}$ nie jest izomorifczny z **2+2** (dochodzi przynajmniej jedna porównywalność).

(2) $\implies$ (3) Załóżmy, że (3) nie jest prawdą, to znaczy istnieją dwa elementy d, b, które mają zbiory poprzedników $Pred(d)$, $Pred(b)$ nieporównywalne przez zawieranie. Skoro ani $Pred(d)$ nie jest podzbiorem $Pred(b)$, ani $Pred(b)$ nie jest podzbiorem $Pred(d)$, to istnieją dwa elementy $c \in Pred(d)$, $a \in Pred(b)$ takie, że $r \notin Pred(b)$ i $p \notin Pred(d)$. Czyli podposet $\{a, b, c, d\}$ jest izomorficzny z **2+2**.

(4) $\implies$ (3) Załóżmy, że (3) nie jest prawdą (jw.), toteż mamy **2+2**, $a <_P b$ i $c <_P d$. Zbiory $Succ(a)$ i $Succ(c)$ nie są porównywalne przez zawieranie.

Analogicznie, (3) $\implies$ (4).

Pozostaje implikacja (3) $\implies$ (1), czyli konstrukcja reprezentacji przedziałowej. Niech $\emptyset = Pred_1 \subset Pred_2 \subset \ldots \subset Pred_e$ będzie uporządkowaniem różnych zbiorów poprzedników. Dla $p \in P$ niech $l(p) \in \{1, \ldots, e\}$ będzie taki, że $Pred_P(p) = Pred_{l(p)}$ (pozycja $Pred_P(p)$ w powyższym porządku zbiorów poprzedników). Niech $r(p) \in \{1, \ldots, e\}$ będzie najmniejszym indeksem takim, że $p \in Pred_{r(p)}$ (a jeśli nie istnieje, tzn. $p \in Max(P)$, to $r(p) = e + 1$). Skoro $p \notin Pred(p)$, zawsze mamy $l(p) \leqslant r(p) - 1$. Pokazujemy, że przedziały $[l(p), r(p) - 1]$ dla $p \in P$ reprezentują $(P, <_P)$. Jeśli $p <_P q$, to $p \in Pred_P(q)$, a więc $r(p) - 1 < l(q)$, czyli przedział dla p jest przed przedziałem dla q. Jeśli p, q są nieporównywalne w P, to $p \notin Pred_P(q)$ implikuje $l(q) \leqslant r(p) - 1$, a zarazem $q \notin Pred_P(p)$ implikuje $l(p) \geqslant r(q) - 1$, więc przedziały dla p, q się przecinają. $\square$

Można analogicznie wykazać, że (4) $\implies$ (1). Otrzymamy taką samą reprezentację przedziałową. Płynie stąd następujący wniosek 3.1.3.

Wniosek 3.1.3. *Dla każdego posetu przedziałowego P zachodzi $|\mathcal{P}_P| = |\mathcal{S}_P|$. Oznaczamy liczbę różnych zbiorów następników oraz liczbę różnych zbiorów poprzedników przez e.*

Twierdzenie 3.1.2 uzasadnia poprawność poniższego algorytmu 3.1.4, za pomocą którego otrzymujemy **kanoniczną reprezentację** posetów przedziałowych.

Algorytm 3.1.4. Kanoniczna reprezentacja posetu przedziałowego.

Wejście: Poset przedziałowy $(P, <_P)$

Wyjście: $\{I_p = [l(p), r(p)]\}_{p \in P}$, rodzina przedziałów reprezentująca $(P, <_P)$.

1. Wygeneruj dla każdego elementu posetu jego zbiór następników i zbiór poprzedników.

2. Uporządkuj różne zbiory następników ze względu na zawieranie:

$$Succ_1 \supset Succ_2 \supset \ldots \supset Succ_e = \emptyset$$

i uporządkuj różne zbiory poprzedników ze względu na zawieranie:

$$\emptyset = Pred_1 \subset Pred_2 \subset \ldots \subset Pred_e.$$

3. Każdemu elementowi $p \in P$ przyporządkuj lewy koniec przedziału $l(p) = i - 1$ taki, że $Pred_i = Pred(p)$ oraz prawy koniec przedziału $r(p) = j - 1$ taki, że $Succ_j = Succ(p)$.

Tabelaryczna reprezentacja posetów przedziałowych

Przytaczając twierdzenie 3.1.2 przypomnieliśmy charakteryzację posetów przedziałowych oraz wynikającą z niego kanoniczną reprezentację takich posetów (algorytm 3.1.4). W dalszej części pracy, a w szczególności w podrozdziale 3.5 dotyczącym algorytmu Mitas, pracujemy z tą reprezentacją, wpisaną dodatkowo do **tabeli** o e wierszach i e kolumnach, gdzie $e = |\mathcal{P}_P| = |\mathcal{S}_P|$. Wiersze i kolumny tabeli są ponumerowane kolejnymi liczbami naturalnymi, poczynając od zera, a więc odnosimy się do nich jako: $row_0, row_1, \ldots, row_{e-1}$ oraz $col_0, col_1, \ldots, col_{e-1}$. Przedział $[l(p), r(p)]$ trafia do komórki w wierszu $l(p)$, kolumnie $r(p)$. Zauważmy, że komórka tabeli może zawierać wielokrotne elementy posetu, a także, że w tabeli nie ma ani jednego pustego wiersza oraz ani jednej pustej kolumny.

Przykład 3.1.5. Na rysunku 3.1.7 umieszczony jest diagram Hassego przykładowego posetu przedziałowego. Obok widoczna jest tabela zawierająca kanoniczną reprezentację tego posetu.

Dla posetu widocznego na rysunku 3.1.7 rozpisujemy zbiory poprzedników i zbiory następników w lewej części tabeli 3.1.6.

Tabela 3.1.6. Przykładowe wykonanie algorytmu 3.1.4

p	$Pred(p)$	$Succ(p)$
1	$\emptyset$	$\{3,4,5,6,7,8,9\}$
2	$\emptyset$	$\{5,6,7,8,9\}$
3	$\{1\}$	$\{6,7,8,9\}$
4	$\{1\}$	$\{8,9\}$
5	$\{1,2\}$	$\{7,8,9\}$
6	$\{1,2,3\}$	$\{8,9\}$
7	$\{1,2,3,5\}$	$\{9\}$
8	$\{1,2,3,4,5,6\}$	$\emptyset$
9	$\{1,2,3,4,5,6,7\}$	$\emptyset$

p	$l(p)$	$r(p)$
1	0	0
2	0	1
3	1	2
4	1	4
5	2	3
6	3	4
7	4	5
8	5	6
9	6	6

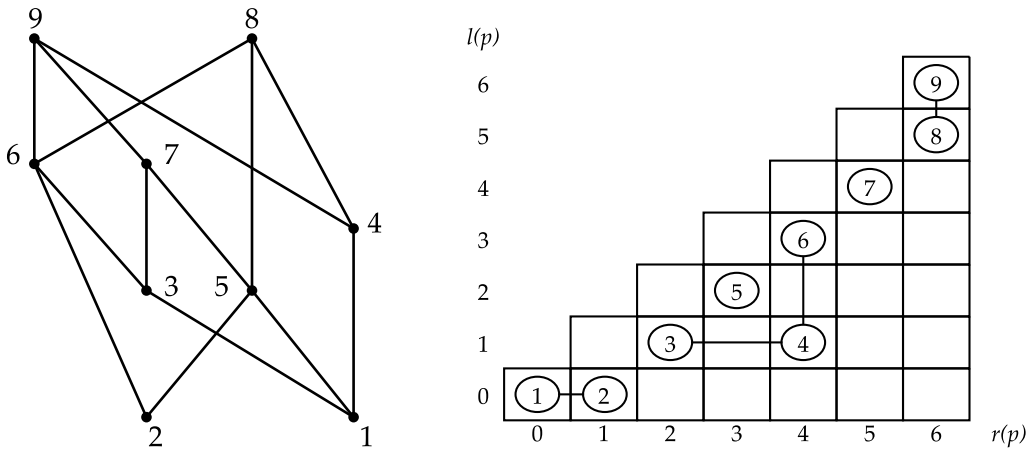
Porządkujemy przez zawieranie zbiory poprzedników:

$$\emptyset \subset \{1\} \subset \{1,2\} \subset \{1,2,3\} \subset \{1,2,3,5\} \subset \{1,2,3,4,5,6\} \subset \{1,2,3,4,5,6,7\}$$

oraz zbiory następników:

$$\{3,4,5,6,7,8,9\} \supset \{5,6,7,8,9\} \supset \{6,7,8,9\} \supset \{7,8,9\} \supset \{8,9\} \supset \{9\} \supset \emptyset$$

Na podstawie pozycji zbiorów poprzedników i następników w tym porządku, wyznaczamy początki i końce przedziałów (prawa część tabeli 3.1.6). Kanoniczne przedziały wpisujemy do tabeli, jak na rysunku 3.1.7.



Rysunek 3.1.7. Kanoniczna reprezentacja przedziałowa posetu

W rozprawie ilustrujemy posety przedziałowe w postaci tabeli, gdyż oprócz samych porównywalności widać w niej przedziałową realizację posetu.

3.2. 3/2-aproksymacja Felsnera

Algorytmy zachłanne dla problemu skoków opierają się na naturalnej zasadzie preferowania kolejno tych elementów minimalnych, których wybór ma szansę zwiększyć liczbę progów w generowanym rozszerzeniu liniowym. Efektem badań prowadzonych w zakresie algorytmów zachłannych dla minimalizacji liczby skoków były propozycje reguł wyboru kolejnego elementu w L. W każdym kroku algorytmu zachłannego próbuje się wybrać taki $p \in P$, który jest początkowym elementem jakiegoś optymalnego rozszerzenia liniowego pozostałej części posetu.

Algorytm Faigla-Schradera

Przypominamy teraz jedną z pierwszych zachłannych heurystyk dla problemu skoków na posetach przedziałowych.

Charakteryzacja posetów przedziałowych (twierdzenie 3.1.2) zapewnia, że dla dowolnych $p, q \in P$ mamy $Succ(p) \subseteq Succ(q)$ albo $Succ(q) \subseteq Succ(p)$. Dla posetu przedziałowego P oznaczmy:

$$SMin(P) = \{p \in P : Succ(p) \supseteq Succ(q) \text{ dla wszelkich } q \in P\}.$$

Oczywiście $SMin(P) \neq \emptyset$ oraz $SMin(P) \subseteq Min(P)$.

Lemat 3.2.1 (Faigle, Schrader [26])**.** *Niech P będzie posetem przedziałowym. Dla każdego $p \in SMin(P)$ istnieje jakieś optymalne rozszerzenie liniowe P rozpoczynające się od p.*

W oparciu o lemat 3.2.1, Faigle i Schrader zaprojektowali heurystyczny algorytm zachłanny [26] dla minimalizacji liczby skoków posetu przedziałowego, opisany poniżej jako algorytm 3.2.2 oraz udowodnili pierwszy współczynnik aproksymacji dla tego problemu (twierdzenie 3.2.3).

Algorytm 3.2.2. Algorytm Faigla-Schradera dla posetów przedziałowych.

Wejście: Poset przedziałowy $(P, <_P)$

Wyjście: Zachłanne rozszerzenie liniowe L posetu P.

1. L jest pustą listą, do której dodajemy kolejne elementy generowanego rozszerzenia liniowego. H jest aktualizowanym co krok zbiorem następników poprzednio dodanego elementu, należących zarazem do $SMin(P)$; początkowo $H = \emptyset$.

2. Powtarzaj, dopóki $P \neq \emptyset$:

(a) Jeśli $H = \emptyset$, to wybierz dowolny $p \in SMin(P)$. W przeciwnym razie wybierz dowolny $p \in H$.

(b) Dodaj wybrany element p na koniec listy L.

(c) Usuń element p z posetu P.

(d) $H := Succ_P(p) \cap SMin(P)$.

3. Zwróć rozszerzenie liniowe L.

Twierdzenie 3.2.3 (Faigle, Schrader [26]). *Algorytm* 3.2.2 *jest algorytmem* 2*-aproksymacyjnym dla problemu skoków na posetach przedziałowych.*

Felsner zmodyfikował algorytm 3.2.2 Faigla-Schradera w sposób przedstawiony w algorytmie 3.2.5 i udowodnił następujące twierdzenie 3.2.4.

Twierdzenie 3.2.4 (Felsner [27]). *Algorytm* 3.2.5 *ma współczynnik aproksymacji ograniczony przez* $\frac{3}{2}$.

Algorytm 3.2.5. Algorytm Felsnera dla liczby skoków na posetach przedziałowych.

Wejście: Poset przedziałowy $(P, <_P)$

Wyjście: Zachłanne rozszerzenie liniowe L posetu P.

1. L jest pustą listą, do której dodajemy kolejne elementy generowanego rozszerzenia liniowego. H jest aktualizowanym co krok zbiorem następników poprzednio dodanego elementu, należących zarazem do $Min(P)$; początkowo $H = \emptyset$. M jest aktualizowanym co krok zbiorem następników poprzednio dodanego elementu, należących zarazem do $SMin(P)$; początkowo $M = \emptyset$.

2. Powtarzaj, dopóki $P \neq \emptyset$:

(a) Jeśli $H = \emptyset$, to wybierz dowolny $p \in SMin(P)$. Jeśli $H \neq \emptyset$ i $M = \emptyset$, to wybierz dowolny $p \in H$. Jeśli $M \neq \emptyset$, to wybierz dowolny $p \in M$.

(b) Dodaj wybrany element p na koniec listy L.

(c) Usuń element p z posetu P.

(d) $H := Succ_P(p) \cap Min(P)$.

(e) $M := Succ_P(p) \cap SMin(P)$.

3. Zwróć rozszerzenie liniowe L.

Słownie, w pierwszej kolejności preferujemy taki następnik poprzedniego elementu, który należy do $SMin(P)$. Jeśli $SMin(P)$ nie zawiera następników poprzednio wybranego elementu, to wybieramy jakikolwiek minimalny następnik poprzedniego elementu (czyli, jak w prostym algorytmie zachłannym, preferujemy próg). Jeśli $Min(P)$ nie zawiera następników poprzedniego elementu (tj. nie ma możliwości dodania progu do L), to wybieramy dowolny element z $SMin(P)$.

Algorytm Felsnera – doświadczenia obliczeniowe

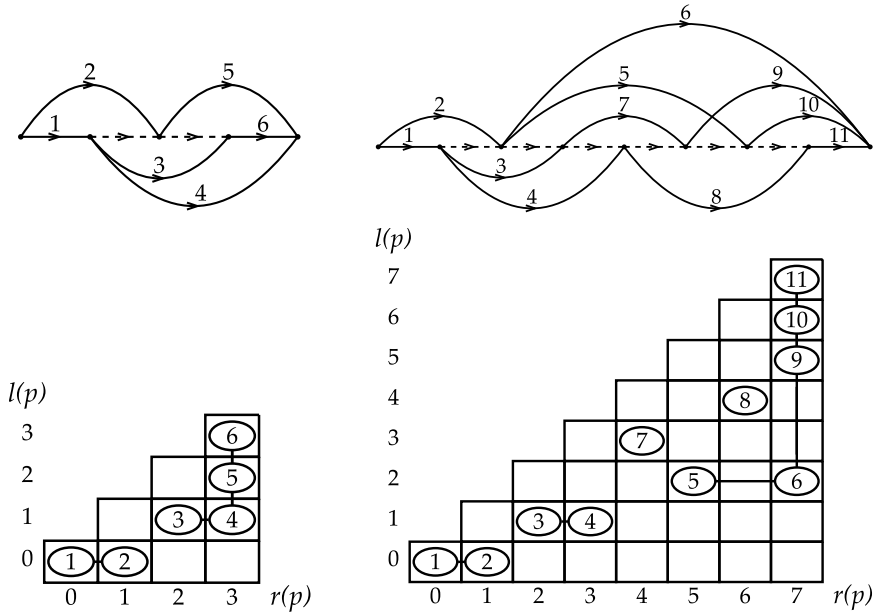
Tabela 3.2.6 pokazuje wyniki zastosowania algorytmu Felsnera do losowych posetów przedziałowych, wygenerowanych z rozkładu jednostajnego i wykładniczego. W tabelach, n_{x-y} oznacza zakres rozmiaru posetu od x do y. Obserwujemy, że algorytm wypada dobrze, gdyż średnie odchylenie od wartości optymalnych wynosi kilka procent. Błąd odrobinę wzrasta, gdy rozważamy większe posety.

Tabela 3.2.6. Średni błąd rozwiązań generowanych algorytmem Felsnera na losowych posetach przedzialowych różnego rozmiaru

Rozkład jednostajny				
próbka	n_{10-20}	n_{20-30}	n_{30-50}	n_{50-100}
10	0,0111	0,0230	0,0283	0,0591
100	0,0104	0,0271	0,0403	0,0586
1000	0,0094	0,0259	0,0444	0,0645
10000	0,0109	0,0272	0,0450	0,0624

Rozkład wykładniczy		
próbka	n_{10-40}	n_{40-100}
10	0,0114	0,0323
100	0,0184	0,0299
1000	0,0149	0,0307
10000	0,0161	0,0304

Na rysunku 3.2.7 pokazano małą rodzinę posetów, które wymuszają 50% błędu, gdy rozważamy najgorszy przebieg algorytmu Felsnera.



Rysunek 3.2.7. Trudne przypadki dla algorytmu Felsnera (posety, dla których osiąga się 50% odchylenia od optimum)

Dla przykładu, 11-elementowy poset ma optymalne rozszerzenie liniowe $(1,4) \oplus (2,6) \oplus (3,7,9) \oplus (5,10) \oplus (8,11)$, ale w pesymistycznym przypadku powtarzania niewłaściwych wyborów, algorytm może wygenerować $(1,3) \oplus (2,5) \oplus (4,8) \oplus (7,9) \oplus (6) \oplus (10) \oplus (11)$, które ma 6 skoków zamiast 4. Obserwujemy, że dwie trudne instancje można złożyć szeregowo, by utworzyć inny poset, dla którego algorym może potwórzyć te same błędy. Zatem trudne przykłady dla algorytmu Felsnera mogą mieć dowolny rozmiar.

3.3. 3/2-aproksymacja Sysły

W podrozdziale 2.3 przedstawiono podejście algorytmiczne Sysły do problemu minimalizacji liczby skoków posetu. Sformułowano algorytm przeszukiwania wyczerpującego (algorytm 2.3.13) oraz heurystyczny algorytm generujący jedno półsilnie zachłanne rozszerzenie liniowe (algorytm 2.3.12). W tym podrozdziale omówione są specjalizacje pojęć użytych do sformułowania tamtych algorytmów w przypadku, gdy ograniczamy rozważania do posetów przedziałowych. W efekcie otrzymuje się algorytm aproksymacyjny ze współczynnikiem $\frac{3}{2}$.

W przypadku posetów przedziałowych zwarty diagram łukowy budujemy algorytmem 3.3.1, a więc korzystamy z kanonicznej reprezentacji przedziałowej. Własności tak zbudowanego diagramu pozwoliły udowodnić twierdzenie 3.3.5 o aproksymacji.

Algorytm 3.3.1. Konstrukcja diagramu łukowego dla posetów przedziałowych.

Wejście: Poset przedziałowy P.

Wyjście: Zwarty diagram łukowy reprezentujący poset P.

1. Wyznacz kanoniczną reprezentację przedziałową $\{I_p\}_{p \in P}$ posetu P (stosując algorytm 3.1.4, dodawszy P do $\mathcal{P}_P$ i do $\mathcal{S}_P$).

2. Zbiorem wierzchołków diagramu jest $V = \{0, 1, \ldots, e\}$, gdzie $e = |\mathcal{S}| = |\mathcal{P}|$.

3. Określ zbiór łuków diagramu:

(a) Dla każdego $p \in P$ przedziałowi $I_p = [l(p), r(p)]$ (otrzymanemu w kroku **1**) odpowiada łuk posetu $(l(p), r(p))$.

(b) Dla każdego $i = 1, 2, \ldots, e-2$, jeśli $(i, i+1)$ nie jest łukiem posetu, dodaj łuk pozorny $(i, i+1)$.

W kroku **1** algorytmu 3.3.1 dodajemy P do rodziny różnych zbiorów następników

oraz poprzedników, co powoduje, że otrzymane przedziały mają prawy koniec zwiększony o 1 w stosunku do kanonicznej reprezentacji otrzymanej algorytmem 3.1.4 bez tej modyfikacji.

W diagramach łukowych dla posetów przedziałowych, wygenerowanych algorytmem 3.3.1, mamy szczegółowe definicje ścieżek silnie i półsilnie zachłannych.

Definicja 3.3.2. Ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$ w diagramie łukowym $D(P)$ posetu przedziałowego spełnia następujące warunki:

(a) żaden z grotów $h(a_i), i = 1, 2, \ldots, l-1$ nie jest grotem żadnego łuku innego niż a_i i albo poset P nie ma elementów innych niż zawarte w ścieżce π, albo istnieje łuk b w $D(P)$ (być może pozorny) taki, że $b \neq a_l$ i $h(b) = h(a_l)$,

(b) każdy z łuków $a_i, i = 1, 2, \ldots, l$, jest łukiem posetu,

(c) przedziały w kanonicznej reprezentacji P, odpowiadające wszystkim łukom ścieżki, za wyjątkiem być może ostatniego, mają długość 1: $t(a_1) = 0$, $h(a_i) = t(a_{i+1}) = i$, $i = 1, 2, \ldots, l-1$, $h(a_l) \geqslant l$.

