

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Inteligentne systemy

System wspierający zakup i cięcie stali

inż. Jakub Kowalczyk
Numer albumu 300238

promotor
dr inż. Sebastian Plamowski

WARSZAWA 2024

System wspierający zakup i cięcie stali

Streszczenie. Niniejsza praca skupia się na opracowaniu systemu, który wykorzystując modele programowania liniowego, rozwiązuje zmodyfikowany problemu cięcia materiału. W formie podstawowej problem ten polega na maksymalnym wykorzystaniu surowca przy cięciu dłuższych elementów na wymagane krótsze odcinki. Wprowadzona w pracy modyfikacji zakłada, że niektóre elementy mogą zostać skrócone, jeżeli pozwala to na uzyskanie lepszego wzorca cięcia. Ta możliwość zmniejszenia długości nazywana jest w pracy relaksacją.

Zaimplementowany w formie strony internetowej SPA system, wykorzystuje dziesięć różnych modeli programowania liniowego. Każdy z nich odpowiada za rozwiązania problemu cięcia w innym formacie. Korzystający ze strony użytkownik ma możliwość wyboru formatu oraz wprowadzenia własnych danych wejściowych. Wykorzystane w implementacji biblioteki, solwery liniowe i inne narzędzia są otwartoźródłowe, co sprawia, że aplikacja jest w pełni darmowa.

We wstępnie pracy uargumentowano potrzebę zaistnienia takiej aplikacji oraz ogólny jej zarys. Następnie omówione zostały różne metody rozwiązywania problemu cięcia materiału. Zasadniczą część pracy poświęcona jest opracowaniu algorytmów uwzględniających relaksację oraz badaniu ich jakości i wydajności, oraz projektowi aplikacji internetowej. Sformułowano założenia funkcjonalne i systemowe aplikacji, przedstawiono projekt, a także omówiono wykorzystane technologie.

Słowa kluczowe: Problem cięcia materiału, Optymalizacja, Metoda generacji kolumn

System supporting the purchase and cutting of steel

Abstract. This thesis focuses on the development of a system that solves a modified cutting stock problem using linear programming models. In its basic form, this problem involves maximizing the utilization of raw material when cutting longer elements into required shorter sections. The modification introduced in this thesis assumes that some elements can be shortened if it allows for a better cutting pattern. This possibility of reducing lengths is referred to in the thesis as relaxation.

The implemented system, developed as a single-page web application (SPA), uses nine different linear programming models. Each model is responsible for solving the cutting stock problem in a different format. Users of the application can select a format and input their own data. The libraries, linear solvers, and other tools used in the implementation are open-source, making the application entirely free.

The introduction of the thesis justifies the need for such an application and provides a general outline of its purpose. Subsequently, various methods of solving the cutting stock problem are discussed. The core part of the thesis is devoted to developing algorithms that incorporate relaxation, evaluating their quality and performance, and designing the web application. Functional and system requirements for the application are defined, the design is presented, and the utilized technologies are discussed.

Keywords: Cutting stock problem, Optimization, Column generation method

Spis treści

1. Wstęp	7
1.1. Cel pracy	8
1.2. Charakterystyka pracy	8
1.3. Struktura pracy	9
2. Problem cięcia materiału w literaturze	10
2.1. Metoda generacji kolumn	10
2.2. Metoda podziału, cięć i cen	14
2.3. Formulacja przepływu w sieci łukowej	14
2.4. Algorytm pakowania pojemników	15
2.5. Podsumowanie	16
3. Relaksacja długości prętów finalnych w problemie cięcia materiału	17
3.1. Minimalizacja kosztu prętów bazowych	17
3.1.1. Relaksacja manualna	17
3.1.2. Przykład	21
3.1.3. Relaksacja automatyczna	24
3.1.4. Relaksacja krokowa	27
3.2. Minimalizacja odpadów	29
3.2.1. Relaksacja manualna	29
3.2.2. Relaksacja automatyczna	30
3.2.3. Relaksacja krokowa	31
3.2.4. Model dodatkowy	32
4. System wspierający cięcie	33
4.1. Założenia funkcjonalne i niefunkcjonalne systemu	33
4.2. Wykorzystane technologie	34
4.2.1. Frontend	34
4.2.2. Backend	35
4.3. Struktura klas	35
4.4. Przykłady użycia	39
4.5. Testy aplikacji	46
4.6. Proces instalacji	48
4.6.1. Pobranie zależności	48
4.6.2. Uruchomienie	48
5. Testy algorytmów	50
5.1. Przygotowanie danych	50
5.2. Testy modeli programowania liniowego	51
5.2.1. Efektywność relaksacji manualnej i automatycznej	51
5.2.2. Wydajność relaksacji manualnej i automatycznej	53