Definicja 3.3.3. Ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$ w diagramie łukowym $D(P)$ posetu przedziałowego jest **ścieżką silnie zachłanną**, jeśli spełnia przynajmniej jeden z warunków:

(a) $h(\pi) = e$, tzn. $h(\pi)$ jest ujściem $D(P)$,

(b) $h(\pi)$ jest grotem łuku posetu $b \neq a_l$ takiego, że żaden wierzchołek spośród $\{0, 1, \ldots, t(b)\}$ nie jest incydentny z łukiem pozornym.

Algorytm generujacy jedno półsilnie zachłanne rozszerzenie liniowe jest sformułowany jako algortym 2.3.12. Algorytm 2.3.14 jest ogólną procedurą rekonstrukcji diagramu łukowego po usunięciu jednej ścieżki zachłannej. Dla posetów przedziałowych specjalna wersja tej procedury jest opisana poniżej jako algorytm 3.3.4.

Algorytm 3.3.4. Usunięcie ścieżki zachłannej w przypadku posetu przedziałowego.

Wejście: Zwarty diagram łukowy D posetu przedziałowego P, ścieżka zachłanna $\pi = (a_1, a_2, \ldots, a_l)$.

Wyjście: Zwarty diagram łukowy pozbawiony elementów ścieżki π.

1. Usuń łuki $a_1, a_2, \ldots, a_l$ z diagramu D.

2. Jeśli $h(a_l) \geqslant t(a_l) + 2$, to wykonaj następujące kroki:

(a) Jeśli istnieje łuk pozorny $a^{(i)} = (t(a_l), t(a_l) + 1)$, to usuń go z diagramu D.

(b) Jeśli żaden łuk posetu różny od a_l nie kończy się wierzchołkiem $h(a_l)$, to usuń z diagramu D łuk pozorny $a^{(ii)} = (h(a_l) - 1, h(a_l))$. Jeśli dodatkowo D zawiera łuk posetu $(h(a_l) - 1, h(a_l) + 1)$ i łuk pozorny $a^{(iii)} = (h(a_l), h(a_l) + 1)$, to usuń $a^{(iii)}$ z diagramu D. Zastąp wierzchołki $h(a_l) - 1$, $h(a_l)$ jednym wierzchołkiem.

3. Zastąp wszystkie wierzchołki o stopniu wejściowym równym 0 jednym wierzchołkiem.

4. Przenumeruj wierzchołki otrzymanego digrafu zaczynając od 0.

W efekcie otrzymuje się następujące twierdzenie 3.3.5 o aproksymacji.

Twierdzenie 3.3.5 (Sysło [69])**.** *Algorytm* 2.3.12 *dla posetów przedziałowych ma współczynnik aproksymacji ograniczony przez* $\frac{3}{2}$.

Dowód jest ściśle związany z poprawnością algorytmu 3.3.4, z którego wynika, że dowolny wybór ścieżki zachłannej (zgodny z regułami algorytmu 2.3.12) pociąga za sobą usunięcie co najwyżej trzech łuków pozornych z diagramu. Dla innych klas posetów nie mamy takiej własności.

Algorytm Sysły – doświadczenia obliczeniowe

W tabeli 3.3.6 pokazano wyniki algorytmu Sysły na losowych posetach przedziałowych wygenerowanych z rozkładu jednostajnego i wykładniczego. Obserwujemy, że średni błąd nie przekracza dwóch procent i narasta, gdy zwiększamy moc posetu. Doświadczenie pokazuje, że algorytm Sysły jest rząd wielkości $(10\times)$ lepszy od algorytmu Felsnera.

Tabela 3.3.6. Średni błąd rozwiązań generowanych algorytmem Sysły na losowych posetach przedzialowych różnego rozmiaru

Rozkład jednostajny				
próbka	n_{10-20}	n_{20-30}	n_{30-50}	n_{50-100}
10	0,0111	0,0000	0,0000	0,0021
100	0,0049	0,0007	0,0062	0,0061
1000	0,0004	0,0032	0,0047	0,0070
10000	0,0008	0,0027	0,0051	0,0065

Rozkład wykładniczy		
próbka	n_{10-40}	n_{40-100}
10	0,0000	0,0041
100	0,0010	0,0010
1000	0,0007	0,0011
10000	0,0008	0,0010

Podobnie jak w przypadku algorytmu Felsnera, szukaliśmy trudnych przypadków. Na rysunku 3.3.7 pokazano poset, który wymusza 50% nieoptymalność w najgorszym przebiegu algorytmu Sysły. Algorytm wybiera spośród łącznie czterech rozszerzeń liniowych. Jedno z nich ma 4 skoki, jedno 5 skoków, a pozostałe mają po 6 skoków:

$$(1,2,4) \oplus (3,6,7,10) \oplus (8,11,12,13,14) \oplus (9,15) \oplus (5,16,17,18,19)$$

$$(1,2,5) \oplus (3,6,7) \oplus (4,10) \oplus (8,11,12,13,14) \oplus (9,16,17,18) \oplus (15,19)$$

$$(1,2,5) \oplus (3,6,7) \oplus (4,9) \oplus (8,11,12) \oplus (10,13,16,17) \oplus (14,18) \oplus (15,19)$$

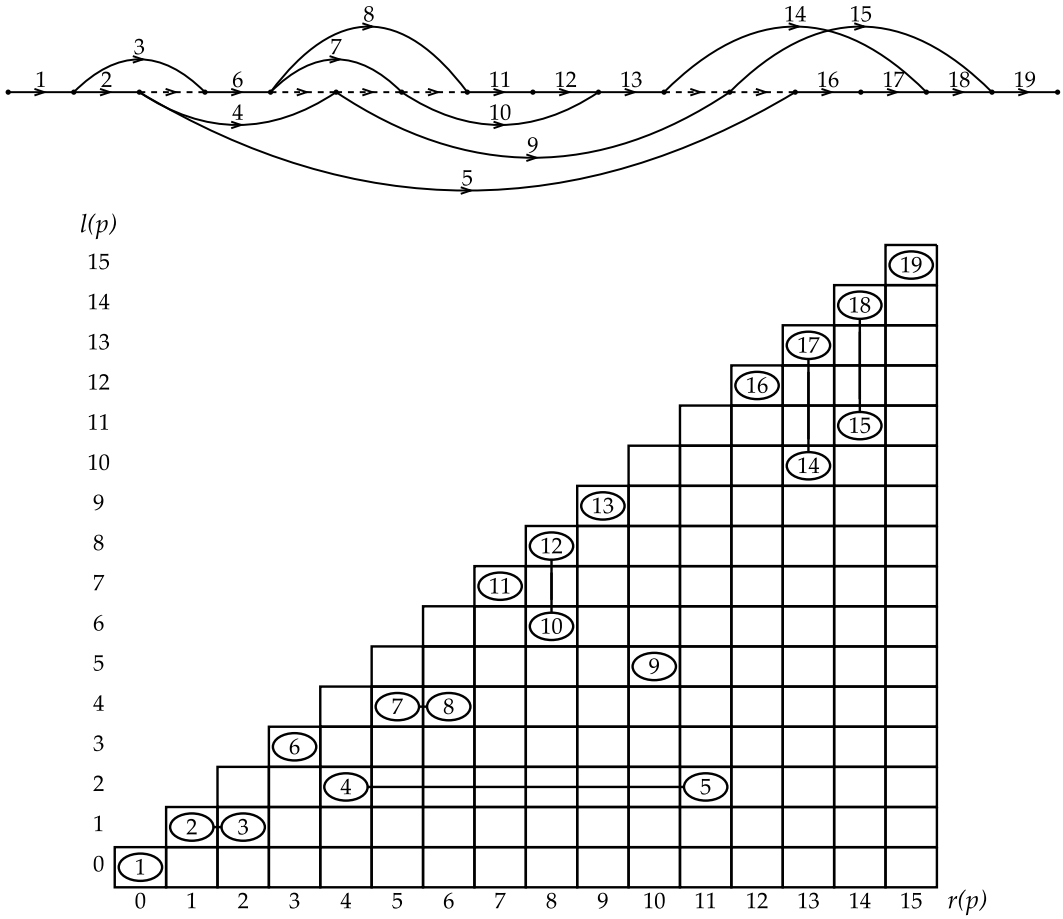
$$(1,2,5) \oplus (3,6,8) \oplus (4,9) \oplus (7,11,12) \oplus (10,13,16,17) \oplus (14,18) \oplus (15,19)$$

Zauważmy, że gdy dwie trudne instancje są połączone, to pesymistyczny przebieg algorytmu może dać ponownie 50%-suboptymalne rozszerzenie liniowe. Wnioskujemy, że istnieje nieskończona rodzina trudnych posetów dla algorytmu Sysły.

Przeszukiwanie z zakazami – losowe posety przedziałowe

W podrozdziale 2.4 podano algorytm przeszukiwania z zakazami, w którym korzystamy z charakteryzacji optymalnych rozszerzeń liniowych wyrażonej w terminach ścieżek półsilnie i silnie zachłannych. Opisujemy teraz doświadczenia obliczeniowe badające jakość rozwiązań generowanych tym algorytmem w zadanym limicie czasowym.

Pierwsze doświadczenie polegało na weryfikacji jakości rozwiązań generowanych przez proponowaną procedurę przeszukiwania z zakazami na losowych posetach przedziałowych. Dla każdej pary (n,k), $n \in \{100,150,200\}$, $k \in \{10,20,\ldots,\frac{n}{2}\}$ wylosowano setkę posetów przedziałowych, złożonych z n elementów i zawierających k łuków pozornych w diagramach łukowych. Dla każdej instancji uruchomiono algorytm 2.4.2 z limitem iteracji równym n. Algorytm był zatrzymywany również w razie znalezienia optymalnego rozwiązania (znanego a priori dzięki dokładnym obliczeniom). Za każdym razem znalezione zostało optymalne rozszerzenie liniowe. Pośród 100-elementowych posetów trwało to w najgorszym razie 45 iteracji (10 sekund), dla instancji zawierającej 40 łuków pozornych. Pośród posetów 200-elementowych najgorszy znaleziony przypadek wymagał 42 iteracji (47 sekund) i zawierał 60 łuków pozornych. Średnio, optimum było znajdowane po 20 iteracjach.

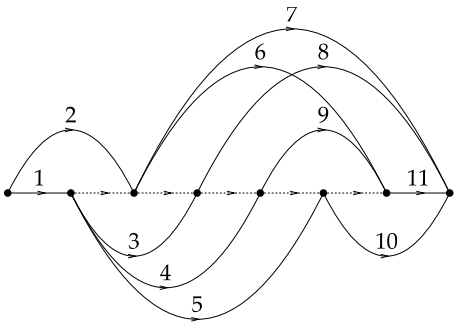


Rysunek 3.3.7. Jeden z trudnych przypadków dla algorytmu Sysły (poset, dla którego osiągane jest 50% odchylenie od optimum)

Przeszukiwanie z zakazami – trudne posety przedziałowe

W trakcie badań zachowania się algorytmów aproksymacyjnych Felsnera, Sysły i Mitas (opisanych odpowiednio w podrozdziałach: 3.2, 3.3, 3.5), rozpoznano wiele posetów, które wymuszają nieoptymalność rozwiązań generowanych przez te algorytmy. Zgromadzono obszerną bazę tych posetów, co opisujemy w podrozdziale 3.10 i w artykule [46], zaś niektóre trudne instancje są pokazane na rysunkach: 3.2.7, 3.3.7, 3.7.9 i 3.8.3. Wykorzystano te posety również do przetestowania procedury przeszukiwania z zakazami, zaproponowanej w podrozdziale 2.4. Dla przykładu, poset na rysunku 3.3.8 zawiera trzy ścieżki półsilnie zachłanne: $(1, 3)$, $(1, 4)$ i $(1, 5)$. Nie wiadomo, którą z nich

należy wybrać, by osiągnąć optymalne rozszerzenie liniowe. Jeśli wiele podobnych posetów połączymy szeregowo, to będzie bardzo wiele punktów decyzyjnych w procesie algorytmu zachłannego.



Rysunek 3.3.8. Poset przedziałowy wymuszający wielokrotne decyzje

W tym doświadczeniu wzięto trudne posety na 100, 150 i 200 elementach. Próbki 10 wyników dla każdej mocy posetu n są opisane w tabelach 3.3.10, 3.3.11 i 3.3.12. Każdy wpis odpowiada jednej instancji wejściowej P, i zawiera kolejno: liczbę łuków pozornych w P, jego liczbę skoków $s(P)$, ciąg przybliżonych rozwiązań generowanych przez algorytm 2.4.2, liczbę iteracji, czas działania w sekundach, i błąd $\frac{s_A - s(P)}{s(P)}$ najlepszego znalezionego rozszerzenia liniowego s_A. Jeśli n iteracji nie wystarczyło do znalezienia optimum, zapisano również czas ostatniej poprawy w nawiasach.

Wyniki doświadczenia, zebrane w tabelach 3.3.10, 3.3.11 i 3.3.12 pokazują, że warto stosować przeszukiwanie z zakazami, by poprawiać pojedyncze rozwiązania uzyskane algorytmami aproksymacyjnymi. Co więcej, znikomy błąd algorytmu sugeruje, że współczyniki aproksymacji udowodnione dla deterministycznych algorytmów być może uległyby poprawie, gdyby zastosować inne metody generowania rozszerzeń liniowych.

Obserwacja 3.3.9. *Zaroponowany algorytm przeszukiwania z zakazami (algorytm* 2.4.2*), zastosowany do n-elementowych posetów przedziałowych, generuje rozszerzenia liniowe o nie więcej skokach niż* 105% *optimum, w n iteracjach.*

Tabela 3.3.10. Wydajność przeszukiwania z zakazami na 100-elementowych posetach przedziałowych

	ł. poz.	$s(P)$	$s_A(P)$	iteracje	czas	błąd
P_1	28	22	$30 \ldots 22$	32	7 s	0.0000
P_2	32	30	$39 \ldots 30$	48	24 s	0.0000
P_3	29	24	$29 \ldots 24$	89	25 s	0.0000
P_4	32	26	$33 \ldots 26$	75	19 s	0.0000
P_5	28	21	$25 \ldots 21$	18	7 s	0.0000
P_6	26	22	$26 \ldots 22$	20	6 s	0.0000
P_7	31	28	$32 \ldots 28$	26	10 s	0.0000
P_8	32	26	$34 \ldots 26$	17	8 s	0.0000
P_9	35	29	$35 \ldots 30$	100 (36)	17 s (8 s)	0.0345
P_{10}	37	31	$36 \ldots 31$	59	38 s	0.0000

Tabela 3.3.11. Wydajność przeszukiwania z zakazami na 150-elementowych posetach przedziałowych

	ł. poz.	$s(P)$	$s_A(P)$	iteracje	czas	błąd
P_1	34	30	$36 \ldots 30$	47	19 s	0.0000
P_2	46	32	$39 \ldots 32$	14	19 s	0.0000
P_3	45	34	$42 \ldots 35$	150 (26)	81 s (18 s)	0.0294
P_4	50	44	$55 \ldots 46$	150 (90)	123 s (85 s)	0.0455
P_5	37	32	$39 \ldots 32$	36	17 s	0.0000
P_6	51	42	$52 \ldots 43$	150 (27)	35 s	0.0238
P_7	38	36	$45 \ldots 36$	36	52 s	0.0000
P_8	42	37	$45 \ldots 37$	52	66 s	0.0000
P_9	45	35	$42 \ldots 35$	39	20 s	0.0000
P_{10}	43	37	$42 \ldots 38$	150 (29)	143 s (31 s)	0.0270

Tabela 3.3.12. Wydajność przeszukiwania z zakazami na 200-elementowych posetach przedziałowych

	ł. poz.	$s(P)$	$s_A(P)$	iteracje	czas (s)	błąd
P_1	50	45	$52 \ldots 46$	200 (164)	380 (336)	0.0222
P_2	54	50	$61 \ldots 52$	200 (36)	352 (101)	0.0400
P_3	57	49	$61 \ldots 51$	200 (13)	258 (40)	0.0408
P_4	54	49	$62 \ldots 51$	200 (158)	415 (362)	0.0408
P_5	56	50	$62 \ldots 52$	200 (16)	306 (43)	0.0400
P_6	56	48	$57 \ldots 49$	200 (126)	328 (228)	0.0208
P_7	63	49	$57 \ldots 50$	200 (66)	327 (132)	0.0204
P_8	54	48	$54 \ldots 50$	200 (6)	160 (10)	0.0417
P_9	54	49	$59 \ldots 50$	200 (73)	235 (109)	0.0204
P_{10}	65	53	$64 \ldots 54$	200 (50)	413 (144)	0.0189

3.4. Porównanie algorytmów Felsnera i Sysły

W tabeli 3.4.1 pokazano wyniki porównań algorytmów aproksymacyjnych Felsnera i Sysły uruchomionych na tym samym posecie. Oznaczamy przez zw_F liczbę posetów, na których algorytm Felsnera zwrócił mniejszą liczbę skoków niż algorytm Sysły, spośród 1000 losowych posetów przedziałowych rozmiaru n, dla $n = 10, 20, \ldots, 100$. Podobnie, zw_S oznacza liczbę „pojedynków" wygranych przez algorytm Sysły.

Jak widać w tabeli 3.4.1, algorytm Sysły zwykle wypada lepiej od algorytmu Felsnera. Różnica wzrasta, gdy rośnie liczba przedziałów, tzn. szanse algorytmu Felsnera na pokonanie algorytmu Sysły maleją wraz ze wzrostem rozmiaru posetu. Doświadczenie zostało potwierdzone w porównaniach wykonanych na trudnych posetach (tabele 3.6.2 i 3.6.3 w podrozdziale 3.6).

Jedną z przyczyn przewagi algorytmu Sysły jest fakt, że diagram często zawiera ścieżkę silnie zachłanną, i algorytm Sysły wybiera taką ścieżkę. Natomiast reguły algorytmu Felsnera zaniedbują takie wybory mimo, że one prowadzą do optimum. Rozważmy 6-elementowy poset na rysunku 3.2.7. Ścieżka $(1, 4)$ jest klasyfikowana jako ścieżka silnie zachłanna, gdyż kończy się w ujściu diagramu. Sukcesywne redukcje $(1, 4)$, $(2, 5)$ i $(3, 6)$ wykonywane w algorytmie Sysły prowadzą do optymalnego rozszerzenia

liniowego. Ale w procedurze Felsnera zaczynamy od elementu 1 i dowolnie wybieramy spośród jego minimalnych następników 3 i 4, gdyż żaden z tych dwóch elementów nie należy do $SMin(P - \{1\})$. Przejście do elementu 3 prowadzi do rozszerzenia liniowego o trzech skoków zamiast dwóch (np. $(1,3) \oplus (2,6) \oplus (4) \oplus (5)$ lub $(1,3) \oplus (2,5) \oplus (4) \oplus (6)$), więc osiągamy w tym przypadku największy dopuszczalny błąd.

Tabela 3.4.1. Porównanie algorytmów Felsnera i Sysły na losowych posetach przedziałowych

Rozkład jednostajny			
n	zw_F	zw_S	*remisy*
10	0	30	970
20	3	193	804
30	0	420	580
40	7	583	410
50	5	695	300
60	6	812	182
70	5	840	155
80	1	890	109
90	5	908	87
100	1	949	50

Rozkład wykładniczy			
n	zw_F	zw_S	*remisy*
10	0	15	985
20	0	131	869
30	3	287	710
40	2	438	560
50	8	573	419
60	4	691	305
70	2	779	219
80	3	831	166
90	4	879	117
100	0	916	84

3.5. 3/2-aproksymacja Mitas

W tym podrozdziale omawiamy trzeci z algorytmów aproksymacyjnych dla problemu skoków na posetach przedziałowych. Podejście Mitas [52] stanowi odrębny nurt badań nad tym problemem, odmienny od strategii zachłannej. Zamiast tego charakteryzuje się optymalne rozwiązania w terminach tabeli zawierającej kanoniczną reprezentację posetu, i w terminach grafów określonych nad tą reprezentacją. W efekcie problem jest redukowany do pakowania krawędzi i cykli nieparzystych w grafie.

W podrozdziale 3.7 zaprezentowano główny teoretyczny wynik rozprawy, tj. poprawę współczynnika aproksymacji tego algorytmu.

W podrozdziale 3.1 omówiono kanoniczną reprezentację posetów przedziałowych oraz jej przedstawienie graficzne w tabeli o e wierszach i e kolumnach, gdzie

$e = |\mathcal{P}_P| = |\mathcal{S}_P|$. Wiersze i kolumny tabeli są ponumerowane: $row_0, row_1, \ldots, row_{e-1}$ oraz $col_0, col_1, \ldots, col_{e-1}$. Przedział $[l(p), r(p)]$ trafia do komórki w wierszu $l(p)$, kolumnie $r(p)$. Jeśli $p <_P q$, to znaczy, że $r(p) < l(q)$, a więc następniki elementu położonego w kolumnie i-tej widoczne są w wierszach $i+1$ oraz wyższych.

Wniosek 3.5.1 (Mitas [52])**.** *Niech P będzie posetem przedziałowym. Wówczas $b(P) \leqslant e - 1$.*

Dowód. W rozszerzeniu liniowym wszystkie elementy z kolumn $0, \ldots, j-1$ muszą być wypisane przed elementami wiersza j. Jeśli p_{i+1} jest w wierszu j, to (p_i, p_{i+1}) jest progiem, o ile p_i jest umiejscowiony w jednej z kolumn $0, \ldots, j-1$. Zatem do każdego z wierszy $1, \ldots, e-1$ może prowadzić co najwyżej jeden próg. □

Realizowalne ciągi progów

Przechodzimy do omówienia rezultatów Mitas, na których oparty jest jej algorytm aproksymacyjny dla liczby skoków posetu przedziałowego.

Zakładamy, że poset przedziałowy jest reprezentowany kanonicznymi przedziałami umieszczonymi w tabeli rozmiaru $e \times e$. Zgodnie z obserwacją, że następniki elementu położonego w kolumnie i-tej widoczne są w wierszach $i+1$ oraz wyższych, sformułowane jest pojęcie ciągu progów.

Definicja 3.5.2. Ciągiem progów nazywamy ciąg par uporządkowanych postaci

$$T = \{(r_k, l_k) : k = 1, \ldots, b; 0 \leqslant r_k, l_k, b \leqslant e - 1\},$$

spełniający $r_k < l_k$ oraz $l_k \leqslant r_{k+1}$ dla każdego k, zaś b jest jego długością. Niech $T_R := \{r_1, \ldots, r_b\}$ oznacza zbiór kolumn używanych w T, a $T_L := \{l_1, \ldots, l_b\}$ zbiór wierszy używanych w T. Mówimy, że kolumny $i \notin T_R$ oraz wiersze $j \notin T_L$ są **omijane** przez T. Ciąg progów T is **realizowalny**, jeśli $T = \{(r, l) :$ istnieje próg (p_i, p_{i+1}) w L taki, że $r(p_i) = r, l(p_{i+1}) = l\}$.

Uzasadnienie. Niech $L = p_1, p_2, \ldots, p_n$ będzie rozszerzeniem liniowym posetu przedziałowego $(P, <_P)$. Jeśli (p_i, p_{i+1}) jest progiem w L, to p_{i+1} jest następnikiem p_i w posecie, a więc $r(p_i) < l(p_{i+1})$. Niech (p_j, p_{j+1}) będzie następnym progiem w L.

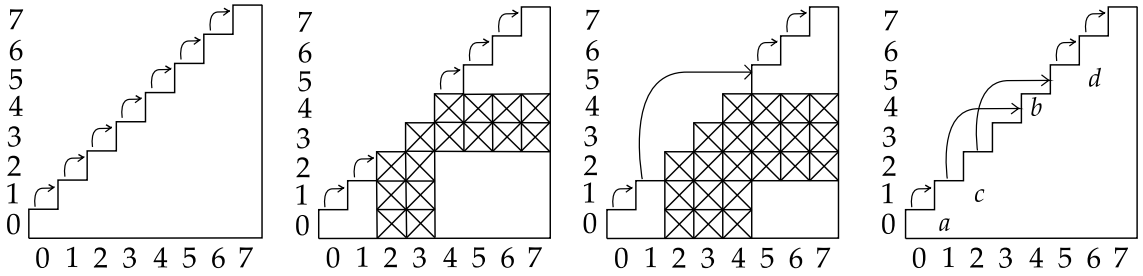
Wówczas p_j nie może być następnikiem p_{i+1} w P, gdyż L nie zawiera tej porównywalności, a wiec nie byłoby rozszerzeniem liniowym. Innymi słowy, $r(p_j) \geqslant l(p_{i+1})$. Co za tym idzie, jeśli $(p_{i_1}, p_{i_1+1}), \ldots, (p_{i_m}, p_{i_m+1})$ są kolejnymi progami w rozszerzeniu liniowym L, to spełnione są nierówności

$$r(p_{i_1}) < l(p_{i_1+1}) \leqslant r(p_{i_2}) < l(p_{i_2+1}) \leqslant \ldots \leqslant r(p_{i_m}) < r(p_{i_m+1}).$$

Na ilustracjach ciągi progów pokazujemy jako ciągi strzałek: dla każdej pary $(r, l) = (col, row)$ rysujemy strzałkę prowadzącą z kolumny *col* do wiersza *row*. Powyższe uzasadnienie definicji ciągu progów oznacza, że te strzałki się na rysunkach nie przecinają, a więc na rysunku 3.5.4 czwarta sytuacja jest niedopuszczalna w żadnym ciągu progów. Ciąg progów rozumiemy jako szablon dla rozszerzenia liniowego. Jak zostało wykazane, każde rozszerzenie liniowe posetu przedziałowego można odczytać wzdłuż jakiegoś ciągu progów. Problem skoków na posetach przedziałowych polega na znalezieniu realizowalnego ciągu progów o jak największej mocy.

Obserwacja 3.5.3 (Mitas)**.** *W realizowalnym ciągu progów omijane wiersze i kolumny zawsze są złożone z parami rozłącznych par postaci* (col_i, row_i) *lub* (col_i, row_{i+1}).

Dowód. Jeśli ciąg progów omija jakieś wiersze i kolumny, to zachodzi przynajmniej jeden z dwu przypadków: próg (r, l) prowadzi z kolumny r do wiersza l w taki sposób, że $l - r > 1$, bądź dla dwóch sąsiadujących progów $(r_i, l_i), (r_{i+1}, l_{i+1})$ zachodzi $l_i < r_{i+1}$. W pierwszym przypadku kolumny $r + 1, \ldots, l - 1$ sa omijane i wiersze $r + 1, \ldots, l - 1$ są omijane przez ciąg progów. W drugim przypadku kolumny $l_i, \ldots, r_{i+1} - 1$ i wiersze $l_i + 1, \ldots, r_{i+1}$ są omijane przez ciąg progów. Obie sytuacje widoczne są na rysunku 3.5.4, obok najdłuższego możliwego ciągu progów. □



Rysunek 3.5.4. Omijanie kolumn i wierszy w ciągu progów

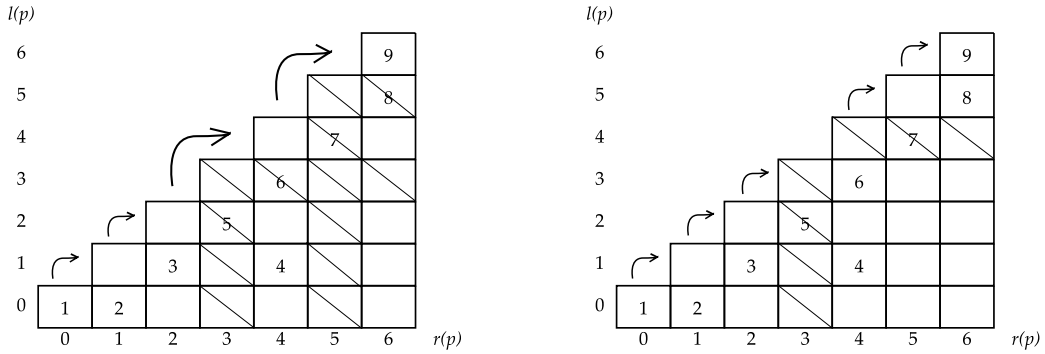
Gdy mamy zbiór takich rozłącznych par, to znaczy podjęta jest decyzja, które kolumny i wiersze mają być omijane przez ciąg progów, to ciąg progów jest jednoznacznie wyznaczony.

Przykład 3.5.5. Dla posetu z rysunku 3.5.6:

(a) Realizowalnym ciągiem progów jest $T_1 = \{(col_0, row_1), (col_1, row_2), (col_2, row_4), (col_4, row_6)\}$ (po lewej). Omijane są kolumny $3, 5, 6$. Omijane są wiersze $0, 3, 5$ (na rysunkach omijane kolumny i wiersze są wykreślone, za wyjątkiem ostatniej kolumny i pierwszego wiersza). Wzdłuż ciągu progów T_1 można odczytać rozszerzenie liniowe $L_1 = (1, 4) \oplus (2, 5) \oplus (3, 7) \oplus (6, 9) \oplus (8)$, $s_{L_1}(P) = 4$, $b_{L_1} = 4 = |T_1|$.

(b) Realizowalnym ciągiem progów jest $T_2 = \{(col_0, row_1), (col_1, row_2), (col_2, row_3), (col_4, row_5), (col_5, row_6)\}$, a zgodnym rozszerzeniem liniowym jest $L_2 = (1, 4) \oplus (2, 5) \oplus (3, 6, 8) \oplus (7, 9)$; $s_{L_2}(P) = 3$, $b_{L_2} = 5 = |T_2|$.

(c) Pełny ciąg progów, nie omijający żadnej kolumny (poza szóstą) i żadnego wiersza (poza zerowym), nie jest realizowalny.



Rysunek 3.5.6. Omijanie kolumn i wierszy w ciągu progów

Jak wykazała Mitas [52], problem liczby skoków redukuje się wielomianowo do wygenerowania realizowalnego ciągu progów o możliwie największej mocy. To znaczy, istnieje algorytm działający w czasie wielomianowym, który dla danego ciągu progów o długości $b_T \leqslant e - 1$ odczytuje rozszerzenie liniowe o co najmniej b_T progach. Fakt ten przytoczony jest w podrozdziale 3.7, sformułowany jako dwa twierdzenia, 3.5.14 i 3.5.15.

Wniosek 3.5.7. *Jeśli istnieje algorytm A generujący realizowalny ciąg progów*

T długości b_T, to używając A można otrzymać rozszerzenie liniowe o $b_A(P) \geqslant b_T$ progach, a więc o $s_A(P) \leqslant |P| - 1 - b_T$ skokach.

Nienasycone składowe grafu przedziałów

W celu scharakteryzowania realizowalnych ciągów progów wprowadza się następujące pojęcia w odniesieniu do tabeli zawierającej kanoniczną reprezentację posetu przedziałowego $(P, <_P)$.

Definicja 3.5.8. Graf przedziałów $G_I(P) = (V, E)$ określamy następująco. Wierzchołkami V są niepuste komórki tabeli. Dwa wierzchołki są połączone krawędzią, jeśli są umiejscowione w tym samym wierszu lub w tej samej kolumnie, i w tym wierszu (odp. w tej kolumnie) nie ma innego wierzchołka pomiędzy nimi.

$$V = \{[row, col] : \text{ istnieje } p \in P \text{ taki, że } l(p) = row \text{ oraz } r(p) = col\},$$

$$E = \{\{[r, c_i], [r, c_j]\} : \text{ nie istnieje } [r, c_m] \in V \text{ taki, że } c_i < c_m < c_j \text{ lub } c_j < c_m < c_i\} \cup$$

$$\{\{[r_i, c], [r_j, c]\} : \text{ nie istnieje } [r_m, c] \in V_P \text{ taki, że } r_i < r_m < r_j \text{ lub } r_j < r_m < r_i\}.$$

Obramowanie R tabeli reprezentującej poset przedziałowy stanowią elementy posetu umiejscowione w dolnym (zerowym) wierszu tabeli, w skrajnej prawej ($(e-1)$-wszej) kolumnie oraz na przekątnej, a także elementy tych komórek tabeli, w których jest więcej niż jeden element.

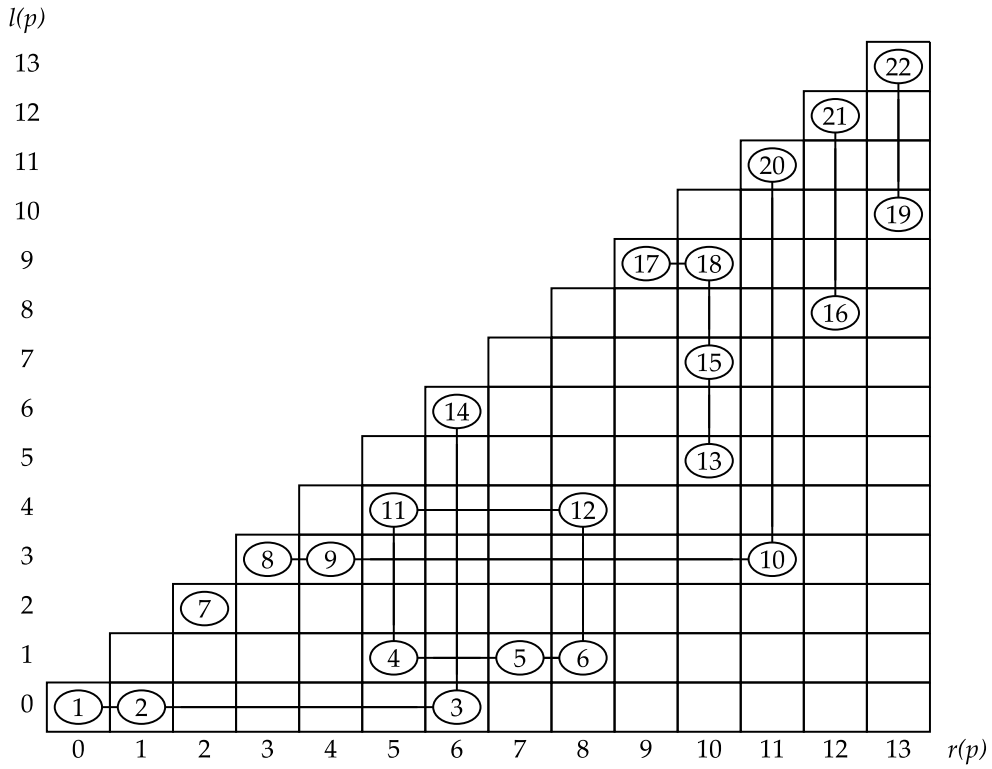
$$R = \{[row, col] \in V : row = col \text{ lub } row = 0 \text{ lub } col = e - 1 \text{ lub istnieją przynajmniej}$$

$$\text{dwa elementy } p, q \in P \text{ takie, że } l(p) = l(q) = row \text{ i } r(p) = r(q) = col\}.$$

Definicja 3.5.9. Składowa C grafu przedziałów G_I jest **nasycona**, jeśli $V_C \cap R \neq \emptyset$ lub jeśli C zawiera cykl. Oznaczmy przez U zbiór nienasyconych składowych grafu G_I, zaś przez $u = |U|$ liczbę nienasyconych składowych.

Przykład 3.5.10. Graf składowych posetu widocznego na rysunku 3.1.7 zawiera trzy nienasycone składowe: $\{3, 4, 6\}$, $\{5\}$, $\{7\}$.

Przykład 3.5.11. Na rysunku 3.5.12 wszystkie składowe grafu przedziałów są nasycone. Jedna z nich zawiera cykl, a wszystkie pozostałe mają wierzchołek na obrzeżu tabeli.



Rysunek 3.5.12. Poset bez nienasyconych składowych

Definicja 3.5.13. Wprowadza się dwie własności nienasyconej składowej C grafu G_I względem danego ciągu progów T.

- ($P1$) Składowa C zawiera wierzchołek położony w kolumnie lub wierszu omijanym przez T.
- ($P2$) Składowa C zawiera element $[j, j+q]$ taki, że kolumny $j, \ldots, j+q-1$ oraz wiersze $j+1, \ldots, j+q$ są omijane przez T.

Mówimy, że składowa pokrywa kolumnę (odp. wiersz), jeśli co najmniej jeden wierzchołek składowej jest umiejscowiony w tej kolumnie (odp. wierszu).

Twierdzenie 3.5.14 (Mitas [52])**.** *Jeśli ciąg progów T jest realizowalny, to każda nienasycona składowa C ma własność $P1$ albo $P2$.*

Dowód. Nienasycona składowa C jest drzewem, skoro nie zawiera cyklu. Korzeń tego drzewa (jakkolwiek wybrany) pokrywa jakiś wiersz i jakąś kolumnę. Każdy nowy

wierzchołek pokrywa nowy wiersz lub nową kolumnę. Zatem $|C|$ wierzchołków składowej C pokrywa $|C|+1$ wierszy i kolumn.

Rozważmy rozszerzenie liniowe L zgodne z ciągiem progów T. Jeśli składowa C nie ma własności $P1$, czyli żadna jej kolumna nie jest omijana przez T ani żaden jej wiersz nie jest omijany przez T, to wszystkie jej kolumny i wiersze są używane w progach T. Zatem któryś element C musi leżeć zarówno w używanej kolumnie, jak i w używanym wierszu, a więc musi występować w dwóch następujących po sobie progach. Niech $[j, j+q]$ będzie tym elementem. W komórce $[j, j+q]$ nie ma innych elementów, bo C jest składową nienasyconą. Dwa progi używające $[j, j+q]$ występują jeden po drugim w L: są to $(j-s, j), (j+q, j+q+t)$ dla jakichś $s, t \geqslant 1$. Dlatego też kolumny $j, \ldots, j+q-1$ oraz wiersze $j+1, \ldots, j+q$ są omijane przez T. Zatem C ma własność $P2$. □

Twierdzenie 3.5.15 (Mitas [52])**.** *Jeśli $T = \{(r_1, l_1), \ldots, (r_b, l_b)\}$ jest ciągiem progów, w którym każda nienasycona składowa C ma własność $P1$ albo $P2$, to istnieje ciąg progów T', w którym używane są wszystkie wiersze używane w T, a więc $|T'| \geqslant |T|$.*

Dowód. W pierwszej części dowodu pokazuje się, że istnieje algorytm etykietujący elementy posetu literami `A` i `B` w tabeli w taki sposób, że

- w każdej kolumnie używanej w T jakiś element otrzymuje etykietę `A`,
- w każdym wierszu używanym w T jakiś element otrzymuje etykietę `B`

i tylko elementy o własności $P2$ mają etykietę `AB`.

W drugiej części dowodu pokazuje się, jak z pomocą tych etykiet odczytać rozszerzenie liniowe posetu reprezentowanego tabelą.

Niech D^+ oznacza zbiór elementów leżących w tabeli na przekątnej, poszerzony o elementy, które mają własność $P2$. Niech R^+ oznacza obramowanie poszerzone o elementy leżące w omijanych wierszach, omijanych kolumnach, a także o elementy mające własność $P2$. Zatem każda składowa C spełnia $V_C \cap R^+ \neq \emptyset$ lub ma cykl.

Algorytm 3.5.16. Algorytm etykietujący literami `A` i `B` elementy posetu w tabeli.

Dla każdej składowej C:

1. Jeśli $V_C \cap R^+ \neq \emptyset$, to wybierz $r \in V_C \cap R^+$. W przeciwnym razie wybierz r z cyklu składowej C tak, by r był incydentny z poziomą krawędzią tego cyklu, po czym usuń tę krawędź.

2. Zbuduj drzewo rozpinające składową C. Niech r będzie korzeniem tego drzewa. Skieruj drzewo „od korzenia".

3. Przydziel etykietę korzeniowi następująco:

(a) Jeśli r jest umiejscowiony w omijanym wierszu, to elementowi z tej komórki przydziel `A`.

(b) Jeśli r jest z cyklu, to elementowi z tej komórki przydziel `A`.

(c) Jeśli w tej komórce są przynajmniej dwa elementy posetu, to jednemu przydziel `A`, a drugiemu `B`.

(d) Jeśli r leży w omijanej kolumnie, to elementowi tej komórki przydziel `B`.

(e) Jeśli $r \in D^+$, to elementowi tej komórki przydziel `AB`.

4. Zaetykietuj pozostałą część składowej:

(a) Jeśli $[i, j]$ ma etykietę i pozioma krawędź drzewa prowadzi z $[i, j]$ do $[i, k]$, to przydziel elementowi $[i, k]$ etykietę `A`.

(b) Jeśli $[i, j]$ ma etykietę i pionowa krawędź drzewa prowadzi z $[i, j]$ do $[k, j]$, to przydziel elementowi $[k, j]$ etykietę `B`.

Pokazujemy poprawność etykietowania uzyskanego powyższym algorytmem.

Rozważmy wiersz i używany w T. Wiersz ten należy do jakiejś składowej C. Jeśli korzeń r tej składowej leży w wierszu i, to:

(a) Jeśli r należy do cyklu, to istnieją inne elementy składowej C na drugim końcu usuniętej krawędzi, a więc jeden z tych elementów jest osiągany w procesie etykietowania pionową krawędzią, toteż dostaje `B`.

(b) Jeśli r leży w komórce zawierającej wielokrotne elementy, to jeden z tych elementów ma `B`.

(c) Jeśli $r \in D^+$, to ma `AB`.

(d) Jeśli r leży w omijanej kolumnie, to ma `B`.

(e) Element w komórce r mógłby otrzymać etykietę `A` jedynie w przypadku, gdyby leżał w omijanym wierszu, ale rozważamy wiersz używany w T

Jeśli korzeń r tej składowej C nie leży w wierszu i, to podczas etykietowania pierwszy element tego wiersza jest osiągany pionową krawędzią, więc otrzymuje `B`.

W analogiczny sposób pokazuje się, że w każdej kolumnie używanej w T znajduje się element oznaczony przez `A`.

Teraz pokazujemy, że wzdłuż realizowalnego ciągu progów $T = \{(r_1, l_1), \ldots, (r_b, l_b)\}$ można skonstruować rozszerzenie liniowe.

Algorytm 3.5.17. Konstrukcja rozszerzenia liniowego.

Wejście: Tabela reprezentująca poset przedziałowy wraz z realizowalnym ciągiem progów.

Wyjście: Rozszerzenie liniowe odczytane wzdłuż wejściowego ciągu progów.