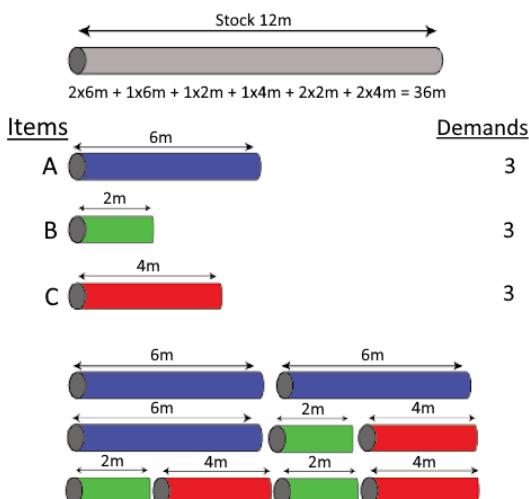
0. Spis treści

5.2.3. Modyfikacja kosztu relaksacji	54
5.2.4. Testy relaksacji krokowej	58
5.2.5. Problem cięcia z możliwością magazynowania prętów	59
5.2.6. Porównanie z algorytmem pakowania pojemników	60
5.2.7. Porównanie solwerów	62
6. Wnioski	64
Bibliografia	67
Spis rysunków	69

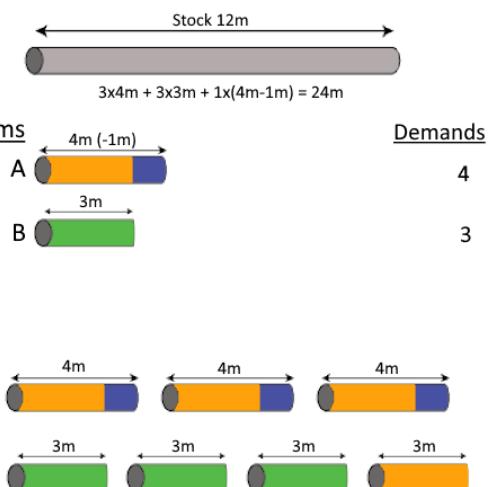
1. Wstęp

Współczesny przemysł, szczególnie sektory związane z produkcją i przetwórstwem materiałów, nieustannie poszukują efektywnych metod optymalizacji procesów produkcyjnych. Jednym z kluczowych zagadnień w tej dziedzinie jest problem cięcia materiałów. Polega on na minimalizacji straty surowca podczas cięcia większych elementów na mniejsze. Problem ten w formie jednowymiarowej badany jest od dawna. Już w 1939 roku sformułowany został przez L. Kantorowicza[1]. Dwanaście lat później, Kantarowicz i V. A. Zalgaller zaproponowali rozwiązanie go za pomocą programowania liniowego[2]. Pomysł przełomowy, na którym praca ta będzie się opierać, padł w 1961 roku, kiedy to P. C. Gilmore oraz R. E. Gomory przedstawili model programowania liniowego rozwiązujący problem cięcia materiału z ograniczoną liczbą zmiennych[3]. Technika, którą wtedy zastosowali, została później nazwana metodą generacji kolumn.

Choć dzisiejsze algorytmy są w stanie znajdować optymalne rozwiązania niedużych instancji problemu cięcia materiału w podstawowej formie, tak dalej prowadzone są badania nad różnego rodzaju wariantami tego zagadnienia[4]. Niniejsza praca analizować będzie odmianę, w której to długość składników zamówienia może być potencjalnie skracana, jeżeli umożliwiłoby to uzyskanie lepszych wzorców cięcia. Algorytm działający w takim wariantie może znaleźć zastosowanie przede wszystkim w branży budowlanej. Wykorzystywane są tam stalowe pręty, których długość w zależności od ich umiejscowienia, może się nieznacznie różnić od tej z projektu. Niewielkiemu skróceniu, to jest rzędu kilku procent, poddać można np. pręty wzmacniające fundamenty lub schody. Oczywiście taką redukcję długości, określającą w tej pracy mianem relaksacji, powinno się stosować jedynie, jeżeli przynosi ona jakiekolwiek korzyści.



Rysunek 1.1. Problem cięcia materiału



Rysunek 1.2. Zastosowanie relaksacji

1.1. Cel pracy

Praca ma dwa główne cele. Pierwszym z nich jest zbadanie algorytmów rozwiązywających problem cięcia z relaksacją. W celu jego realizacji zaplanowano wykonanie następujących zadań:

1. Przegląd najpopularniejszych metod rozwiązywania klasycznego problemu cięcia.
2. Rozwinięcie istniejących modeli o możliwość relaksacji.
3. Implementacja algorytmów.
4. Przeprowadzenie szczegółowych testów weryfikacyjnych i porównawczych.
5. Sformułowanie wniosków na podstawie uzyskanych wyników.