1. Wypisz wszystkie elementy kolumn $0, 1, \ldots, l_1 - 1$.

2. Następnie wypisz jakiś `B`-element wiersza l_1.

3. Wypisz wszystkie elementy kolumn $l_1, \ldots, l_2 - 1$ (nie wypisane wcześniej).

4. Następnie wypisz jakiś `B`-element wiersza l_2.

5. Kontynuuj, aż zostaną wypisane wszystkie elementy kolumn $l_m, \ldots, e - 1$.

Oczywiście w ten sposób otrzymujemy rozszerzenie liniowe. Pozostaje pokazać, że w każdym z wierszy $l_1, \ldots, l_m$ kończy się jakiś próg.

Jeśli w trakcie konstruowania rozszerzenia liniowego nie ma progu wchodzącego w wiersz l_i, to znaczy, że w poprzednim kroku, przed wypisaniem `B`-elementu wiersza l_i, wypisano wszystkie elementy kolumn $l_{i-1}, \ldots, l_i - 1$. To znaczy, że wszystkie te elementy miały etykietę `B`, ale co najmniej jeden element (używanej w T) kolumny r_i ma również `A`.

Rozważmy element $[k, r_i]$ z etykietami `AB`, a więc o własności $P2$. Wówczas wiersze $k + 1, \ldots, r_i$ są omijane przez T, ale z drugiej strony wiersz l_{i-1} jest używany w T, a więc $k \geqslant l_{i-1}$. Ponieważ wypisano do tej pory `B`-elementy tylko z wierszy $l_1, \ldots, l_{i-1}$, to $k = l_{i-1}$ i $[k, r_i]$ jest ostatnim do tej pory wypisanym elementem. Zatem zagwarantowany jest próg do wiersza l_i. □

Podaliśmy twierdzenie Mitas, charakteryzujące realizowalne ciągi progów za pomocą własności $P1$ i $P2$, związanych z omijaniem kolumn i wierszy. Teraz streścimy dalszy ciąg pracy Mitas, w której uzasadnia się $\frac{3}{2}$-aproksymację liczby skoków. W dalszej części pracy poprawiamy współczynnik aproksymacji algorytmu Mitas, a także formułujemy algorytm dokładny, wykorzystujące podane tu rezultaty.

Ogólny schemat algorytmu Mitas przedstawiony jest jako algorytm 3.5.18.

Algorytm 3.5.18. Ogólny schemat algorytmu Mitas dla problemu skoków.

Wejście: Poset przedziałowy $(P, <_P)$.

Wyjście: Rozszerzenie liniowe posetu P.

1. Wygeneruj kanoniczną reprezentację wejściowego posetu z przedziałami w tabeli.

2. Skonstruuj graf przedziałów $G_I(P)$, wyznacz jego spójne składowe i zweryfikuj, które z nich są nienasycone.

3. Wyznacz realizowalny ciąg progów, a więc taki, że każda nienasycona składowa ma własność $P1$ albo $P2$.

4. Odczytaj rozszerzenie liniowe wzdłuż ciągu progów otrzymanego w kroku **3**.

Pierwszy krok został wyjaśniony w podrozdziale 3.1 (twierdzenie 3.1.2 i wynikający z niego algorytm 3.1.4). Drugi krok algorytmu po prostu angażuje definicje grafu przedziałów (3.5.8) i nienasyconych składowych (3.5.9). Sedno problemu leży w kroku trzecim. Należy podjąć decyzję, które rozłączne pary postaci (col_i, row_i) lub (col_i, row_{i+1}) mają być omijane przez ciąg progów. Ta decyzja jednoznacznie wyznacza ciąg progów T. Wtedy, gdy T jest realizowalny, można odczytać rozszerzenie liniowe z tabeli, używając wielomianowego algorytmu (algorytm 3.5.16, a następnie algorytm 3.5.17).

Prosty sposób uzyskania realizowalnego ciągu progów jest sformułowany jako algorytm 3.5.19. Efektem działania tego algorytmu jest ciąg progów, względem którego każda nienasycona składowa ma własność $P1$ i $|T| = e - 1 - u$.

Algorytm 3.5.19. Realizowalny ciąg progów długości $e - 1 - u$.

---

1. Dla każdej nienasyconej składowej C wybierz dowolną kolumnę i_C pokrytą przez C.

2. Wszystkie pary (col_{i_C}, row_{i_C+1}) są omijane. (To znaczy, wygenerowany ciąg progów składa sie ze wszystkich pozostałych par postaci (col_i, row_{i+1})).

Redukcja do pakowania podgrafów

W porównaniu z algorytmem 3.5.19, strata progów byłaby często mniejsza, gdyby wybierać do ominięcia te pary (col_i, row_i) i (col_i, row_{i+1}), które łączą dwie nienasycone składowe, to znaczy takie, że kolumna col_i jest pokryta jedną składową, a wiersz row_i bądź row_{i+1} jest pokryty inną składową.

Definicja 3.5.20. W **grafie nienasyconych składowych** $G_U(P)$ wierzchołki odpowiadają nienasyconym składowym grafu przedziałów $G_I(P)$, to znaczy dla każdej nienasyconej składowej $C \in U$ dodajemy wierzchołek v_C. Krawędzią łączymy dwa wierzchołki v_C i v_D dokładnie wtedy, gdy nienasycona składowa C pokrywa kolumnę i, zaś D pokrywa wiersz i lub wiersz $i+1$.

Znajdując najliczniejsze skojarzenie w G_U, na ogół otrzymujemy dłuższy ciąg progów niż przy użyciu powyższej, prostszej metody, opisanej algorytmem 3.5.19. Dla każdej skojarzonej krawędzi ciąg progów omija parę (col_i, row_i) lub (col_i, row_i) stowarzyszoną z tą krawędzią. Dla każdego wierzchołka v_C spoza znalezionego skojarzenia, wybieramy do ominięcia dowolną z kolumn składowej C, powiedzmy i_C, oraz wiersz $j \geqslant i_C$, najmniejszy o tej własności nie wybrany do tej pory. Co ciekawe, ewentualna własność $P2$ związana jest z występowaniem nieparzystych cykli w grafie G_U. Poniżej podsumowujemy własności konstrukcji G_U.

Obserwacja 3.5.21. *Następujące fakty dotyczą konstrukcji grafu nienasyconych składowych G_U:*

1. Jeśli w G_U istnieje krawędź łącząca wierzchołki v_C i v_D, to odpowiadające składowe C i D mają własność $P1$, gdy jedna para (col_i, row_i) lub (col_i, row_{i+1}) jest omijana przez ciąg progów.

*2. Jeśli istnieje nienasycona składowa C z elementem $[j, j+q]$ takim, że każda z kolumn $j, \ldots, j+q-1$ i każdy z wierszy $j+1, \ldots, j+q$ zawiera wierzchołek z innej nienasyconej składowej, to wszystkie te $2q$ składowych mają własność $P1$, a C ma własność $P2$, gdy wszystkie q par $(col_j, row_{j+1}), \ldots, (col_{j+q-1}, row_{j+q})$ są omijane przez ciąg progów. Wówczas te $2q+1$ składowych stanowią nieparzysty cykl w G_U. Nazywamy takie cykle $P2$-**cyklami**.*

Lemat 3.5.22. *Jeśli T jest realizowalnym ciągiem progów, w którym element $[j, j+q] \in C$, $C \in U$ ma własność $P2$, a pod kolumnami $j, \ldots, j+q-1$ i wierszami $j+1, \ldots, j+q$ jest mniej niż $2q$ różnych nienasyconych składowych, to T można zmienić tak, że liczba omijanych par nie wzrośnie, a rozważane tu nienasycone składowe będą mieć własność $P1$.*

Dowód. Jeśli w którejś z kolumn $k \in \{j, \ldots, j+q-1\}$ jest wierzchołek składowej nasyconej lub takiej składowej nienasyconej, która ma drugi wierzchołek wśród rozważanych wierszy i kolumn, to kolumnę k używamy w ciągu progów T, zaś omijamy kolumnę $j+q$. Analogicznie, jeśli w któryms z wierszy $w \in \{j+1, \ldots, j+q\}$ jest wierzchołek składowej nasyconej lub takiej składowej nienasyconej, która ma drugi wierzchołek wśród rozważanych wierszy i kolumn, to wiersz w używamy w ciągu progów T, zaś omijamy wiersz j.

W efekcie nadal omijamy $2q$ rozłącznych par (col_i, row_i) lub (col_i, row_{i+1}), ale wszystkie składowe z kolumn $j, \ldots, j+q-1$, z wierszy $j+1, \ldots, j+q$ i składowa C zawierająca $[j, j+q]$ mają własność $P1$. □

Dzięki lematowi 3.5.22, szukając optymalnych ciągów progów wykorzystujących własność $P2$, wystarczy skupić się na takich układach nienasyconych składowych, w których wokół elementu $[j, j+q]$ każda z kolumn $j, \ldots, j+q-1$ i każdy z wierszy $j+1, \ldots, j+q$ zawiera wierzchołek z innej nienasyconej składowej.

Prawdziwy jest też następujący wniosek 3.5.23.

Wniosek 3.5.23 (Mitas [52])**.** *Dla dowolnego posetu przedziałowego P, liczba $P2$-cykli w grafie nienasyconych składowych $G_U(P)$ jest ograniczona przez rozmiar tabelki reprezentującej P, tj. liczbę e.*

Reasumując, problem skoków na posetach przedziałowych jest rozwiązywany przez Mitas jako uogólnienie problemu skojarzenia.

Wniosek 3.5.24 (Mitas [52])**.** *Problem liczby skoków dla posetów przedziałowych redukuje się do pakowania wierzchołkowo rozłącznych krawędzi i nieparzystych cykli w grafie nienasyconych składowych G_U.*

Dowód. Jeśli wyznaczymy zbiór wierzchołkowo rozłącznych krawędzi i $P2$-cykli w G_U, to otrzymamy zbiór rozłącznych par typu (col_i, row_i) lub (col_i, row_{i+1}) taki,

że każda nienasycona składowa będzie miała własność $P1$ albo $P2$ w ciągu progów omijającym wszystkie te pary. □

Poniżej wyjaśniamy (obserwacja 3.5.26), w jaki sposób otrzymana redukcja przekłada się na liczbę skoków otrzymywanych rozszerzeniach liniowych.

Dolne ograniczenie i aproksymacja Mitas

Teraz przedstawiamy algorytm aproksymacyjny Mitas, oparty na faktach podanych w poprzednich podrozdziałach. W dalszej części pracy prezentujemy usprawnienia tego algorytmu, prowadzące do mniejszego współczynnika aproksymacji.

Algorytm oparty jest na następującym dolnym ograniczeniu liczby skoków.

Twierdzenie 3.5.25 (Mitas [52])**.** *Niech P będzie posetem przedziałowym. Wówczas $b(P) \leqslant e - 1 - \frac{u}{3}$.*

Dowód. Niech L będzie rozszerzeniem liniowym P odczytanym wzdłuż realizowalnego ciągu progów T. Niech U_1 oznacza zbiór nienasyconych składowych o własności $P1$. Niech $U_2 = U - U_1$. Oznaczmy $u_1 = |U_1|$, $u_2 = |U_2|$. Zauważmy, że U_2 na mocy twierdzenia 3.5.14 jest zbiorem składowych o własności $P2$.

(a) Pokazujemy, że $b_L(P) \leqslant e - 1 - \frac{u_1}{2}$.

Policzmy wiersze i kolumny, które są omijane przez T. Dla każdej nienasyconej składowej $C \in U_1$ przynajmniej jeden wiersz lub kolumna jest omijana. Ponadto omijane są: wiersz 0 i kolumna $e - 1$. Dlatego co najwyżej $2e - 2 - u_1$ wierszy jest używanych w T. Stąd $b_L(P) \leqslant e - 1 - \frac{u_1}{2}$.

(b) Teraz pokazujemy, że $b_L(P) \leqslant e - 1 - u_2$.

Składowa $C \in U_2$ ma własność $P2$. Niech $[j, j+q_C]$ będzie elementem, wokół którego kolumny $j, \ldots, j+q_C-1$ i wiersze $j+1, \ldots, j+q_C$ są omijane, zaś wiersz j oraz kolumna $j+q_C$ są używane w T, skoro C nie ma własności $P1$. Łatwo widać, że omijane kolumny stowarzyszone z C nie nachodzą na omijane kolumny związane z inną składową z U_2 (analogicznie, omijane wiersze stowarzyszone z C nie nachodzą na omijane wiersze związane z inną składową z U_2). Zatem przynajmniej $1 + \sum_{C \in U_2} q_C \geqslant 1 + u_2$ kolumn jest omijanych, a więc $b_L(P) \leqslant e - 1 - u_2$.

Korzystając z tych dwóch faktów dowodzimy tezy twierdzenia. Niech L będzie rozszerzeniem liniowym posetu przedziałowego P. Jeśli $u_2 \geqslant \frac{u}{3}$, to $b_L(P) \leqslant e - 1 - u_2 \leqslant e - 1 - \frac{u}{3}$.

Jeśli natomiast $u_2 < \frac{u}{3}$, to $u_1 = u - u_2 > \frac{2}{3}u$, i $b_L(P) \leqslant e - 1 - \frac{u_1}{2} < e - 1 - \frac{u}{3}$. $\square$

Z drugiej strony, na mocy algorytmu 3.5.19, mamy ograniczenie $b(P) \geqslant e - 1 - u$. Innymi słowy, prawdziwe są ograniczenia na liczbę skoków:

(a) $s(P) \leqslant |P| - e + u$,

(b) $s(P) \geqslant |P| - \frac{u}{3}$.

Wprowadzamy teraz dalsze oznaczenia, użyteczne w dyskusji o algorytmach optymalizujących ciągi progów.

Dla ciągu progów długości $|T|$ jego **koszt** jest liczbą z_T taką, że $e - 1 - z_T = |T|$, to znaczy jest liczbą omijanych kolumn, lub liczbą omijanych wierszy. Na mocy twierdzenia 3.5.25, $z_T \geqslant \frac{u}{3}$, a z algorytmu 3.5.19 wynika, że możemy ograniczyć uwagę do ciągów progów o koszcie $z_T \leqslant u$. Dla ciągu progów T określamy też **oszczędność**, czyli liczbę $w_T = u - z_T$. Naszym zadaniem jest szukanie realizowalnych ciągów progów o koszcie z_T pomiędzy $\frac{u}{3}$ a u, oraz o oszczędności w_T pomiędzy 0 a $\frac{2}{3}u$. Naturalnie, celem projektowynych algorytmów jest minimalizacja kosztu, czyli maksymalizacja oszczędności.

Jak wiemy z wniosku 3.5.24, problem znalezienia optymalnego ciągu progów redukuje się do wyznaczenia optymalnego upakowania wierzchołkowo rozłącznych krawędzi i cykli nieparzystych w grafie nienasyconych składowych G_U. Koszt oraz oszczędność ciągu progów zależą zarówno od liczby wierzchołków w upakowaniu, jak i od liczby cykli w upakowaniu.

Obserwacja 3.5.26. *Szczegóły redukcji opisanej we wniosku* 3.5.24 *są następujące:*

- *Dla każdej krawędzi skojarzenia ciąg progów T omija jedną parę (col_i, row_i) bądź (col_i, row_{i+1}), zatem koszt z_T wzrasta o* 1*, zaś oszczędność w_T również wzrasta o* 1*.*

- *Dla cyklu na $2q + 1$ wierzchołkach, T omija q par $(col_i row_{i+1})$, zatem koszt z_T wzrasta o q, zaś oszczędność w_T wzrasta o $q + 1$.*

- *Dla wierzchołka nie uwzględnionego w upakowaniu koszt z_T wzrasta o* 1 *(analogicznie jak w algorytmie* 3.5.19*).*
- *Ogólnie, upakowanie zajmujące v wierzchołków oraz c cykli przekłada się na ciąg progów o oszczędności $w_T = \frac{v+c}{2}$ oraz o koszcie $z_T = u - \frac{v+c}{2}$.*

Mówiąc o upakowaniu, przekładamy bezpośrednio pojęcia oszczędności i kosztu, ponieważ jest to dokładnie informacja o liczbie zaoszczędzonych, odpowiednio traconych progów.

Teraz formułujemy algorytm $\frac{3}{2}$-aproksymacyjny dla problemu liczby skoków posetu przedziałowego, podany przez Mitas.

Cornuéjols, Hartvigsen i Pulleyblank [15] podali algorytm dla problemu pakowania wierzchołkowo rozłącznych krawędzi i cykli nieparzystych, maksymalizujący liczbę wierzchołków w upakowaniu, działający w czasie wielomianowym. Ogólniej, upakowanie może składać się z dowolnego podzbioru krawędzi grafów oraz skończonej rodziny grafów podskojarzalnych, wielomianowego rozmiaru. Wówczas maksymalizacja liczby wierzchołków w upakowaniu jest obliczalna w czasie wielomianowym. Rezultat ten przytoczony jest w podrozdziale 1.3. Mitas zaproponowała użycie tego algorytmu do poszukiwania ciągów progów, to znaczy do przybliżonej maksymalizacji oszczędności omijanych kolumn i wierszy.

Główna część algorytmu Mitas dla problemu skoków, czyli krok **3** schematu 3.5.18, jest sformułowany jako algorytm 3.5.27.

Algorytm 3.5.27. Aproksymacja ciągu progów za pomocą pakowania podgrafów.

---

1. Wygeneruj graf nienasyconych składowych G_U.

2. Wyznacz upakowanie krawędzi i $P2$-cykli o największej liczbie wierzchołków.

3. Wygeneruj ciąg progów, który omija wszystkie pary postaci (col_i, row_i) lub (col_i, row_{i+1}) stowarzyszone z krawędziami i cyklami znalezionymi w kroku **1**.

Twierdzenie 3.5.28 (Mitas [52])**.** *Algorytm* 3.5.27 *jest algorytmem $\frac{3}{2}$-aproksymacyjnym dla problemu minimalizacji liczby skoków posetu przedziałowego.*

Dowód. Dla optymalnego rozszerzenia liniowego istnieje optymalne upakowanie wierzchołkowo rozłącznych krawędzi i cykli nieparzystych, czyli upakowanie maksyma-

lizujące oszczędność. Oszczędność jest równa $w_O = \frac{v_O+c_O}{2} \leqslant \frac{v_O+\frac{1}{3}v_O}{2}$, ponieważ wśród v_O wierzchołków stanowiących upakowanie, jest co najwyżej $\frac{1}{3}v_O$ cykli w upakowaniu.

Liczba skoków jest równa $s_O = |P| - e + u - w_O$.

Mamy również: $w_O \leqslant \frac{u+\frac{u}{3}}{2} = \frac{2}{3}u$ (co najwyżej u wierzchołków w upakowaniu, a więc w tym co najwyżej $\frac{u}{3}$ cykli).

Algorytm aproksymacyjny wyznacza upakowanie obejmujące v_A wierzchołków. Zauważmy, że $v_A \geqslant v_O$. Mamy więc ograniczenie na oszczędność:

$$w_O \leqslant \frac{v_O + \frac{1}{3}v_O}{2} \leqslant \frac{v_A + \frac{1}{3}v_A}{2} = \frac{2}{3}v_A.$$

Z drugiej strony, algorytm oszczędza przynajmniej $w_A \geqslant \frac{v_A}{2}$.

Podsumowując:

$$\frac{s_A(P)}{s_O(P)} \leqslant \frac{|P| - e + u - w_A}{|P| - e + u - w_O} = 1 + \frac{w_O - w_A}{|P| - e + u - w_O} \leqslant 1 + \frac{w_O - w_A}{\frac{1}{2}w_O}.$$

A skoro $\frac{w_A}{w_O} \geqslant \frac{3}{4}$, mamy $\frac{s_A(P)}{s_O(P)} \leqslant 3 - 2\frac{w_A}{w_O} \leqslant \frac{3}{2}$. □

Algorytm Mitas – doświadczenia obliczeniowe

W tabeli 3.5.29 pokazano wydajność algorytmu Mitas na losowych posetach przedziałowych wygenerowanych z rozkładów jednostajnego i wykładniczego. Średnie odchylenie od optymalnej wartości jest bardzo niskie, nawet niższe, niż w przypadku algorytmów Felsnera i Sysły. Wartości widoczne w tej tabeli zostały uzyskane prostszą metodą, angażującą najliczniejsze skojarzenie zamiast upakowania krawędzi i cykli.

Tabela 3.5.29. Średni błąd rozwiązań generowanych algorytmem Mitas na losowych posetach przedzialowych różnego rozmiaru

Rozkład jednostajny				
próbka	n_{10-20}	n_{20-30}	n_{30-50}	n_{50-100}
10	0,0000	0,0000	0,0000	0,0000
100	0,0053	0,0023	0,0005	0,0000
1000	0,0028	0,0021	0,0007	0,0000
10000	0,0034	0,0023	0,0007	0,0001

Rozkład wykładniczy		
próbka	n_{10-40}	n_{40-100}
10	0,0000	0,0000
100	0,0000	0,0000
1000	0,0001	0,0000
10000	0,0001	0,0000

Szukaliśmy też trudnych instancji posetów, które wymuszają istotny błąd algorytmu Mitas. Na udowodniony współczynnik aproksymacji silny wpływ ma ułożenie elementów posetu w tabeli, ponieważ główna część obliczeń angażuje tylko graf składowych. Zatem trudne instancje dla procedury Mitas mają względnie małą szerokość. Zostało zaobserwowane podczas naszych doświadczeń, ale jest również sugerowane przez dowód współczynnika aproksymacji, że wyniki moga być odległe od optimum zwłaszcza wtedy, gdy graf składowych zawiera dużą liczbę nieparzystych cykli, które mogą utworzyć optymalne upakowanie. Poset na rysunku 3.8.3 jest małym przykładem: jego sześć nienasyconych składowych $\{4\}$, $\{6\}$, $\{7\}$, $\{9\}$, $\{11\}$ i $\{12\}$ stanowi graf składowych, w którym dwa trójkąty $\{4, 5, 7\}$ i $\{9, 11, 12\}$ mogłyby utworzyć upakowanie. Jednakże ten graf ma najliczniejsze skojarzenie nasycające wszystkie wierzchołki. Te dwa rozwiązania (tj. rodzina niezależnych trójkątów oraz rodzina niezależnych krawędzi, gdzie każda z rodzin angażuje tyle samo wierzchołków) są nierozróżnialne dla algorytmu pakowania Cornuéjolsa i in. [15] (podrozdział 1.3). Jest jasne, że da się skonstruować poset przedziałowy dowolnego rozmiaru, w którym graf składowych powiela ten schemat. Zatem można utworzyć dowolnie duże trudne przykłady dla tego algorytmu, skoro optymalne upakowanie trójkątów zużywa do $\frac{u}{3}$ wierzchołków, tak że 1 próg jest tracony za każde trzy nienasycone składowe, ale jeśli wygenerowane zostanie skojarzenie, to $\frac{u}{2}$ progów jest tracone na cały graf.

Widać, że algorytm wypada najgorzej wtedy, gdy graf nienasyconych składowych zawiera upakowanie o dużej liczbie $P2$-trójkątów.

3.6. Porównanie algorytmów zachłannych z algorytmem Mitas

Można zobaczyć w tabeli 3.6.1, że algorytmy Mitas i Sysły są niemal porównywalne, gdy stosuje się je do posetów losowych. Podobnie jak w podrozdziale 3.4, zw_M oznacza liczbę posetów, na których algorytm Mitas zwrócił mniejszą liczbę skoków niż algorytm Sysły, spośród 1000 losowych posetów przedziałowych rozmiaru n, dla $n = 10, 20, \ldots, 100$. Tabela 3.6.1 zawiera tylko wyniki wariantu skojarzeniowego, gdyż pakowanie cykli dostarcza podobnych wyników na losowych posetach. Jednakże, testy angażujące trudne posety (tabele 3.6.2 i 3.6.3) wskazują, że silniejsza metoda zaproponowana przez Mitas może często dostarczać ściślejszych wyników. Półsilnie zachłanne

rozszerzenia liniowe okazują się lepsze na tych posetach, które były sklasyfikowane jako trudne instancje dla algorytmu Mitas (tabela 3.6.3), mimo, że użyto tutaj wariantu z pakowaniem podgrafów. Te posety, gdy widziane w reprezentacji tabelowej, są wąskie (rysunki 3.7.9 i 3.8.3), a więc diagram łukowy zawiera względnie małą liczbę ścieżek zachłannych w kolejnych krokach algorytmu Sysły.

Tabela 3.6.1. Porównanie algorytmów Sysły i Mitas na losowych posetach przedziałowych

Rozkład jednostajny			
n	zw_S	zw_M	*remisy*
10	0	1	999
20	3	10	987
30	3	34	963
40	2	39	956
50	3	65	932
60	1	73	926
70	1	102	897
80	1	109	890
90	2	110	888
100	0	115	885

Rozkład wykładniczy			
n	zw_S	zw_M	*remisy*
10	0	1	999
20	2	9	989
30	0	12	988
40	2	28	970
50	3	41	956
60	3	53	944
70	3	73	924
80	2	60	938
90	2	92	906
100	1	96	903

Tabela 3.6.2. Porównanie algorytmów na dużych posetach złożonych z trudnych instancji przechowywanych w bazie danych

	alg. Sysły vs alg. Felsnera			*alg. Sysły vs alg. Mitas*		
instancje wejściowe	zw_F	zw_S	*remisy*	zw_S	zw_M	*remisy*
1000 posetów, $n \approx 100 - 200$ trudne dla alg. zachłannych	0	1000	0	0	938	62
1000 posetów $n \approx 100 - 200$ trudne dla alg. Mitas	0	158	842	264	126	610

Tabela 3.6.3. Średni błąd rozwiązań generowanych algorytmami na trudnych instancjach

instancje wejściowe	*rozmiar próbki*	*Felsner*	*Sysło*	*Mitas (s.)*	*Mitas (p.)*
posety rozmiaru $\approx 50 - 100$,	10	1.23	1.18	1.04	1.01
trudne dla alg. zachłannych	100	1.24	1.19	1.05	1.01
posety rozmiaru $\approx 50 - 100$,	10	1.02	1.01	1.38	1.02
trudne dla alg. Mitas	100	1.01	1.00	1.37	1.06

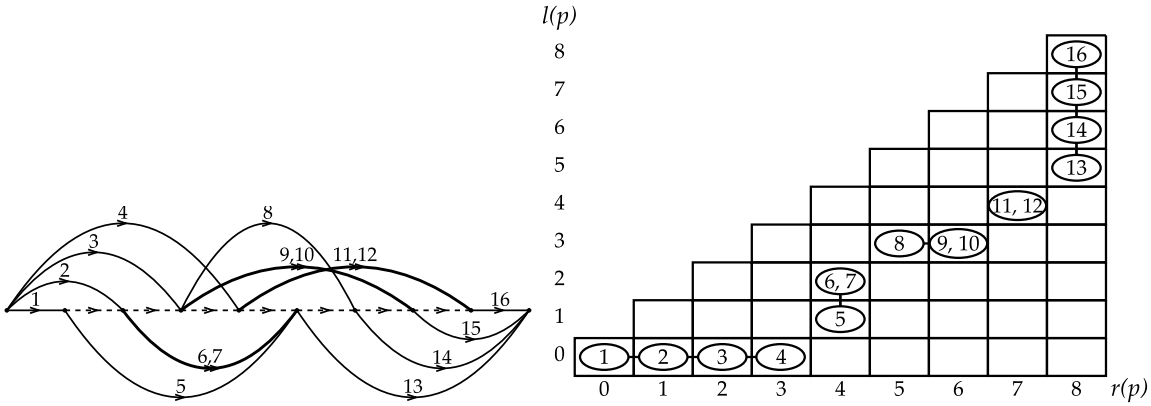
Szczególnie dobrze znaną rodziną posetów, dla których algorytmy zachłanne generują optymalne rozszerzenie liniowe, są posety N-wolne, wprowadzone w podrozdziale 2.2 (twierdzenie 2.2.5). W reprezentacji za pomocą diagramów łukowych te posety nie zawierają łuków pozornych [65]. W efekcie, algorytm Sysły buduje rozszerzenia liniowe złożone tylko z silnie zachłannych ścieżek (a więc optymalne rozszerzenie liniowe). Zaobserwujmy, że algorytm Mitas nie jest gorszy w tym aspekcie. W konstrukcji diagramu łukowego dla posetów przedziałowych (algorytm 3.3.1), łuki pozorne są wstawiane w miejscu brakujących przedziałów długości jeden, $(i, i+1)$ wzdłuż osi. Gdy myślimy o nich w terminach tabeli, te przedziały są długości zero; to znaczy, są umiejscowione na przekątnej w tabeli. Jeśli diagram nie ma łuków pozornych, to jego oś jest pełna łuków ciągłych, odpowiadających elementom posetu. Reprezentacja tabelowa takich posetów z kolei nie ma pustych komórek na przekątnej. Zatem bez względu na rozmieszczenie pozostałych elementów w tabeli, każda ze składowych grafu przedziałów zawiera wierzchołek na przekątnej, więc nie ma nienasyconych składowych. W efekcie algorytm Mitas generuje rozwiązanie optymalne. W ten sposób udowodniliśmy następujący lemat 3.6.4.

Lemat 3.6.4. *Jeśli P jest przedziałowym posetem N-wolnym, to algorytm Mitas generuje optymalne rozszerzenie liniowe dla P w czasie wielomianowym.*

Lemat 3.6.4 oznacza, że algorytmy Felsnera i Sysły na posetach N-wolnych nie mają przewagi nad algorytmem Mitas, mimo, że algorytmy zachłanne generują na takich posetach rozwiązania optymalne.

Przypadek, gdy graf przedziałów nie zawiera nienasyconych składowych, jest bardziej ogólny. Rysunek 3.6.5 zawiera poset przedziałowy tego rodzaju. Algorytm Mitas generuje optymalne rozszerzenie liniowe, bez pomocy ani skojarzenia, ani upakowania. Algorytm Sysły może błędnie wybrać ścieżkę półsilnie zachłanną i zwrócić rozwiązanie

nieoptymalne. Po zredukowaniu ścieżek $(1,5)$ i $(2,6)$ staniemy przed wyborem: $(3,8)$ czy $(3,9)$? Wybór $(3,8)$ prowadzi do rozszerzeń liniowych zawierających 8 skoków, podczas gdy $s(P) = 7$, np. $(1,5) \oplus (2,6) \oplus (3,9) \oplus (4,11) \oplus (7,13) \oplus (8,14) \oplus (10,15) \oplus (12,16)$.



Rysunek 3.6.5. Półporządek, w którym wszystkie składowe są nasycone

3.7. Poprawa aproksymacji do współczynnika 1.484

W tym podrozdziale przedstawiony jest najważniejszy wynik rozprawy. Prezentowane tu twierdzenia zostały opublikowane w artykule [44], a także omówione podczas międzynarodowej konferencji *Combinatorics 2012* w Perugii [45].

W latach 90-tych ustanowiono trzy podobne rezultaty, dowodzące wielomianowej aproksymacji liczby skoków dla posetów przedziałowych. Każdy z tych algorytmów ma współczynnik równy $\frac{3}{2}$. Teraz pokazujemy, że podejście Mitas daje się usprawnić dzięki zmianie podejścia algorytmicznego: zamiast pakować podgrafy, można szukać najtańszego pokrycia zbioru. Otrzymany poprawiony współczynnik aproksymacji jest wnioskiem z analizy trudnych przypadków dla algorytmu Mitas.

Grafy nienasyconych składowych o małej liczbie $P2$-trójkątów

Bezpośrednim wnioskiem z twierdzenia 3.5.28 o aproksymacji jest ściślejszy współczynnik aproksymacji algorytmu Mitas (3.5.27) w przypadku, gdy w grafie nienasyconych składowych nie da się upakować aż $\frac{1}{3}u$ wierzchołkowo rozłącznych trójkątów. Wówczas w optymalnym upakowaniu mamy nieco mniej trójkątów, a więc ograniczenie na zaoszczędzone progi jest nieco ściślejsze.

Wniosek 3.7.1. *Niech $\tau \in [0,1]$ będzie ustaloną liczbą wymierną, P posetem przedziałowym. Załóżmy, że liczba wierzchołków w jakimkolwiek upakowaniu wierzchołkowo rozłącznych $P2$-trójkątów w grafie nienasyconych składowych $G_U(P)$ jest równa co najwyżej τu. Wówczas współczynnik aproksymacji algorytmu* 3.5.27 *zastosowany do P jest równy $1 + \frac{2\tau+3}{12-2\tau}$.*

Dowód. W optymalnym (pod względem oszczędności) upakowaniu jest co najwyżej τu wierzchołków upakowanych w trójkątach, czyli co najwyżej $\frac{1}{3}\tau u$ trójkątów. Pozostałe wierzchołki grafu mogą być upakowane w pięciokątach lub większych cyklach, lub w krawędziach, lub w ogóle. Zauważmy, że pakowanie większych cykli sprawia, że liczba cykli w upakowaniu maleje.

Zatem w optymalnym upakowaniu jest co najwyżej $\frac{1}{3}\tau u + \frac{1}{5}(1-\tau)u$ wierzchołkowo rozłącznych cykli, a więc

$$w_O \leqslant \frac{u + \frac{1}{3}u + \frac{1}{5}(1-\tau)u}{2}.$$

Podobnie, algorytm optymalny zaoszczędza co najwyżej

$$w_O = \frac{v_O + c_O}{2} \leqslant \frac{v_O + \frac{1}{3}\tau u + \frac{1}{5}(1-\tau)u}{2}.$$

Jak poprzednio, $v_O \leqslant v_A$. Niemniej, mamy teraz silniejsze ograniczenie od poprzedniego. Mamy też ponownie $w_A \geqslant \frac{1}{2}v_A$.

Podstawmy $r = \frac{2\tau+18}{30}$, $s = \frac{2\tau+3}{30}$. Nasze ograniczenia można przepisać jako:

- $w_O \leqslant ru = (s + \frac{1}{2})u$,
- $w_O \leqslant \frac{1}{2}v_A + su$,
- $w_A \geqslant \frac{1}{2}v_A$.

Zatem:

$$w_O - w_A \leqslant w_O - \frac{1}{2}v_A \leqslant \frac{1}{2}v_A + su - \frac{1}{2}v_A = su$$

oraz

$$u - w_O \geqslant u - ru = u(1-r),$$

więc błąd jest równy co najwyżej

$$\frac{w_O - w_A}{u - w_O} \leqslant \frac{su}{u(1-r)} = \frac{s}{1-r} = \frac{2\tau + 3}{12 - 2\tau}.$$

□

W następnym punkcie zajmiemy się przeciwnym przypadkiem, gdy w grafie nienasyconych składowych można upakować więcej $P2$-trójkątów niż $\frac{1}{3}\tau u$.

Grafy nienasyconych składowych o dużej liczbie $P2$-trójkątów

Teraz uzupełniamy podejście Mitas, stosując inny algorytm. Pokazujemy, że problem skoków w pewnych sytuacjach warto jest widzieć jako problem pokrycia zbioru.

W tym celu najpierw dowodzimy, że zamiast zbioru parami rozłącznych par postaci (col_i, row_i) i (col_i, row_{i+1}), omijanych przez ciąg progów, możemy dopuścić nachodzące się pary tej postaci.

Lemat 3.7.2. *Zbiór par postaci (col_i, row_i) lub (col_i, row_{i+1}) można przekształcić w zbiór parami rozłącznych takich par, który ma nie większą moc, i taki, że wszystkie wiersze i kolumny w zbiorze wejściowym są w nim zawarte.*

W efekcie można poszukiwać krawędzi i cykli (a także pojedynczych wierzchołków) w grafie nienasyconych składowych, które się nachodzą. Otrzymamy być może nachodzące się pary powyższej postaci, ale łatwo przekształcimy je w parami rozłączne pary, a więc uzyskamy prawidłowy ciąg progów. W ten sposób problem skoków jest redukowany do problemu pokrycia zbioru: wyznaczyć najtańszy zbiór krawędzi, nieparzystych cykli i wierzchołków, pokrywających cały graf nienasyconych składowych. Koszty zbiorów, którymi należy pokrywać, pozostają zgodne z liczbą omijanych przez ciąg progów par: reprezentują liczbę par (col_i, row_i) lub (col_i, row_{i+1}) omijanych przez ciąg progów w przypadku wybrania danej krawędzi, cyklu lub wierzchołka (zgodnie z obserwacją 3.5.26).

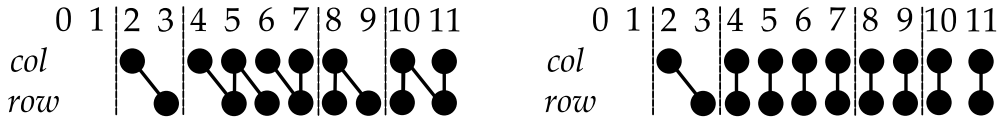
Dowód. Pary wejściowe widzimy jako paski pionowe (col_i, row_i) lub paski skośne (col_i, row_{i+1}), prowadzące z wiersza górnego do wiersza dolnego, jak na rysunku 3.7.3. Chcemy przekształcić zbiór takich pasków w taki sposób, żeby uzyskać rozłączne paski,

to znaczy dla każdych dwóch pasków (col, row), (col', row') ma być $col \neq col'$ oraz $row \neq row'$. Dla każdego i, jeśli nie ma paska skośnego (col_i, row_{i+1}), to zbiór pasków dzielimy. Otrzymujemy spójne podzbiory pasków i każdy z nich rozpatrujemy z osobna.

Spójny zbiór pasków zaczyna się albo paskiem pionowym, albo skośnym. Podobnie, kończy się albo paskiem pionowym, albo skośnym. Rozważmy wszystkie przypadki.

- Jeśli zaczyna się pionowo, (col_i, row_i) i kończy się pionowo, $(col_{i+k}, row_{i+k}), k \geqslant 0$, to układamy zwarty ciąg $k+1$ pasków pionowych.
- Jeśli zaczyna się ukośnie, (col_i, row_{i+1}) i kończy się ukośnie, (col_{i+k}, row_{i+k+1}), to układamy zwarty ciąg $k+1$ pasków ukośnych.
- Jeśli zaczyna się pionowo, (col_i, row_i), a kończy się skośnie, (col_{i+k}, row_{i+k+1}), to układamy zwarty ciąg $k+2$ pasków pionowych, od (col_i, row_i) do $(col_{i+k+1}, row_{i+k+1})$ włącznie.
- Jeśli zaczyna się skośnie, (col_i, row_{i+1}), a kończy się pionowo, $(col_{i+k+1}, row_{i+k+1})$, to układamy zwarty ciąg $k+2$ pasków pionowych, od (col_i, row_i) do $(col_{i+k+1}, row_{i+k+1})$ włącznie.

Można łatwo zobaczyć, że otrzymany zbiór jest nie większy pod względem mocy od zbioru wejściowego, że wszystkie wiersze i kolumny incydentne z pierwotnymi paskami są również incydentne z nowymi paskami, oraz że nowe paski się nie nachodzą (są parami rozłączne). Zauważmy, że w trzecim przypadku, skoro mamy spójny zbiór pasków, to jest ich przynajmniej $k+1$, a więc z ostatnim, pionowym paskiem (col_i, row_i) mamy na wejściu co najmniej $k+2$ pasków, i tyle kładziemy na wyjściu. W dodatku, nie ma skośnego paska $(col_{i+k+1}, row_{i+k+2})$ na końcu ciągu, więc pionowy pasek $(col_{i+k+1}, row_{i+k+1})$ nie nachodzi na następny spójny zbiór. Czwarty przypadek dowodu jest analogiczny. □



Rysunek 3.7.3. Przykładowy zbiór par postaci (col_i, row_i) i (col_i, row_{i+1}), przed i po przekształceniu.

Wniosek 3.7.4. *Jeśli istnieje algorytm A znajdujący pokrycie grafu G_U o koszcie z, to przekształcając jego wynik zgodnie z lematem* 3.7.2 *otrzymamy realizowalny ciąg progów o długości przynajmniej $e-1-z$, a więc i rozszerzenie liniowe o przybliżonej liczbie skoków $s_A(P) \leqslant |P| - e + z$.*

W celu poprawienia współczynnika aproksymacji dla problemu skoków, wystarczy ograniczyć rozważania do przypadków komplementarnych niż rozważane w poprzednim punkcie. To znaczy, interesują nas teraz takie grafy nienasyconych składowych, na których istnieje upakowanie dużej liczby wierzchołkowo rozłącznych trójkątów. Formułujemy więc algorytm 3.7.5. Stosujemy w nim algorytm $\frac{4}{3}$-aproksymacyjny dla najtańszego 3-pokrycia zbioru, podany przez Duh-a i Fürer-a [25], przytoczony w podrozdziale 1.3. To znaczy, w grafie składowym interesują nas pojedyncze wierzchołki, krawędzie i $P2$-trójkąty.

Algorytm 3.7.5. Aproksymacja ciągu progów za pomocą 3-pokrycia zbioru.

1. Wygeneruj instancję problemu 3-pokrycia zbioru na G_U.
2. Zastosuj algorytm $\frac{4}{3}$-aproksymacyjny do otrzymanej instnacji.
3. Zastosuj przekształcenie opisane w dowodzie lematu 3.7.2 tak, by uzyskane pary postaci (col_i, row_i) or (col_i, row_{i+1}) były rozłączne.
4. Wygeneruj ciąg progów, który omija wszystkie te pary.

Lemat 3.7.6. *Niech $\tau \in [0,1]$ będzie ustaloną liczbą wymierną, P posetem przedziałowym. Załóżmy, że największa liczba wierzchołków w jakimś upakowaniu wierzchołkowo rozłącznych $P2$-trójkątów w grafie nienasyconych składowych $G_U(P)$ jest równa przynajmniej τu. Wówczas współczynnik aproksymacji algorytmu 3.7.5 jest ograniczony przez $\frac{4}{3}(3-2\tau)$.*

Dowód. Skoro istnieje upakowanie trójkątów angażujące co najmniej τu wierzchołków, to istnieje również rozwiązanie problemu pokrycia, złożone z tych trójkątów, i z co najwyżej $(1-\tau)u$ pojedynczych wierzchołków. Koszt tego rozwiązania to $z_{APX} \leqslant \frac{1}{3}\tau u + (1-\tau)u$. Na mocy twierdzenia 3.5.25, mamy ograniczenie $z_{OPT} \geqslant \frac{u}{3}$, a więc omawiane rozwiązanie dostarcza współczynnika aproksymacji $\frac{z_{APX}}{z_{OPT}} \leqslant \tau + 3(1-\tau)$. Oczywiście, koszt z_{3OPT} najtańszego pokrycia trójkami jest nie większy, tzn. $z_{3OPT} \leqslant z_{APX}$.