Drugim celem pracy jest stworzenie aplikacji umożliwiającej definiowanie parametrów wejściowych do takich algorytmów w prosty i intuicyjny sposób oraz czytelną prezentację wyników. Oba cele są ze sobą ściśle powiązane – badania nad algorytmami dostarczają podstaw do implementacji praktycznego narzędzia wspierającego ich zastosowanie.

1.2. Charakterystyka pracy

Do rozwiązywania problemu cięcia materiału wykorzystywane mogą być różne metody, np. programowanie dynamiczne[5] modele przepływu łukowego [6] lub wspomniana już metoda generacji kolumn. Często też stosuje się modele hybrydowe[7], łączące zalety różnych technik. Ponieważ praca ta skupia się na zbadaniu wpływu relaksacji, a nie na sformułowaniu algorytmu optymalnego, stopień złożoności wszystkich zaproponowanych tu modeli, będzie ograniczał się jedynie do pewnego rozwinięcia metody generacji kolumn.

W ramach pracy zaimplementowana została aplikacja internetowa, tytułowy system wspierający cięcie stali. Jego użytkownicy mogą tworzyć własne zamówienia, składające się ze zbioru prętów bazowych oraz ze zbioru prętów finalnych. Ponieważ wszystkie wykorzystywane w programie biblioteki, frameworki oraz solwery¹ są otwartoźródłowe, nienalożone są żadne ograniczenia co do wielkości zamówień. To, że aplikacja jest w pełni darmowa, jest jej istotną cechą. Na rynku bowiem znajdziemy wiele innych programów oferujących rozwiązywanie problemu cięcia, lecz zdecydowana większość z nich jest nastawiona na zysk poprzez blokowanie użytkownikom dostępu do wszystkich funkcji. Od innych dostępnych rozwiązań, zaprojektowana tutaj aplikacja przede wszystkim różni się wykorzystywanymi modelami programowania liniowego. Dostępnych do użytku jest w sumie dziewięć, w czym siedem z nich korzysta z relaksacji. Różnią się one sposobem nakładania relaksacji oraz głównym celem, którym może być minimalizacja kosztu zamówienia albo minimalizacja odpadów.

Praca ta ma na celu nie tylko rozwinięcie istniejących technik, ale również wniesienie nowej perspektywy do zagadnienia problemu cięcia materiału. Dzięki zastosowaniu relaksacji długości składników zamówienia, przedstawione rozwiązania mają potencjał

¹ Solwer (ang. *solver*) to oprogramowanie służący do rozwiązywania problemów matematycznych, w tym problemów liniowego programowania matematycznego.

znać zastosowanie w sytuacjach, gdzie tradycyjne modele byłyby niewystarczające lub nieoptymalne. W efekcie uzyskane wyniki pracy mogą posłużyć jako punkt wyjścia do dalszych badań i rozwoju bardziej zaawansowanych narzędzi wspomagających procesy produkcyjne.

1.3. Struktura pracy

W kolejnym rozdziale przeanalizowano wybrane metody rozwiązywania problemu cięcia, które są stosowane w literaturze. Ponieważ liczba wszystkich stosowanych technik jest zbyt duża, w pracy ograniczono się jedynie do tych, na których wzorowano się w pracy lub tych dających obecnie najlepsze wyniki.

Następnie przedstawiono opracowane w ramach pracy modele programowania liniowego rozszerzone o możliwość relaksacji. Uwzględniono ich model matematyczny, jego odwzorowanie w języku AMPL² oraz korzystające z nich algorytmy.

Rozdział 4 zawiera informacje o budowie aplikacji internetowej. Przedstawiono wymagania funkcjonalne aplikacji oraz szczegółowo omówiono jej poszczególne komponenty.

W rozdziale 5 przedstawiono testy porównawcze i weryfikacyjne algorytmów z relaksacją. Omówiono wyniki oraz przedstawiono wnioski.

Na końcu pracy sformułowano krótkie podsumowanie oraz wskazano dalsze kierunki rozwoju.

² AMPL (ang. *A Mathematical Programming Language*) to język programowania przeznaczony do modelowania i rozwiązywania problemów optymalizacyjnych.

2. Problem cięcia materiału w literaturze

Przed przystąpieniem do realizacji celów pracy, konieczne było zapoznanie się z dotychczasowymi metodami stosowanymi do rozwiązywania problemu cięcia. Ich analiza pozwoliła wybrać stosowne podejście, które dało się rozwinać o możliwość relaksacji. Ponieważ problem cięcia materiału jest już badany od niemalże wieku, powstała znaczna ilość prac naukowych na jego temat. Autorzy stosują w nich różne techniki do jego rozwiązania, a także tak jak ta praca, definiują nowe warianty problemu. Z uwagi na to, że praca skupia się przede wszystkim na cięciu stali, składniki bazowe (te, które podlegają cięciu) i składniki finalne (składniki zamówione) będą określane od tej pory jako pręty bazowe i pręty finalne. Przedstawione badania i wyniki mają jednak zastosowanie do wszelkich innych materiałów, gdzie problemem jest określenie podziału w jednym wymiarze.

2.1. Metoda generacji kolumn

W artykule "A Linear Programming Approach to the Cutting Stock Problem I" [3] P. C. Gilmore oraz R. E. Gomory przedstawili innowacyjny sposób rozwiązywania problemu cięcia za pomocą programowania liniowego - metodę generacji kolumn. Był to kluczowy krok w badaniach nad problemem, który stał się później podstawą wielu prac naukowych. Jego cechą charakterystyczną było zredukowanie liczby zmiennych decyzyjnych w problemie do niezbędnego minimum. Próbując rozwiązać zagadnienie "tradycyjną" metodą korzystając z programowania liniowego, dla większej ilości danych wejściowych (to jest większego zamówienia) z pewnością natkniemy się na problem zbyt dużej liczby zmiennych wymaganych w modelu. Każdy unikalny wzorzec, gdzie wzorcem nazywamy sposób cięcia pręta bazowego, będzie takową zmienną. Stosując metodę generacji kolumn, wychodzimy z założenia, że w rozwiązaniu końcowym większość możliwych do utworzenia wzorców nie będzie w ogóle użyta. W związku z tym stopniowo generowane są tylko te wzorce, które mają potencjał wejść do rozwiązania końcowego.

Ogólny algorytm metody generacji kolumn zakłada, że oryginalny problem da się podzielić na dwa inne problemy: problem główny oraz podproblem. Problem główny odpowiada problemowi oryginalnemu z różnicą taką, że ma dostęp tylko do części jego zmiennych. Zadaniem podproblemu jest generowanie kolejnych zmiennych, które trafiając do problemu głównego, poprawią jego funkcję celu.

Algorytm 1 Ogólny algorytm metody generacji kolumn

Krok 1. Zainicjalizuj problem główny i podproblem.**Krok 2.** Rozwiąż problem główny w dziedzinie liczb rzeczywistych.**Krok 3.** Rozwiązuając podproblem, spróbuj znaleźć zmienną, która poprawi wynik problemu głównego.**Krok 4.** Jeżeli taka zmienna istnieje, wróć do **kroku 2.** w przeciwnym wypadku przejdź do **kroku 5.****Krok 5.** Rozwiąż problem główny w dziedzinie liczb całkowitych i zakończ algorytm

W przypadku problemu cięcia materiału problemem głównym(1) jest minimalizacja liczby użyć poszczególnych wzorców p_j ze zbioru wszystkich wzorców M . Jest ona równa liczbie wykorzystanych bazowych prętów. Wzorce opisane są przez wektory, czyli kolumny O_j macierzy O . Wzorce określają sposób podziału pręta bazowego na krótsze wymagane odcinki. Współrzędna $o_{i,j}$ wektora O_j definiują liczbę odcinków typu i we wzorcu j . Przez typ odcinka rozumiany jest odcinek o długości w_i należący do zbioru prętów finalnych. Jedynym ograniczeniem (2) problemu głównego jest wymuszenie spełnienia zamówienia na liczbę d_i wymaganych typów odcinków. Ograniczenie to musi być spełnione dla wszystkich N typów prętów finalnych.

$$\min_p \sum_{j=1}^M p_j \quad (1)$$

$$\text{przy ograniczeniu: } \forall i \in N \quad \sum_{j=1}^M o_{i,j} p_j \geq d_i \quad (2)$$

$$\forall j \in M \quad p_j \geq 0 \quad (3)$$

Rozwiązuając tak sformułowany problem główny, otrzymamy konkretną liczbę użyć każdego wzorca p_j , która pozwoli nam wykorzystać jak najmniej prętów bazowych. Wymagane jest jednak posiadanie pewnej puli wzorców M , która jest w stanie spełnić założenia modelu. Najprostszym sposobem na jej wygenerowanie jest wyznaczenie $\lfloor \sigma / w_i \rfloor$ dla każdego $\forall i \in N$, gdzie σ jest długością pręta bazowego. Zbiór wzorców powiększany będzie potencjalnie po rozwiązaniu podproblemu.