Algorytm Duh-a i Fürer-a generuje 3-pokrycie o koszcie $z_{DF} \leqslant \frac{4}{3} z_{3OPT} \leqslant \frac{4}{3} z_{APX}$, to znaczy

$$\frac{z_{DF}}{z_{OPT}} \leqslant \frac{4}{3}(\tau + 3(1-\tau)) = \frac{4}{3}(3 - 2\tau).$$

Podstawiając otrzymane ograniczenia do wzorów na przybliżoną i optymalną liczbę skoków, $s_A(P) \leqslant |P| - e + z_{DF}$ i $s_O(P) = |P| - e + z_{OPT}$, widzimy, że ten sam współczynnik aproksymacji jest zapewniony również dla minimalizacji liczby skoków. □

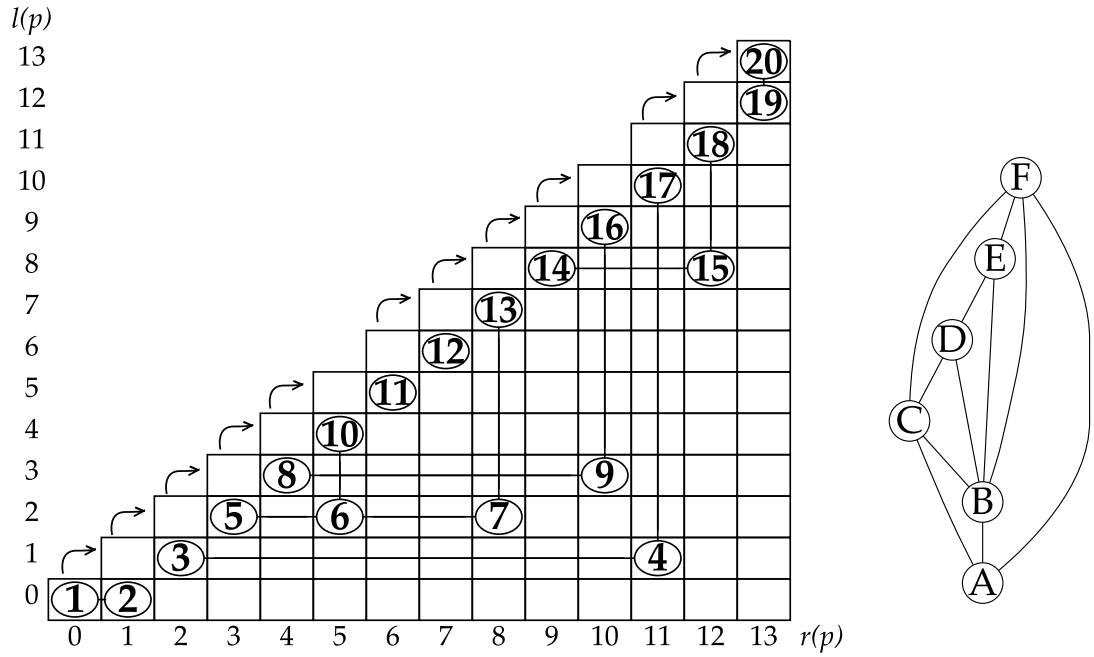
Teraz formułujemy główny wynik tej rozprawy.

Twierdzenie 3.7.7. *Istnieje algorytm działający w czasie wielomianowym, przybliżający liczbę skoków posetu przedziałowego ze współczynnikiem aproksymacji ograniczonym przez* 1.484*.*

Dowód. Na mocy wniosku 3.7.1 i lematu 3.7.6 mamy dwa ograniczenia na stosunek $\frac{s_A(P)}{s_O(P)}$. Jeśli A jest algorytmem zwracającym dłuższy ciąg progów z dwóch uzyskanych algorytmami 3.5.27 i 3.7.5, to największe odchylenie od optimum może być osiągane dla τ bliskiego $\frac{30-\sqrt{504}}{8}$. Oczywiście jest to punkt przecięcia obu ograniczeń, gwarantujący aproksymację ze współczynnikiem mniejszym niż $(2\sqrt{14} - 6)$. Ta wartość jest ograniczona przez $\frac{89}{60}$, a nawet przez 1.483315. □

Przykład 3.7.8. Graf przedziałów dla posetu na rysunku 3.7.9 zawiera 6 nienasyconych składowych: $A = \{3, 4, 17\}$, $B = \{5, 6, 7, 10, 13\}$, $C = \{8, 9, 16\}$, $D = \{11\}$, $E = \{12\}$, $F = \{14, 15, 18\}$. Aby wypisać wszystkie $P2$-trójkąty, sprawdzamy dla każdej komórki $[i, i+1]$, czy komórka $[i, i+1]$, kolumna i oraz wiersz $i+1$ zawierają wierzchołki trzech różnych nienasyconych składowych. W ten sposób, otrzymujemy następujące $P2$-trójkąty: $3, 5, 8 \to \{A, B, C\}$, $8, 10, 11 \to \{B, C, D\}$, $10, 11, 12 \to \{B, D, E\}$, $11, 12, 13 \to \{B, D, E\}$, $12, 13, 14 \to \{B, E, F\}$, $13, 14, 16 \to \{B, C, F\}$, $14, 16, 17 \to \{A, C, F\}$, $16, 17, 18 \to \{A, C, F\}$. Te trójkąty stanowią instancję problemu pokrycia trójkami, razem z krawędziami i wierzchołkami grafu nienasyconych składowych pokazanego na tym samym rysunku. Inne trójkąty (tj. $\{A, B, F\}$) nie odpowiadają własności $P2$. Ten graf ma zarówno skojarzenie doskonałe, jak i upakowanie doskonałe trójkątów. Algorytm Cornuéjolsa i in. może zwrócić skojarzenie doskonałe. Z perspektywy liczby skoków, korzystniej jest wygenerować upakowanie maksymalizujące nie tylko liczbę wierzchołków, ale i cykli. Zatem upakowanie (i pokrycie) z_{APX}

złożone z $\{B, D, E\}$ $(10, 11, 12)$ oraz $\{A, C, F\}$ $(16, 17, 18)$ jest preferowane nad skojarzeniem, i jest generowane przez algorytm 3.7.5. Odpowiadający realizowalny ciąg progów jest narysowany jako ciąg strzałek nad tabelą ((col_5, row_6) i (col_{10}, row_{11}) są omijane). Stosując proces etykietowania (algorytm 3.5.16) odczytujemy optymalne rozszerzenie liniowe $L = (1, 4) \oplus (2, 6) \oplus (3, 9) \oplus (5, 10) \oplus (8, 11, 13) \oplus (12, 15) \oplus (7, 16) \oplus (14, 17, 19) \oplus (18, 20)$.



Rysunek 3.7.9. Poprawa aproksymacji

Uzyskaliśmy specjalny przypadek problemu pokrycia zbioru. Być może jego struktura umożliwi jeszcze ściślejszą aproksymację? Istotność tego pytania jest wzmocniona przez fakt, że w praktyce otrzymane przybliżone liczby skoków okazują się być bliższe optymalności, niż gwarantuje otrzymany dowód. Jednakże, wartość $|P| - e$ jest ważnym czynnikiem w obliczeniu odchylenia, przyczyniającym się do zmniejszenia $\frac{s_A}{s_O}$. Zauważmy, że $|P| - e$ jest równe zeru tylko dla łańcuchów.

Wzorzec przykładu 3.7.8 można łatwo powtórzyć w większych posetach. W lemacie 3.7.6 byłoby interesującym pozbyć się czynnika $\frac{4}{3}$ przez zwrócenie z_{APX}, które istnieje z założenia. Jednakże, problem pakowania trójkątów jest w ogólności NP-trudny [36]. Konsekwentnie, użyty został algorytm aproksymacyjny dla 3-pokrycia zbioru. Współczynnik aproksymacji dla liczby skoków oczywiście obniża się, jeśli istnieje algorytm dla

3-pokrycia zbioru o współczynniku ściślejszym niż $\frac{4}{3}$.

3.8. Algorytm genetyczny dla posetów przedziałowych

Opisujemy teraz, jak fakty ustanowione przez Mitas można przełożyć na prosty algorytm genetyczny dla problemu liczby skoków na posetach przedziałowych. To znaczy, opisujemy metodę optymalizacji ciągu progów tak, by ostatecznie każda nienasycona składowa miała własność $P1$ albo $P2$. Innymi słowy, podajemy tutaj alternatywny sposób realizacji kroku **3** algorytmu 3.5.18.

Chcemy z pomocą optymalizacji genetycznej wyznaczać własność $P2$ dla pewnych nienasyconych składowych. Jeśli element posetu $[j, j+q]$ ma własność $P2$ względem ciągu progów T, to T omija pewną liczbę par postaci (col_i, row_{i+1}) otaczających ten element. Zdefiniujmy wektor binarny B długości $e-1$ następująco: $B[i] = 1$ dokładnie wtedy, gdy (col_i, row_{i+1}) jest omijana przez ciąg progów, $i = 0, \ldots, e-2$. Operatory genetyczne krzyżowania i mutacji mają działać na takich kandydackich wektorach, zainicjalizowanych losowo. Jakość kandydata B jest oceniana w sposób promujący niską liczbę omijanych wierszy i kolumn w odpowiadającym ciągu progów $T(B)$, zachowując warunek, że $T(B)$ ma być realizowalny. Jest to wyrażone algorytmem 3.8.1.

Zatem, celem proponowanego hybrydowego algorytmu genetycznego jest wykorzystanie własności $P2$ wszędzie, gdzie to możliwe. Dla każdego kandydackiego rozwiązania, gdy pewne nienasycone składowe spełniają żądane własności, uruchamiamy skojarzeniowy wariant algorytmu Mitas na pozostałej części grafu. Zauważmy, że ta adaptacja zawsze zwraca dopuszczalne rozszerzenie liniowe. Co więcej, algorytm dziedziczy zalety metody Mitas.

Algorytm 3.8.1. Optymalizacja genetyczna ciągu progów – funkcja dopasowania.

---

1. Sprawdź, które nienasycone składowe mają własność $P2$, gdy pary (col_i, row_{i+1}) są omijane, dla tych i, że $B[i] = 1$.

2. Sprawdź dodatkowo, które nienasycone składowe mają własność $P1$ jako efekt tej decyzji.

3. Zbuduj graf składowych, złożony tylko z tych nienasyconych składowych, które nadal nie mają własności ani $P1$, ani $P2$, i wyznacz najliczniejsze skojarzenie w tym grafie. W ten sposób można odrzucić dodatkowe wiersze i kolumny.

4. Policz każdą z pozostałych składowych C (spośród tych, które nie mają ani własności $P1$, ani $P2$ po poprzednim kroku) jako jeden stracony próg.
5. Zwróć łączną liczbę progów w $T(B)$ (łączna liczba omijanych progów jest sumą liczby jedynek w B, liczby skojarzonych krawędzi, i liczby wierzchołków nie uwzględnionych w wygenerowanym skojarzeniu; tę wartość odejmujemy od $e-1$).

Przykład 3.8.2. Rozważmy poset na rysunku 3.8.3. Jeśli $B[7]=1$ i B zawiera zera na wszystkich pozostałych miejscach, to para (col_7, row_8) ma być omijana przez ciąg progów, więc składowa $\{7\}$ ma własność $P1$. W grafie pozostałych składowych, $\{4\}$ oraz $\{6\}$ są połączone krawędzią odpowiadającą (col_4, row_4), a trzy składowe $\{9\}, \{11\}, \{12\}$ stanowią trójkąt. Jeśli najliczniejsze skojarzenie składa się z $\{4,6\}$ i np. $\{9,11\}$, to (col_4, row_4) i (col_9, row_9) będą omijane przez ciąg progów. Dla pozostałej składowej $\{12\}$ dodatkowa para jest omijana, (col_{12}, row_{12}). Tracimy 4 progi, każda nienasycona składowa ma własność $P1$ i jest $e-1-4=9$ progów w ciągu progów

$$T_1 = \{(0,1),(1,2),(2,3),(3,5),(5,6),(6,7),(8,10),(10,11),(11,13)\}.$$

Powstałe rozszerzenie liniowe zawierające 5 skoków to

$$(1,2,4) \oplus (3,5,7) \oplus (6,8,9) \oplus (10,12) \oplus (11,13,15) \oplus (14).$$

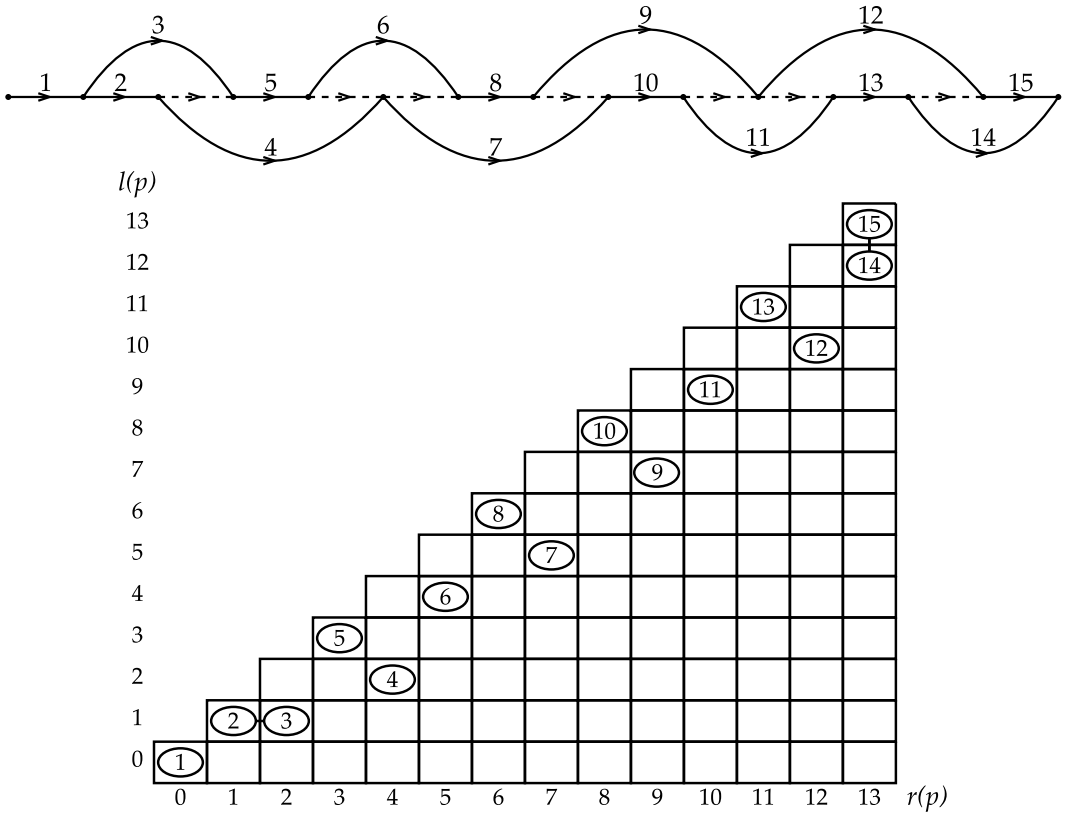
Niech teraz $B[4]=1, B[0]=1$ i $B=0$ na pozostałych pozycjach. Zauważmy, że wszystkie nienasycone składowe mają własność $P1$ albo $P2$, gdy (col_4, row_5) i (col_9, row_{10}) są omijane przez ciąg progów. Dokładniej, $\{6\}$ i $\{11\}$ spełniają $P2$, podczas gdy $\{4\}$, $\{7\}$, $\{9\}$ i $\{12\}$ spełniają $P1$. Odpowiadającym ciągiem progów jest

$$T_2 = \{(0,1),(1,2),(2,3),(3,4),(5,6),(6,7),(7,8),(8,9),(10,11),(11,12),(12,13)\},$$

który przekłada się na rozszerzenie liniowe

$$(1,2,4) + (3,5,6,8,9) + (7,10,11,13,14) + (12,15).$$

z 11 progami i 3 skokami.



Rysunek 3.8.3. Jeden z trudnych przykładów dla algorytmu Mitas

Wykonaliśmy doświadczenie, by dowiedzieć się, ile iteracji potrzebuje ten hybrydowy algorytm genetyczny do wygenerowania optymalnego rozwiązania. Najpierw dokładny algorytm półsilnie zachłanny (algorytm 2.3.13) był użyty na losowo wybranym trudnym posecie do wyznaczenia $s(P)$. Następnie proponowane genetyczne rozszerzenie procedury Mitas było testowane na tym samym posecie bez limitu czasu, tzn. jedynym kryterium stopu było osiągnięcie dokładnej wartości $s(P)$ podczas procesu optymalizacji. W tabeli 3.8.4 porównano liczbę wywołań funkcji celu w algorytmie genetycznym z liczbą wszystkich półsilnie zachłannych rozszerzeń liniowych na ośmiu trudnych posetach rozmiaru 100. Posety użyte w tych testach składały się z mieszanki trudnych instancji dla trzech algorytmów aproksymacyjnych. Obserwujemy, że w większości przypadków algorytm dokładny Sysły (`OptLinExt`) wymaga mniej iteracji niż hybrydowy algorytm genetyczny. Jednakże sporadycznie liczba półsilnie zachłannych rozszerzeń liniowych może być bardzo duża (co widać też na rysunku 2.3.15), podczas gdy czas optymalizacji genetycznej jest podobny dla każdej instancji. To samo zachowanie było

zaobserwowane, gdy przeprowadzano więcej eksperymentów tego rodzaju, i rozmiar posetu zmieniał się od 60 do 100.

Tabela 3.8.4. Porównanie liczby iteracji, w których znajdowane jest optymalne rozszerzenie liniowe przez proponowane genetyczne rozszerzenie algorytmu Mitas, z liczbą wszystkich półsilnie zachłannych rozszerzeni liniowych, w próbce 8 losowo wybranych trudnych instancji P_1 – P_8

algorytm	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
`OptLinExt`	2016	672	544	136	96	660	15872	1008
genetyczny Mitas	3546	1066	1484	1000	1180	1396	1210	1084

3.9. Algorytm dokładny dla posetów przedziałowych

Następujący rezultat jest wnioskiem z pracy Mitas dotyczącej aproksymacji liczby skoków.

Wniosek 3.9.1. *Liczbę skoków posetu przedziałowego P można obliczyć w czasie* $\mathcal{O}(2^n \cdot \mathrm{poly}(n))$.

Dowód. Jak wykazała Mitas, dla optymalnego ciągu progów istnieje optymalne upakowanie względem $\frac{v+c}{2}$. Można zauważyć, że liczba $P2$-cykli jest ograniczona przez $e \leqslant n$. Zatem wystarczy przeliczyć wszystkie upakowania wierzchołkowo rozłącznych $P2$-cykli, i uzupełnić każde takie upakowanie H o najliczniejsze skojarzenie na grafie $G_U(P) - H$. Ze wszystkich upakowań wybieramy upakowanie maksymalizujące $\frac{v+c}{2}$. □

Wykazano w pierwszej części pracy, że taką samą złożoność zapewnia redukcja do problemu komiwojażera. Niemniej, doświadczenia obliczeniowe pokazują, że warto stosować algorytm opisany w dowodzie wniosku 3.9.1. W praktyce, ze względu na typową strukturę powstających grafów, dzięki zaangażowaniu prostych heurystyk ten algorytm ma satysfakcjonującą szybkość działania. Liczba $P2$-cykli w $G_U(P)$ jest często ograniczona. Co więcej, te cykle zwykle się nachodzą, więc algorytm przeszukiwania wyczerpującego może unikać wielu podzbiorów cykli, które mają przynajmniej jeden wspólny wierzchołek. Zatem liczbę skoków można policzyć w rozsądnym czasie nawet na posetach przedziałowych P mających kilkaset elementów (tj. poniżej 10 sekund).

Bardziej formalnie, proponujemy szybki algorytm dokładny oparty na idei rozgałęziania, sformułowany jako algorytm 3.9.2.

Algorytm 3.9.2. Algorytm dokładny dla liczby skoków na posetach przedziałowych.

Wejście: Poset przedziałowy $(P, <_P)$

Wyjście: Optymalne rozszerzenie liniowe posetu P.

1. Wyznacz kanoniczną reprezentację tabelową posetu P.

2. Wygeneruj graf nienasyconych składowych i wszystkie $P2$-cykle.

3. Przelicz wszystkie upakowania wierzchołkowo rozłącznych $P2$-cykli następująco:

(a) Wybierz jeden z dostępnych $P2$-cykli C.

(b) Cykl C jest albo w upakowaniu, albo poza upakowaniem. Dla każdego z tych dwóch przypadków konstruuj dalej upakowanie, powtarzając krok **3**, przy czym C przestaje być dostępny. Upakowanie jest gotowe, gdy nie ma więcej dostępnych cykli, dla których nie podjęto decyzji, czy są w upakowaniu, czy poza upakowaniem.

4. Dla każdego wygenerowanego upakowania cykli H, uzupełnij upakowanie wyznaczając najliczniejsze skojarzenie na pozostałym grafie $G_U(P) - H$.