Zadaniem podproblemu jest wyznaczenie takich wzorców, które pozwolą na poprawienie rozwiązania problemu głównego. Przybiera on postać typowego problemu pakowania plecakowego(4-6). Plecakiem w tym przypadku jest pręt bazowy, w którym staramy się umieścić jak najdroższe pręty finalne. Cenę prętów finalnych c_i pozyskać można, rozwiązując problem dualny głównego problemu. Uzyskane w ten sposób zmienne dualne informują nas, jaki jest aktualny koszt inkrementacji ilości zamówionych odcinków d_i wyrażony w surowych prętach. Potrzeba wyznaczenia zmiennych dualnych dla podpro-

2. Problem cięcia materiału w literaturze

blemu oznacza, że problem główny rozwiązywany jest w dziedzinie liczb rzeczywistych. Problemy liniowe bez ograniczeń całkowitoliczbowych są też mniej złożone obliczeniowo niż problemy całkowitoliczbowe.

$$\max_x \sum_{i=1}^N c_i x_i \quad (4)$$

przy ograniczeniach: $\sum_{i=1}^N w_i x_i \leq \sigma \quad (5)$

$$\forall i \in N \quad (x_i \geq 0 \wedge x_i \in \mathbb{Z}) \quad (6)$$

Wzorcem, który w największym stopniu poprawi rozwiązanie problemu głównego, jest ten o maksymalnym możliwym koszcie(4). Na koszt składają się cena prêtów finalnych c_i oraz liczba ich wystąpień x_i w nowym wzorcu. Rozwiązania szukamy w dziedzinie liczb całkowitych 6. Gilmore i Gomory proponują w tym celu wykorzystać programowanie dynamiczne[3], lecz obecnie częściej korzysta się z metody podziału i ograniczeń³ (ang. *branch and bound, B&B*). Ograniczenie (5) zapewnia, że suma użytych długości w_i elementów we wzorcu X nie przekroczy długości pręta bazowego σ . Jednak nie wszystkie wyznaczone w ten sposób wzorce będą poprawiać wynik problemu głównego. Możliwym jest bowiem, że zbiór, który potrzebny jest do optymalnego cięcia, został już skompletowany. O tym, czy nowo utworzony wzorzec poprawia wynik problemu głównego dowiedzieć się można, porównując koszt wzorca z kosztem pręta bazowego γ (7).

$$\sum_{i=1}^N c_i x_i > \gamma \quad (7)$$

Jeżeli koszt wzorca jest większy, to poprawia on rozwiązanie. W takim wypadku macierz O rozszerzana jest o nowy wzorzec, który dodawany jest do niej jako kolumna. W przeciwnym wypadku przechodzi się do kroku 5. algorytmu (1) i rozwiązuje problem główny z ograniczeniami całkowitoliczbowymi. Wykorzystać do tego można technikę podziału i ograniczeń. Gilmore i Gomory proponują też zastosowanie zwykłego przybliżenia do liczb całkowitych, w takim wypadku krok 5. jest pomijany. W przypadku zaprezentowanego powyżej modelu koszt pręta bazowego $\gamma = 1$. Wartość ta ma związek z tym, że wynikiem problemu głównego jest liczba użytych prêtów bazowych. Zwiększenie rozwiązania problemu głównego o koszt jednego bazowego pręta to też zwiększenie go o liczbę jednego pręta.

Powyższy model zakłada, że dostępny jest jeden typ prêtów bazowych. Aby móc obsługi-

³ Metoda podziału i ograniczeń – metoda polegająca na rozbiciu (w formie drzewa) problemu na odpowiednio ograniczone podproblemy i eliminacji tych gałęzi, które nie mogą prowadzić do rozwiązania optymalnego.

giwać większą ich ilość, konieczne jest jego rozwinięcie. W książce Johannes'a Bisschop'a [8] zaproponowany jest model (8), który pozwala ograniczać koszt i ilość prętów bazowych.

$$\min_p \sum_{k=1}^S \sum_{j=1}^M \gamma_k p_{kj} \quad (8)$$

$$\text{przy ograniczeniach: } \forall i \in N \quad \sum_{k=1}^S \sum_{j=1}^M o_{kij} p_{kj} \geq d_i \quad (9)$$

$$\forall k \in S \quad \sum_{j=1}^M p_{kj} \leq \delta_k \quad (10)$$

$$\forall k \in S, \forall j \in M \quad p_{kj} \geq 0 \quad (11)$$

Dochodzi tutaj zbiór prętów bazowych S , cena prętów bazowych γ_k oraz dostępna ilość prętów bazowych δ_k . Wynikiem funkcji celu jest łączny koszt użytych prętów. Pojawia się także ograniczenie (10) niepozwalające przekroczyć liczby dostępnych prętów bazowych. Są to już wszystkie różnice pomiędzy modelem podstawowym a rozszerzonym. Drobnym zmianom ulega zaś stosowany algorytm. Dla każdego pręta bazowego podproblem musi być rozwiązywany oddzielnie. Oznacza to, że dla pięciu prętów bazowych w jednej iteracji algorytmu problem główny rozwiązywany będzie tylko raz, podproblem zaś albo pięć razy, albo do momentu odnalezienia pierwszego poprawiającego rozwiązanie wzorca X . Pozostaje kwestia podejmowania decyzji o dołączeniu wzorca X do macierzy O . W tym modelu pręty bazowe mają wprost podany koszt γ_k , jednakże uwzględnić trzeba również wpływ ograniczenia (10) na potencjalny wynik. Założmy, że zmienna dualna α_k odpowiada ograniczeniu (10). Mówi ona nam, jak bardzo zmieni się wynik problemu głównego, jeżeli zwiększymy dostępność prętów bazowych δ_k o jedną jednostkę. Zauważmy więc, że nowo wyznaczony wzorzec dla pręta bazowego k musi nie tylko posiadać większy koszt niż koszt pręta bazowego γ_k , ale także nadrobić stratę w kosztach związaną z wykorzystaniem pręta o ograniczonej dostępności. Ostatecznie spełnione musi być równanie (12), aby nowy wzorzec poprawiał rozwiązanie.

$$\sum_{i=1}^N c_i x_i > \gamma_k - \alpha_k \quad (12)$$

Powyższy model programowania liniowego cechuje się wysoką wydajnością nawet w przypadku skomplikowanych zamówień, przewyższających wielkością te, które są spotykane w praktyce[9] (zostanie to jeszcze poddane testom w następnych rozdziałach). Ma on jednak jedną wadę. Zbiór wzorców, który otrzymujemy po wykonaniu kroków 1-4 algorytmu (1) daje optymalne rozwiązanie w dziedzinie liczb rzeczywistych. Po wykonaniu ostatniego kroku, jakim jest rozwiązanie problemu głównego w dziedzinie liczb całkowitych, otrzymamy rozwiązanie optymalne w dziedzinie liczb rzeczywistych, ale nie będące rozwiązaniem całkowitym.

2. Problem cięcia materiału w literaturze

tych, otrzymany wynik nie musi być już optymalny. Ten pojedynczy defekt sprawia, że wciąż poszukiwane i badane są inne metody rozwiązywania problemu cięcia materiału.

2.2. Metoda podziału, cięć i cen

Rozwój algorytmów optymalizacyjnych, jak i wzrost mocy obliczeniowej komputerów w ostatnich latach [10] umożliwił szersze i bardziej efektywne zastosowanie programowania całkowitoliczbowego w rozwiązywaniu złożonych problemów decyzyjnych[6]. Tyczy się to także problemu cięcia materiału. Jak wcześniej wspomniano, programowanie całkowitoliczbowe może zostać zastosowane w finalnym etapie metody generacji kolumn zaproponowanej przez Gilmore'a i Gomory'ego. Jednak jego potencjał w tym obszarze jest znacznie większy.

Zamiast wykorzystywać metodę podziału i ograniczeń do wyznaczenia ostatecznego rozwiązania, zastosować można bardziej zaawansowaną technikę – metodę podziału i cen⁴. Od B&B wyróżnia się tym, że w gałęziach przeszukiwanego drzewa też stosowana jest metoda generacji kolumn do wyznaczania nowych wzorców. W ten sposób nie ograniczamy się do zestawu wzorców wygenerowanego przed nałożeniem ograniczeń całkowitoliczbowych[11].

Do metody podziału i cen można dodać jeszcze jedno rozwinięcie. Jest nim wykorzystanie płaszczyzn tnących⁵, np. cięć Gomory'ego, które pomogą zawężać przestrzeń możliwych rozwiązań. Stosuje się je przed podziałem na gałęzie.

W dzisiejszych czasach do najbardziej zaawansowanych algorytmów rozwiązywających problem cięcia materiału zalicza się między innymi te, które wykorzystują metodę podziału, cięć i cen[4], czyli metodę hybrydową wykorzystującą wszystkie wymienione w tym rozdziale techniki.

2.3. Formulacja przepływu w sieci łukowej

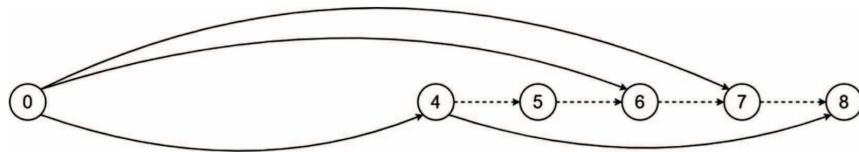
Drugim obiecującym pod względem wyników modelem jest ten formułujący problem cięcia za pomocą sieci przepływów[12] (arc-flow formulation).