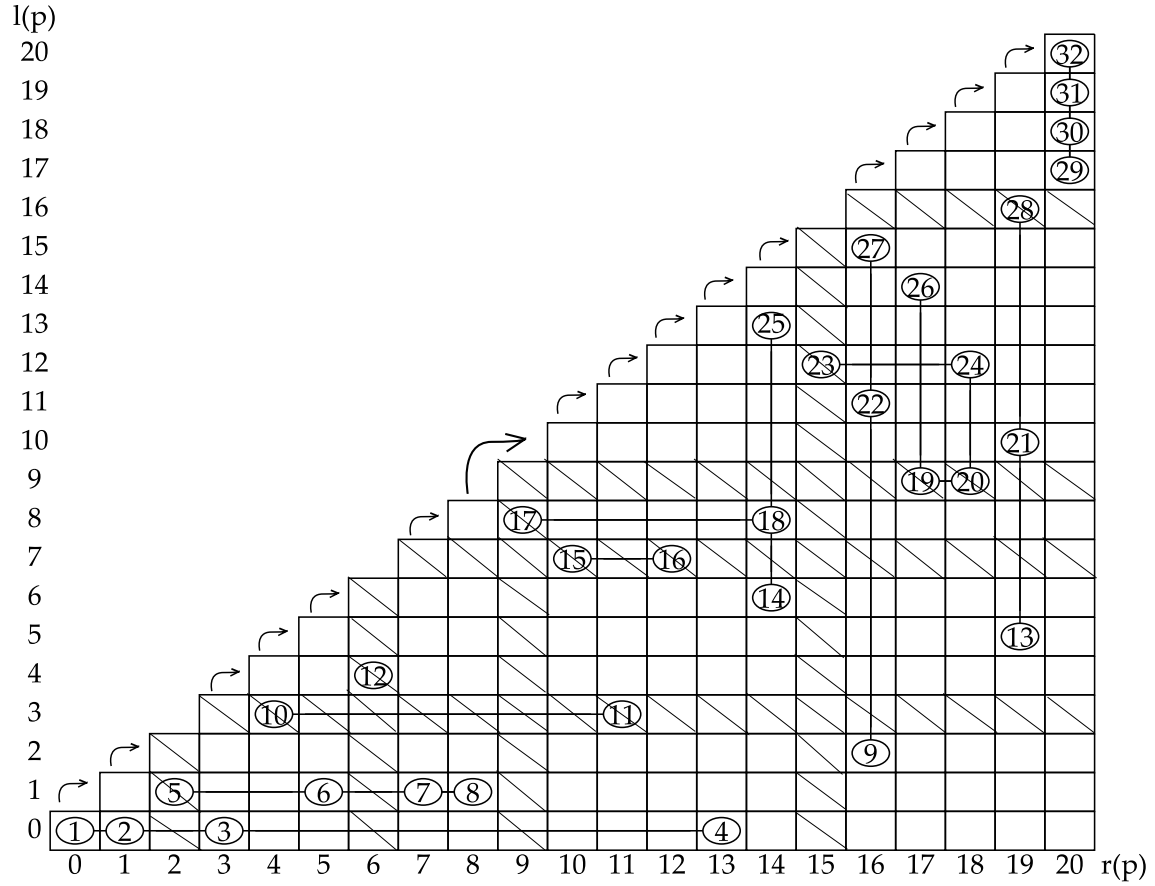
5. Wybierz upakowanie maksymalizujące $\frac{v+c}{2}$, a następnie na podstawie tego upakowania zbuduj optymalny ciąg progów i odczytaj odpowiadające rozszerzenie liniowe.

Twierdzenie 3.9.3. *Algorytm* 3.9.2 *wyznacza liczbę skoków posetu przedziałowego w czasie* $\mathcal{O}(1.47^n \cdot \text{poly}(n))$.

Dowód. Ponieważ optymalny ciąg progów jest stowarzyszony z optymalnym upakowaniem, a przeglądamy wszystkie upakowania rozłącznych cykli i każde uzupełniamy największym możliwym skojarzeniem, jest jasne, że algorytm działa poprawnie. Przejdźmy do złożoności czasowej.

Poset n-elementowy ma co najwyżej $e \leqslant n$ $P2$-cykli, które można uwzględnić w upakowaniu. Algorytm 3.9.2 dla każdego $P2$-cyklu rozgałęzia problem na dwie mniejsze instancje. Jeśli cykl C jest brany do upakowania, to przynajmniej 3 elementy posetu wchodzą do upakowania, a więc nie można ich rozważać w ramach innych cykli ani krawędzi. Zatem mniejsza instancja ma co najwyżej $n - 3$ elementy posetu i, co za tym idzie, co najwyżej $n-3$ $P2$-cykle. Jeśli C nie jest brany do upakowania, to pomniejszona instancja ma co najwyżej $n - 1$ $P2$-cykli. Przeliczanie upakowań rozłącznych cykli ma więc złożoność czasową zależną od liczby wierzchołków w drzewie poszukiwań, którą można opisać rekurencją $T(n) \leqslant T(n-3) + T(n-1)$. To oznacza, że algorytm działa w czasie $\mathcal{O}(1.47^n \cdot \text{poly}(n))$, gdyż jedyny rzeczywisty pierwiastek równania charaktery-

stycznego powyższej rekurencji, $x^3 - x^2 - 1 \geqslant 0$, jest ograniczony z góry przez 1.47, a nawet przez 1.4656. □



Przykład 3.9.4. Poset przedziałowy na rysunku 3.9 składa się z 32 elementów, z grafem przedziałów tworzącym 10 składowych, z których 8 jest nienasyconych. Jego graf składowych (definicja 3.5.20) ma 8 wierzchołków, i trzy $P2$-cykle: jeden pięciokąt, i dwa trójkąty. Widać istotne ograniczenie przestrzeni poszukiwań – pomimo, że poset ma 32 elementy, rozpatrujemy co najwyżej 8 podzbiorów $P2$-cykli, spośród których interesują nas tylko te, które są wierzchołkowo rozłączne. Każde z nich w czasie wielomianowym uzupełniamy najliczniejszym skojarzeniem na pozostałym podgrafie. Optymalne upakowanie jest złożone z trójkąta $\{\{9, 22, 27\}, \{13, 21, 28\}, \{19, 20, 23, 24, 26\}\}$ i krawędzi $\{\{12\}, \{15, 16\}\}$, $\{\{10, 11\}, \{5, 6, 7, 8\}\}$. Odpowiadający ciąg progów jest narysowany w postaci strzałek nad tabelą, oznaczających kolejne progi optymalnego rozszerzenia liniowego.

3.10. Baza trudnych posetów przedziałowych

Badania prowadzone w ramach pracy doktorskiej przez cały czas miały dwojaki charakter. Obok rozważań teoretycznych, osnutych wokół kombinatorycznych własności posetów przedziałowych, przeprowadzane były testy znanych wcześniej oraz nowo proponowanych algorytmów. Oczywiście w pierwszej kolejności to algorytmy aproksymacyjne Felsnera, Sysły i Mitas zostały intensywnie przetestowane i porównane. Zarazem uzyskano obszerny zbiór trudnych instancji, na których osiągane jest najgorsze możliwe odchylenie od optimum w pesymistycznych przebiegach każdego z algorytmów. W tym podrozdziale omawiamy bazę tych posetów. Rezultaty te są bezcenne dla badaczy zainteresowanych rozwijaniem nowych algorytmów, gdyż duża ilość nietrywialnych posetów jest niezwykle pomocna w weryfikowaniu pomysłów algorytmicznych. Co więcej, przeprowadzając te doświadczenia uzyskano głęboki wgląd w problem skoków. Baza posetów przedziałowych została opisana w artykule [46], a zebrane posety są dostępne na stronie internetowej autora [47].

Naszym celem było wyznaczenie trudnych posetów przedziałowych, zmuszających algorytmy Felsnera, Sysły i Mitas do generowania uporządkowań, które są o 50% gorsze od optymalnych rozszerzeń liniowych. Do realizacji tego celu wykorzystano algorytm genetyczny. Zawsze, gdy trzeba znaleźć trudny poset n-elementowy, chromosom składa się z n przedziałów. Operator krzyżowania miesza dwa chromosomy i produkuje dwa nowe chromosomy. Ta procedura jest realizowana następująco: losujemy liczbę r w zakresie od 1 do n; pierwszych r przedziałów pierwszego rodzica R_1 kopiujemy do nowo utworzonego potomka C_1, a resztę przedziałów rodzica R_1 kopiujemy do nowo utworzonego potomka C_2. Następnie przedziały o numerach od $(r+1)$ do n w R_2 są kopiowane do C_1 i analogicznie brakujące przedziały od $(r+1)$ do n w R_1 są kopiowane do C_2. W procedurze mutacji losowo wybrany przedział jest usuwany, a inny (losowy) jest dodawany.

Jest naturalnym zdefiniować funkcję celu w taki sposób, by porównywała rozwiązanie otrzymane algorytmem aproksymacyjnym z optymalną liczbą skoków i obliczała współczynnik aproksymacji na posecie reprezentowanym chromosomem. Oczywiście, trzeba ją wykonać po każdym zastosowaniu mutacji lub krzyżowania. Pełny przegląd rozwiazań ograniczonych do wszystkich półsilnie zachłannych rozszerzen liniowych może być wykonany względnie szybko. De facto mamy trzy podobne funkcje celu, z których

każda poddaje próbie inny algorytm. W praktyce, podczas badania algorytmu Sysły lub Felsnera, otrzymujemy 50%-nieoptymalne posety ustalonego rozmiaru 30 w czasie nie przekraczającym pół godziny na komputerze z procesorem Intel® Core™ i5-2500K CPU taktowanym zegarem 3.30 GHz, wyposażonym w 24 GB pamięci RAM. W przypadku algorytmu Mitas generujemy tylko jedno przybliżone upakowanie i porównujemy je z optimum.

Trudne posety przedziałowe znalezione przez nasz algorytm genetyczny stanowią pomoc w projektowaniu nowych algorytmów. Po stronie wizualnej, dostępnej przez stronę internetową [47], zbiór posetów można obejrzeć i pobrać. Użytkownik widzi przedziały stanowiące poset w postaci kanonicznego diagramu łukowego oraz w postaci tabeli. Po stronie programistycznej, można zażądać posetu podanego rozmiaru, który jest trudny dla znanych do tej pory algorytmów. Podany rozmiar może przekraczać rozmiar największego posetu z bazy. W takim przypadku, istniejące posety przedziałowe sa składane (przez złożenie szeregowe) i duża instancja jest zwracana użytkownikowi. Złożenie szeregowe kanonicznego posetu przedziałowego realizujemy kładąc przedziały posetu Q tuż po skrajnym prawym przedziale posetu P. Baza trudnych posetów została opublikowana na stronie internetowej autora [47], więc każdy zainteresowany projektowaniem nowych algorytmów ma dostęp do najtrudniejszych posetów, problematycznych dla obecnych algorytmów.

Przykłady trudnych posetów w tej rozprawie widoczne są na rysunkach: 3.2.7 (dla algorytmu Felsnera), 3.3.7 (dla algorytmu Sysły), 3.7.9 i 3.8.3 (dla algorytmu Mitas). Trudne instancje zostały wykorzystane w doświadczalnym badaniu algorytmów: przeszukiwania z zakazami (algorytm 2.4.2, tabele 3.3.10, 3.3.11 i 3.3.12), hybrydowego algorytmu genetycznego opartego o twierdzenia Mitas (algorytm 3.8.1, tabela 3.8.4), a także przy porównywaniu algorytmów aproksymacyjnych Felsnera, Sysły i Mitas (tabele 3.6.2 i 3.6.3 w podrozdziale 3.6). Otrzymano dobry wgląd w kombinatoryczną strukturę problemu, której efektem jest poprawiony współczynnik aproksymacji omówiony w podrozdziale 3.7.

4. Problem skoków na posetach dwuwymiarowych

Klasa posetów dwuwymiarowych od lat przyciąga badaczy. Wiele problemów posetowych, po ograniczeniu do posetów dwuwymiarowych, okazało się rozwiązywalnych wielomianowo. Do dziś nie wiadomo jednak, czy problem skoków na posetach dwuwymiarowych (w skrócie: posetach 2D) ma algorytm wielomianowy, czy pozostaje NP-trudny. W tym rozdziale podajemy pewne wyniki dotyczące tej klasy posetów. Po pierwsze, zauważamy, że w oparciu o twierdzenie udowodnione w części 3.7 rozprawy (lemat 3.7.2), otrzymujemy natychmiast poprawiony współczynnik aproksymacji na posetach przedziałowych, które są zarazem dwuwymiarowe. Następnie podajemy wyniki obliczeniowe algorytmu przeszukiwania z zakazami, który zaproponowano w podrozdziale 2.4, gdy jest on stosowany do posetów dwuwymiarowych.

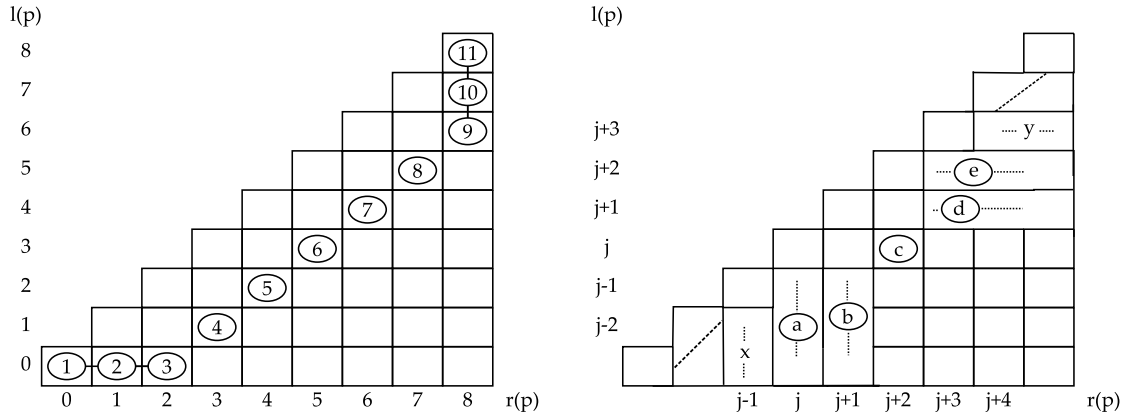
4.1. Posety przedziałowe wymiaru 2

W tym podrozdziale dowodzimy, że na posetach przedziałowych wymiaru 2 liczbę skoków można aproksymować w czasie wielomianowym ze współczynnikiem 4/3. Jest to konsekwencja lematu 3.7.2 z podrozdziału 3.7, gdzie problem skoków posetu przedziałowego został zredukowany do pokrycia zbioru. Pokazujemy w tym celu, że graf nienasyconych składowych w przypadku takich posetów nie zawiera $P2$-cykli rozmiaru $(2k+1)$ dla $k \geqslant 2$. W takich grafach występują jedynie trójkąty.

Lemat 4.1.1. *Niech P będzie posetem przedziałowym wymiaru 2. Wówczas graf nienasyconych składowych $G_U(P)$ nie zawiera $P2$-cykli rozmiaru $(2k+1)$ dla żadnego $k \geqslant 2$.*

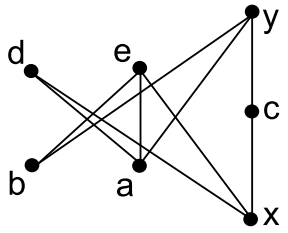
Dowód. Pokazujemy, że $G_U(P)$ nie zawiera $P2$-pięciokątów. Dowód prowadzimy niewprost. Załóżmy, że w tabeli da się ułożyć pięć elementów posetu w taki sposób, że każdy z nich należy do innej nienasyconej składowej, przy czym jeden element jest pośrodku tego układu w komórce $[j, j+2]$ (c), dwa leżą odpowiednio w kolumnach j i $j+1$ (a oraz b), a dwa leżą odpowiednio w wierszach $j+1$, $j+2$ (d oraz e). Na rysunku 4.1.2 przedstawiona jest najmniejsza taka sytuacja ($a=4$, $b=5$, $c=6$, $d=7$, $e=8$), a także sytuacja ogólna. Oprócz tych przynajmniej pięciu elementów należących

do różnych nienasyconych składowych, w tabeli muszą być umieszczone elementy uzupełniające, żeby nie zostawić ani jednej pustej kolumny i ani jednego pustego wiersza. Zatem w kolumnie $j-1$ istnieje element x, który jest poprzednikiem elementów c, d, e oraz jest nieporównywalny z a, b. Podobnie, w wierszu $j+3$ istnieje element y, który jest następnikiem elementów a, b, c oraz jest nieporównywalny z d, e.



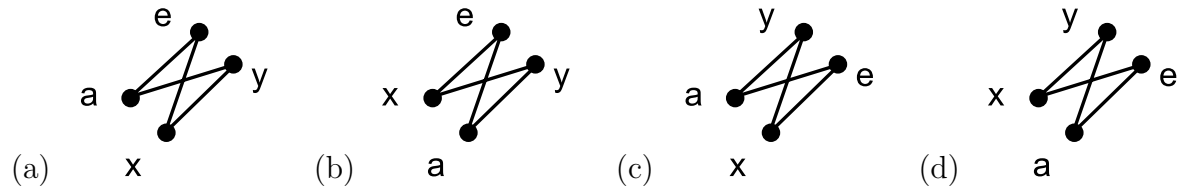
Rysunek 4.1.2. Najmniejszy poset przedziałowy zawierający $P2$-pięciokąt w grafie nienasyconych składowych, oraz sytuacja ogólna

Spróbujmy znaleźć dwuwymiarowy realizator podposetu złożonego z elementów a, b, c, d, e, x, y, którego diagram Hassego widoczny jest na rysunku 4.1.3.



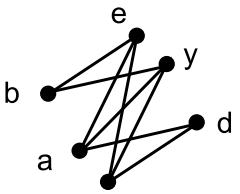
Rysunek 4.1.3. Podposet posetu z rysunku 4.1.2, który nie jest dwuwymiarowy

Podposet $\{a, x, e, y\}$ można ułożyć na cztery sposoby:

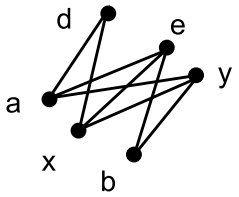


W przypadkach (b) i (c) od razu widać, że nie da się dostawić elementu c, gdyż w każdym miejscu pomiędzy x i y będzie on większy od a lub mniejszy od e.

W przypadku (a) mamy dwie możliwości dostawienia elementów b, d:

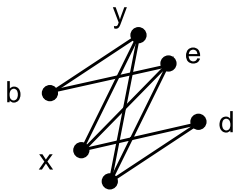


(a1)

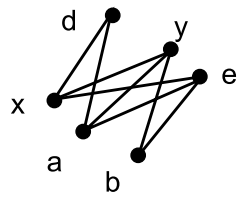


(a2)

Podobnie, w przypadku (d) mamy dwie możliwości:



(d1)



(d2)

W przypadkach (a1) i (d2), dostawienie elementu c pomiędzy x i y sprawi, że będzie on większy od a lub mniejszy od d. W przypadkach (a2) i (d1), dostawienie elementu c pomiędzy x i y sprawi, że będzie on większy od b lub mniejszy od e.

Widać, że jeśli rozważymy jakikolwiek dwuwymiarowy realizator tego posetu bez uwzględnienia „centralnego" elementu c, to nie da się dostawić elementu c w taki sposób, by spełniał wymagane porównywalności, i zarazem nie spełniał żadnych innych. Zatem ze względu na monotoniczność wymiaru [73, Proposition 4.6], poset przedziałowy zawierający $P2$-pięciokąt nie może być dwuwymiarowy.

Przypadek większych $(2k+1)$-cykli jest analogiczny. □

Wniosek 4.1.4. *Algorytm* 3.7.5 *jest algorytmem* $\frac{4}{3}$*-aproksymacyjnym dla problemu skoków na posetach przedziałowych wymiaru* 2*.*

Dowód. Optymalny ciąg progów jest stowarzyszony z optymalnym upakowaniem, w którym oszczędność jest równa w_O. Liczba skoków jest równa $s_O = |P| - e + u - w_O$. To optymalne upakowanie jest zarazem pokryciem, złożonym z tych samych cykli i krawędzi, co upakowanie, i do tego z wierzchołków nieobjętych upakowaniem. Koszt tego pokrycia jest równy $z_O = u - w_O$. Stosując algorytm 3.7.5 otrzymamy rozszerzenie liniowe o przybliżonej liczbie skoków $s_A = |P| - e + z_A$, a skoro $z_A \leqslant \frac{4}{3} z_O$ (i $|P| - e \geqslant 0$), ten sam współczynnik aproksymacji jest zapewniony dla liczby skoków. □

4.2. Dolne ograniczenie dla posetów dwuwymiarowych

Jak zaobserwował Ceroi [9], dla posetów dwuwymiarowych liczbę skoków można interpretować jako problem znalezienia zbioru niezależnego o największej wadze spośród osiowo równoległych prostokątów odpowiadających pewnym łańcuchom w P. Niech P będzie posetem dwuwymiarowym z realizatorem $\{L_1, L_2\}$. Z każdym $p \in P$ stowarzyszamy punkt $(x, y) \in \mathbb{R}^2$ taki, że x jest pozycją p w L_1 i y jest pozycją p w L_2. Wtedy każdy łańcuch P jest widziany jako prostokąt w $\mathbb{R}^2$. Rozszerzenia liniowe są złożone tylko z łańcuchów wypukłych, tzn. spełniających: $\forall p <_P q <_P r \in P$, jeśli $p \in C$ oraz $r \in C$, to $q \in C$. Jeśli przez $R(P) = (V_R, E_R)$ oznaczymy graf przecięć prostokątów, w którym waga każdego wierzchołka $v \leftrightarrow C_v$ jest równa jego progom, tj. $w(v) = |C_v| - 1$, to mamy następujący wynik.