W takim modelu, węzły sieci odpowiadają za możliwe pozycje w pręcie bazowym, łuki zaś za wykorzystane pręty finalne lub za odpady. Na przykładowym grafie 2.1 zamodelowany został pręt bazowy o długości $\sigma = 8$ oraz jego przykładowy wzorzec. Składają się na niego pręty finalne o długościach $w_i = (4, 6, 7)$. Linie przerywane oznaczają odpady, linie proste zaś użyte pręty finalne.

Autor tego modelu przedstawia techniki pozwalające na usuwanie łuków niemogących prowadzić do optymalnego rozwiązywania, tym samym zmniejszając rozmiar problemu.

⁴ Metoda podziału i cen (ang. *branch and price*) – technika będąca hybrydą pomiędzy metodą podziału i ograniczeń, a metodą generacji kolumn

⁵ metoda płaszczyzn tnących (ang. *cutting-plane method*) – technika stosowana w programowaniu całkowitoliczbowym, która dodając nowe ograniczenia, eliminuje stopniowo rozwiązania niecałkowitoliczbowe. Technika B&B wykorzystująca cięcia nazywana jest metodą podziału i cięć (ang. *branch and cut*)



Rysunek 2.1. Model problemu cięcia w postaci sieci przepływów[6]

Także i tutaj wykorzystywana jest metoda podziałów i cen, aby nie uwzględniać od razu wszystkich możliwych łuków, co dla większej różnorodności danych wejściowych mogłoby być problematyczne.

2.4. Algorytm pakowania pojemników

Problem cięcia materiału, w którym występuje tylko jeden typ pręta bazowego ($|S| = 1$) oraz ograniczenie (9) jest równością, sklasyfikować można jako problem pakowania pojemników⁶ (ang. *Bin Packing Problem*). Postać pojemnika przyjmuje w tym wypadku pręt bazowy, w którym to "układane" są pręty finalne.

Istnieją skuteczne algorytmy heurystyczne, pozwalające rozwiązywać problem pakowania pojemników bez potrzeby użycia programowania liniowego. Przykładowo heurystyka pierwszego dopasowania malejącego (ang. *First-Fit Decreasing*) pozwala rozwiązać problem w czasie $O(n \log n)$, gdzie n to liczba zamówionych prętów[13]. Uzyskane rozwiązanie będzie nie gorsze niż $\frac{11}{9} OPT + \frac{6}{9}$, gdzie OPT to liczba pojemników w rozwiązaniu optymalnym[14]. Kolejne kroki algorytmu wykorzystującego tę heurystykę przedstawione zostały poniżej.

Algorytm 2 Pakowanie do pojemników z heurystyką *First Fit Decreasing*

Krok 1. Posortuj przedmioty od najdłuższego do najkrótszego. Weź pierwszy z nich.

Krok 2. Znajdź pojemnik, w którym przedmiot zmieści się.

Krok 3. Jeżeli taki pojemnik istnieje, umieść tam przedmiot. Jeżeli takiego pojemnika brak, utwórz nowy pojemnik i umieść w nim przedmiot.

Krok 4. Weź kolejny przedmiot i wróć do **kroku 2**. W przypadku ich braku zakończ algorytm.

W odróżnieniu od innych metod przedstawionych w tym rozdziale, algorytm 2 charakteryzuje się niskim poziomem skomplikowania. Ceną tej prostoty jest jednak brak obsługi różnych typów prętów bazowych.

⁶ Problem pakowania pojemników – problem którego celem jest umieszczenie wszystkich dostępnych przedmiotów w jak najmniejszej liczbie pojemników o stałym rozmiarze.

2.5. Podsumowanie

W rozdziale tym nie wymieniono wszystkich stosowanych przez badaczy metod rozwiązywania problemu cięcia materiału, gdyż jest ich zdecydowanie za dużo. Zawrzeć można byłoby jeszcze techniki takie jak programowanie dynamiczne[5], głębokie uczenie ze wzmacnieniem[15] oraz wiele różnych heurystyk do opisanych wyżej algorytmów. Do najbardziej zaawansowanych obecnie modeli[4], należą te opisane w podrozdziałach 2.2 i 2.3, jednakże zaimplementowanie ich wymaga podejścia skupiającego się na jednym konkretnym typie problemu, co może ograniczać ich zastosowanie w szerszym kontekście. W pracy skoncentrowano się więc na rozbudowie modelu korzystającego jedynie z generacji kolumn, ponieważ cechuje się on największą elastycznością i uniwersalnością. Stanowi on nie tylko solidną podstawę do dalszego rozwoju, ale również pozwala w przejrzysty sposób wprowadzać nowe ograniczenia, co jest konieczne do zrealizowania celów pracy. Oczywiście nie uniemożliwia to w przyszłości zastosowania bardziej wyrafinowanych technik, gdyż jak można zauważyć, wszystkie opisane wyżej metody bazują na generacji kolumn.