Lemat 4.2.1 (Ceroi [9])**.** *Największa liczba progów $b(P)$ jest równa największej wadze zbioru niezależnego w $R(P)$.*

Wobec tego, w celu policzenia $b(P)$, wystarczy rozwiązać całkowity program liniowy dla zbioru niezależnego o największej wadze (MWIS), $\max \sum_v w(v) \cdot x(v)$ zgodnie z $x(v) \in \{0, 1\}$ dla każdego $v \in V_R$, i $x(u) + x(v) \leqslant 1$ dla każdej $(u, v) \in E_R$. Niemniej, $|V_R|$ zwykle jest większe niż $|P|$ i dokładne obliczenie MWIS szybko staje się czasochłonne. Zatem dostajemy tylko górne ograniczenie $b_{UB}(P) \geqslant b(P)$, rozwiązując LP-relaksację tego sformułowania, to jest przyjmując $0 \leqslant x(v) \leqslant 1$ (a więc dolne ograniczenie $s_{LB}(P) = \lceil n - 1 - b_{UB}(P) \rceil$ na liczbę skoków).

4.3. Doświadczenia na posetach dwuwymiarowych

Algorytm przeszukiwania z zakazami opisany w podrozdziale 2.4 został przetestowany. Wyniki tych eksperymentów opisano w [48].

W pierwszym doświadczeniu dotyczącym posetów dwuwymiarowych n było ustawione na 30. Dla 30-elementowych posetów optymalne rozwiązanie można wyznaczyć za pomocą algorytmu `OptLinExt`. Na większych instancjach stosujemy LP-relaksację, która dostarcza górnego ograniczenia na $b(P)$.

Wylosowano dziesiątki losowych 30-elementowych posetów dwuwymiarowych o zróżnicowanej liczbie łuków pozornych (od 10 do 50). Najpierw obliczana była liczba skoków

$s(P)$. Następnie uruchamiano procedurę szukania z zakazami (tj. 2.4.2) z ograniczeniem iteracji równym n. Optimum było znajdowane przez algorytm 2.4.2 we wszystkich przypadkach poza jednym posetem o 50 łukach pozornych, dla którego wygenerowane rozszerzenie miało 12 skoków zamiast 11. Na wszystkich instancjach, algorytm wymagał co najwyżej 21 iteracji i nie więcej niż 2 sekund.

Przy tej okazji $b(P)$ było dodatkowo porównywane z relaksacją programowania liniowego $b_{UB}(P)$ równoważnego problemu MWIS. Pewne typowe odchylenia tych dwóch wartości są pokazane w tabeli 4.3.2. Liczba wierzchołków w $R(P)$, tj. liczba wypukłych łańcuchów w P, zmieniała się pomiędzy 96 a 138.

W następnym doświadczeniu, wygenerowano 150 posetów z $n = 60$. Tym razem, przybliżona liczba skoków była porównywana tylko z dolnym ograniczeniem uzyskanym za pomocą LP-relaksacji MWIS na $R(P)$, tj. $s_{LB}(P) = \lceil n - 1 - b_{UB}(P) \rceil$. Wyniki dla próbki 6 posetów są odnotowane w tabeli 4.3.3.

Średni błąd $s_{APX}(P)$, porównanej z $s_{LB}(P)$, wynosił 0.12. W najgorszym napotkanym przypadku, wyniósł 0.29. Liczba łuków pozornych w tych posetach przebiegała od 290 do 403. Czas działania był zawsze poniżej 12 sekund.

Ostatecznie, wzięto 30 dwuwymiarowych posetów o $n = 90$ elementach. Błąd s_{APX} uzyskanej w n iteracjach algorytmu wynosił średnio 0.15 i w najgorszym razie 0.29. Czas optymalizacji był zawsze poniżej 45 sekund (znacznie dłużej trwało obliczanie dolnego ograniczenia $s_{LB}(P)$ za pomocą programowania liniowego). 6 reprezentatywnych przypadków odnotowano w tabeli 4.3.4.

Obserwacja 4.3.1. *Algorytm* 2.4.2 *przeszukiwania z zakazami, zastosowany do n-elementowych posetów dwuwymiarowych, generuje rozszerzenia liniowe o liczbie skoków nie przekraczającej* 130% *optimum, w n iteracjach.*

Parametry przeszukiwania tabu we wszystkich testach były ustawione następująco: $TabuSize = 10, CheckedNeighbours = 7, MaxDummies = 15$. Te wartości otrzymano eksperymentalnie jako kompromis pomiędzy czasem działania i zbieżnością algorytmu.

Czas działania algorytmu 2.4.2 mógłby być poprawiony przez użycie szybszych metod przebudowy diagramu łukowego w każdym kroku. Na przykład, w przypadku posetów przedziałowych diagram łukowy można wygenerować algorytmem 3.3.1. Jest to

znacznie szybsze od ogólnego algorytmu 2.3.9. Wypróbowaliśmy tę konstrukcję i okazało się, że czasy działania podane w podrozdziale 3.3 zmniejszają się trzykrotnie.

Tabela 4.3.2. Rozbieżność pomiędzy liczbą progów i jej LP relaksacją, $n = 30$

$b(P)$	16	17	18	17	16
$b_{UB}(P)$	17.25	18.66	19.0	17.5	17.5
$b(P)$	17	19	18	16	17
$b_{UB}(P)$	18.33	19.75	19.88	17.33	18.11

Tabela 4.3.3. Wydajność szukania z zakazami na 60-elementowych posetach 2D

	ł. poz.	$\|V_R\|$	$b_{UB}(P)$	$s_{LB}(P)$	$s_{APX}(P)$	błąd
P_1	84	341	41.5	18	$23 \ldots 19$	0.0556
P_2	81	339	39.5	20	$25 \ldots 22$	0.1000
P_3	74	375	38.5	21	$24 \ldots 22$	0.0476
P_4	66	349	41.5	18	$23 \ldots 20$	0.1111
P_5	100	337	40.75	19	$22 \ldots 19$	0.0000
P_6	90	297	39.125	20	$26 \ldots 23$	0.1500

Tabela 4.3.4. Wydajność szukania z zakazami na 90-elementowych posetach 2D

	ł. poz.	$\|V_R\|$	$b_{UB}(P)$	$s_{LB}(P)$	$s_{APX}(P)$	błąd
P_1	166	593	63.57	26	$35 \ldots 29$	0.1154
P_2	150	566	62.44	27	$33 \ldots 30$	0.1111
P_3	158	570	60.96	29	$38 \ldots 31$	0.0690
P_4	167	599	61.58	28	$39 \ldots 36$	0.2857
P_5	163	631	63.00	26	$33 \ldots 29$	0.1154
P_6	173	557	60.17	29	$39 \ldots 34$	0.1724

Problemy i konkluzje

Tematem rozprawy było algorytmiczne wyznaczanie liczby skoków posetu. Przeważającą część pracy poświęcono posetom przedziałowym. Przedstawiono też pewne wyniki dla posetów dwuwymiarowych, a także dla ogólnego przypadku dowolnych posetów. Poniżej zawarto podsumowanie dysertacji.

W pierwszym rozdziale przytoczono pojęcia zarówno ze złożoności obliczeniowej, jak i z teorii grafów, używane w dalszej części pracy.

W drugim rozdziale sformułowano problem minimalizacji liczby skoków posetu, po czym zarysowano aktualny stan wiedzy o problemie oraz omówiono algorytm Sysły, czyli jeden z pierwszych ogólnych algorytmów dla tego problemu. Rozdział drugi zakończył się prezentacją nowego algorytmu przeszukiwania z zakazami dla dowolnych posetów, w którym wykorzystano łańcuchy silnie- i półsilnie zachłanne, a więc stanowiący rozwinięcie algorytmu Sysły. W dalszych rozdziałach zawarto eksperymenty obliczeniowe pokazujące, że stosując zaproponowany algorytm często otrzymuje się rozsądną aproksymację liczby skoków.

Otwartym problemem pozostaje pytanie o aproksymowalność liczby skoków posetu.

Problem 4.3.5. Podać wielomianowy algorytm aproksymacyjny dla liczby skoków posetu lub udowodnić, że taki algorytm nie istnieje.

Trzeci, najobszerniejszy rozdział, dotyczy posetów przedziałowych. Zaprezentowano i zbadano doświadczalnie znane algorytmy aproksymacyjne dla minimalizacji liczby skoków na posetach przedziałowych. Wyodrębniono przykłady trudnych instancji wejściowych. Gruntownie przetestowano dotychczas znane algorytmy zarówno na posetach losowych, jak i trudnych. Następnie przedstawiono nowy algorytm aproksymacyjny dla klasy posetów przedziałowych, będący rozwinięciem algorytmu Mitas. Udowodniono, że podany algorytm ma współczynnik aproksymacji ograniczony przez 1.484. Ponadto, dla posetów przedziałowych sformułowano algorytm dokładny wyznaczający liczbę skoków w czasie $\mathcal{O}(1.47^n \cdot \mathrm{poly}(n))$. Opisano też algorytm genetyczny dla tej klasy posetów.

Następujące problemy teoretyczne pozostają aktualne.

Problem 4.3.6. Obniżyć współczynnik aproksymacji dla problemu skoków na posetach przedziałowych.

Problem 4.3.7. Obniżyć złożoność czasową dokładnego obliczania liczby skoków na posetach przedziałowych.

W czwartym rozdziale zaprezentowano częściowe wyniki dla klasy posetów dwuwymiarowych. Wykazano, że jeśli ograniczy się rozważania do dwuwymiarowych posetów przedziałowych, to istnieje algorytm $\frac{4}{3}$-aproksymacyjny dla liczby skoków. Przetestowano też algorytm przeszukiwania z zakazami na posetach dwuwymiarowych.

Następujące problemy pozostają otwarte.

Problem 4.3.8. Podać wielomianowy algorytm aproksymacyjny dla liczby skoków posetu dwuwymiarowego.

Problem 4.3.9. Odpowiedzieć na pytanie, czy istnieje wielomianowy algorytm obliczający liczbę skoków posetu dwuwymiarowego.

Co więcej, ostatni problem jest otwarty także na przedziałowych posetach dwuwymiarowych.

Spis cytowanej literatury

[1] A.V. Aho, M.R. Garey, J.D. Ullman, The transitive reduction of a directed graph, *SIAM J. Comput.* 1 (1972), 131–137.

[2] A. von Arnim, C. de la Higuera, Computing the jump number on semi-orders is polynomial, *Discrete Appl. Math.* 51, 219–232 (1994).

[3] G. Ausiello et al., *Complexity and approximation: Combinatorial optimization problems and their approximability properties*, Springer-Verlag, 2003.

[4] V.K. Balakrishnan, *Network optimization*, Chapman & Hall, 1995.

[5] C. Berge, Two theorems in graph theory, *Proc. Natl. Acad. Sci. U.S.* 43 (1957), 842–844.

[6] V. Bouchitté, M. Habib, NP-completeness properties about linear extensions, *Order* 4, 143–154 (1987).

[7] V. Bouchitté, M. Habib, The calculation of invariants for ordered sets, w: *Algorithms and Order*, red. I. Rival , NATO ASI Series, Kluwer Academic Publishers, 231–279 (1989).

[8] A. Brandstädt, The jump number problem for biconvex graphs and rectangle covers of rectangular regions, w: *FCT 1989*, LNCS 380, red. J. Csirik, J. Demetrovics, F. Gécseg, Springer, Heidelberg, 68–77 (1989).

[9] S. Ceroi, A weighted version of the jump number problem on two-dimensional orders is NP-complete, *Order* 20, 1–11 (2003).

[10] S. Ceroi, Jump number of 2-dimensional orders and intersection graphs of rectangles, artykuł niepublikowany (2000).

[11] G. Chaty, M. Chein, Ordered matchings and matchings without alternating cycles in bipartite graphs, *Utilitas Mathematica* 16, 183–187 (1979).

[12] M. Chein, M. Habib, The jump number of dags and posets: an introduction, *Ann. Discrete Math.* 9, 189–194 (1980).

[13] M. Chein, P. Martin, Sur le nombre de sauts d'une forét, *C.R. Acad. Sci. Paris (A)* 275 (1972), 159–161.

[14] C.J. Colbourn, W.R. Pulleyblank, Minimizing setups in ordered sets of fixed width, *Order* 1, 225–229 (1985).

[15] G.P. Cornuéjols, D. Hartvigsen, W. Pulleyblank, Packing subgraphs in a graph, *Oper. Res. Lett.* 1, 139–143 (1982).

[16] E. Dahlhaus, The computation of the jump number of convex graphs, w: *ORDAL 1994*, LNCS 831, red. V. Bouchitté, M. Morvan, Springer, Heidelberg, 176–185 (1994).

[17] R. Diestel, *Graph theory*, Graduate Texts in Mathematics 173, Springer, 2006.

[18] R.P. Dilworth, A decomposition theorem for partially ordered sets, *Ann. Math.* 51, 161–166 (1950).

[19] M. Dorigo, *Optimization, Learning and Natural Algorithms*, rozprawa doktorska, Politecnico di Milano, 1992.

[20] D. Duffus, I. Rival, P. Winkler, Minimizing setups for cycle-free ordered sets, *Proc. Amer. Math. Soc.* 85, 509–513 (1982).

[21] J. Edmonds, Paths, trees, and flowers, *Canad. J. Math.* 17 (1965), 449–467.

[22] M.H. El-Zahar, J.H. Schmerl, On the size of jump-critical ordered sets, *Order* 1 (1984), 3–5.

[23] M.H. El-Zahar, I. Rival, Examples of jump-critical ordered sets, *SIAM J. Algebr. Discrete Meth.* 6 (1985), 713–720.

[24] M.H. El-Zahar, On jump-critical posets with jump-number equal to width, *Order* 17 (2000), 93–101.

[25] R. Duh, M. Fürer, Approximation of k-set cover by semi-local optimization, *Proc. STOC 1997*, 256–264 (1997).

[26] U. Faigle, R. Schrader, A setup heuristic for interval orders, *Oper. Res. Lett.* 4, 185–188 (1985).

[27] S. Felsner, A 3/2-approximation algorithm for the jump number of interval orders, *Order* 6, 325–334 (1990).

[28] S. Felsner, *Interval orders: combinatorial structure and algorithms*, rozprawa doktorska, Technische Universität Berlin, Fachbereich Mathematik, 1992.

[29] P.C Fishburn, *Interval orders and interval graphs. A study of partially ordered sets*, Wiley, New York (1985).

[30] M.R. Garey, D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman and Company, 1979.

[31] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

[32] M. Habib, Comparability invariants, w: *Orders: Description and Roles*, Ann. Discrete Math. 23 (1984), red. M. Pouzet, D. Richard, pp. 371–386.

[33] M. Habib, R.H. Möhring, G. Steiner, Computing the bump number is easy, *Order* 5, 107–129 (1988).

[34] D.B. Hartvigsen, *Extensions of matching theory*, rozprawa doktorska, Carnegie-Mellon University, Pittsburgh, 1984.

[35] M. Held, R.M. Karp, A dynamic programming approach to sequencing problems, *J. Soc. Ind. Appl. Math.* 10 (1962), 196–210.

[36] P. Hell, D.G. Kirkpatrick, On the complexity of general graph factor problems, *SIAM J. Comput.* 12 (1983) 601–609.

[37] P. Hell, D.G. Kirkpatrick, Packings by cliques and by finite families of graphs, *Discrete Math.* 49 (1984), 45–59.

[38] C. Higuera, L. Nourine, Drawing and encoding two-dimensional posets, *Theor. Comput. Sci.* 175 (1997), 293–308.

[39] R.M. Karp, Reducibility among combinatorial problems, w: *Complexity of Computer Computations*, red. R.E. Miller, J.W. Thatcher, NY: Plenum Press, 1972, pp. 85–103.

[40] H.A. Kierstead, NP-completeness results concerning greedy and super greedy linear extensions, *Order* 3, 123–134 (1986).

[41] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, Optimization by Simulated Annealing, *Science* 220 (4598), 671–680 (1983).

[42] D. Kratsch, S. Kratsch, The jump number problem: exact and parameterized, LNCS 8246, Springer Verlag, 2013, 230–242.

[43] P. Krysztowiak, M.M. Sysło, An experimental study of approximation algorithms for the jump number problem on interval orders, *Discrete Appl. Math.*, w recenzji.

[44] P. Krysztowiak, An improved approximation ratio for the jump number problem on interval orders, *Theor. Comput. Sci.* 513 (2013), 77–84.

[45] P. Krysztowiak, Improved approximation algorithm for the jump number of interval orders, *Electron. Notes Discrete Math.* 40 (2013), 193–198.

[46] P. Krysztowiak, The database of interval orders difficult for the jump number minimizing algorithms, *Ann. UMCS Inf.*, 1 (2011) 15–22.

[47] P. Krysztowiak, `http://forever.studentlive.pl/HardIntervalOrders.aspx`, Dostęp: 16 kwietnia 2014 roku.

[48] P. Krysztowiak, M.M. Sysło, A tabu search approach to the jump number problem, *Algebra Discrete Math.* 18 (2014), No. 1 (w druku).

[49] J. Kuntzmann, A. Verdillon, *Recherche d'un ordre total minimal compatible avec un ordre partial donné*, Sém. Inst. de Math. de Grenoble, 1971.

[50] C. McCartin, An improved algorithm for the jump number problem, *Inform. Process. Lett.* 79, 87–92 (2001).

[51] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer, 1996.

[52] J. Mitas, Tackling the jump number of interval orders, *Order* 8, 115–132 (1991).

[53] R.H. Möhring, Computationally tractable classes of ordered sets, w: *Algorithms and Order*, NATO ASI Series, Kluwer Academic Publishers, red. I. Rival, 105–193 (1989).

[54] H. Müller, Alternating cycle-free matchings, *Order* 7, 11–21 (1990).

[55] A. Ngom, Genetic algorithm for the jump number scheduling problem, *Order* 15, 59–73 (1998).

[56] W.R. Pulleyblank, On minimizing setups in precedence-constrained scheduling, Report No. 81185 – OR, artykuł niepublikowany, 1981.

[57] I. Rival, Optimal linear extensions by interchanging chains, *Proc. Am. Math. Soc.* 89 (1983), 387–394.

[58] I. Rival (ed.), *Graphs and Orders*, NATO ASI Series, D. Reidel Publishing Company, 1985.

[59] I. Rival (ed.), *Algorithms and Orders*, NATO ASI Series, Kluver Academic Press, Dordrecht, 1989.

[60] J. Spinrad, J. Valdes, Recognition and isomorphism of two dimensional partial orders, *Automata, Languages and Programming*, LNCS 154 (1983), 676–686.

[61] G. Steiner, L.K. Stewart, A linear time algorithm to find the jump number of 2-dimensional bipartite orders, *Order* 3, 359–367 (1987).

[62] J.A. Soto, C. Telha, Jump number of two-directional orthogonal ray graphs, w: *IPCO 2011*, LNCS 6655, red. O. Günlük, G.J. Woeginger, Springer-Verlag, 389–403 (2011).

[63] M.M. Sysło, A graph-theoretic approach to the jump-number problem, w: *Graphs and Order* (D. Reidel, Dodrecht, 1985, red. I. Rival), 185–215.

[64] M.M. Sysło, Remarks on Dilworth partially ordered sets, *Proceedings WG '85*, Trauner Verlag, 355–362 (1985).

[65] M.M. Sysło, Minimizing the jump number for partially ordered sets: a graph-theoretic approach, *Order* 1, 7–19 (1984).

[66] M.M. Sysło, K.M. Koh, W.L. Chia, A characterization of bipartite Dilworth posets, w: *Proceedings of the International Conference on Optimization Techniques and Applications*, Singapore, 451–459 (1987).

[67] M.M. Sysło, Minimizing the jump number for partially-ordered sets: a graph-theoretic approach, II, *Discrete Mathematics* 63, 279–295 (1987).

[68] M.M. Sysło, An algorithm for solving the jump number problem, *Discrete Mathematics* 72, 337–346 (1988).

[69] M.M. Sysło, The jump number problem on interval orders: A 3/2 approximation algorithm, *Discrete Math.* 144, 119–130 (1995).

[70] M.M. Sysło, On some new types of greedy chains and greedy linear extensions of partially ordered sets, *Discrete Appl. Math.* 60, 349–358 (1995).

[71] M.M. Sysło, N. Deo, J.S. Kowalik, *Algorytmy optymalizacji dyskretnej z programami w języku Pascal*, Wydawnictwo Naukowe PWN, Warszawa, 1993.

[72] E. Szpilrajn, Sur l'extension de l'ordre partiel, *Fundamenta Mathematicae* 16 (1930), 386–389.

[73] W.T. Trotter, *Combinatorics and partially ordered sets*, The Johns Hopkins University Press, 1992.

[74] V.V. Vazirani, *Algorytmy aproksymacyjne*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2005.