3. Relaksacja długości prętów finalnych w problemie cięcia materiału

W tym rozdziale omówione zostaną przygotowane w ramach pracy rozwinięcia modelu generacji kolumn (8) o relaksację długości prętów finalnych. Powstało w sumie siedem nowych modeli. Trzy z nich minimalizują liczbę lub koszt prętów bazowych. Cztery pozostałe minimalizują ilość odpadów. Wszystkie modele zostały sformułowane i zaimplementowane w środowisku dedykowanym do modelowania matematycznego – **AMPL**.

3.1. Minimalizacja kosztu prętów bazowych

3.1.1. Relaksacja manualna

Pierwszy model uwzględniający relaksację zakłada, że w danych wejściowych podana będzie maksymalna wartość relaksacji m_i dla każdego pręta finalnego. Z uwagi na to, że to od użytkownika zależy możliwy zakres skracania prętów, ten sposób relaksacji nazywamy manualnym. W końcowym wyniku powinny zawierać się jedynie te zrelaksowane wzorce, które rzeczywiście pozwalają poprawić rozwiązanie. Skracanie długości nie powinno też być stosowane nadmiernie. To znaczy, że jeżeli maksymalna relaksacja jest równa $m_i = 100$, a do uzyskania nowego wzorca wystarczy skrócenie długości w_i o 50 jednostek, to relaksacja powinna być równa $r_i = 50$.

$$\max_{x,r} \sum_{i=1}^N c_i x_i - r_i \lambda_i \quad (13)$$

$$\text{przy ograniczeniach: } \sum_{i=1}^N w_i x_i - r_i \leq \sigma \quad (14)$$

$$\forall i \in N \quad r_i \leq m_i x_i \quad (15)$$

$$\forall i \in N \quad (x_i \geq 0 \wedge r_i \geq 0) \quad (16)$$

$$\forall i \in N \quad x_i, r_i \in \mathbb{Z} \quad (17)$$

Aby móc wprowadzić zagadnienie relaksacji do modelu cięcia (8-11), rozszerzamy funkcję celu podproblemu (4) i ograniczenie podproblemu (5) o zmienne odpowiadające wartości relaksacji r_i . Zawierają one informacje, o ile w sumie została skrócona długość wszystkich prętów finalnych w_i w nowym wzorcu. Wartość skrócenia pojedynczego pręta można więc uzyskać licząc $\frac{r_i}{x_i}$. Aby relaksacja nie była wprowadzana zachłannie, w funkcji celu (13) dodana jest stała λ_i , będąca kosztem relaksacji. Sposób przypisywania wartości tej stałej zostanie szczegółowo omówiony w podrozdziale 3.1.3. Ogólnie rzecz biorąc, wartość ta powinna być niewielka, aby nie ingerowała zanadto w końcowy wynik. Pojawia się

3. Relaksacja długości prętów finalnych w problemie cięcia materiału

także dodatkowe ograniczenie (15), zapewniające, że wartość relaksacji r_i nie przekroczy progu maksymalnej relaksacji m_i .

Listing 1 przedstawia model AMPL problemu głównego metody generacji kolumn (8-11) z manualną relaksacją. Uwzględniono też deklarację wszystkich zmiennych i parametrów użytych w modelu wraz z opisem. Symbol **[dat.]** oznacza, że wartość parametru jest pobierana z oddzielnego pliku, który to przechowuje dane wejściowe problemu.

Listing 1. Problem główny metody generacji kolumn z relaksacją manualną w AMPL

```
1  set STOCK;                                # zbiór prętów bazowych [.dat]
2  param stockLengths {STOCK} > 0;           # długość prętów bazowych [.dat]
3  param stockNum {STOCK} > 0;                # ilość prętów bazowych [.dat]
4  param stockCost {STOCK} > 0;              # koszt prętów bazowych [.dat]
5  set ORDERS;                               # zbiór prętów finalnych [.dat]
6  param orderLengths {ORDERS} > 0;          # długość prętów finalnych [.dat]
7  param orderNum {ORDERS} > 0;              # zamówiona ilość prętów finalnych [.dat]
8  param maxRelax {ORDERS};                  # maksymalna relaksacja prętów finalnych [.dat]
9
10 param nPAT {STOCK} integer >= 0;           # ilość wzorców dla każdego pręta bazowego
11 set PATTERNS {i in STOCK} := 1..nPAT[i];    # zbiór wzorców
12 param lfep {i in STOCK,ORDERS,PATTERNS[i]} integer >= 0; # liczba prętów we wzorcu
13 param rfep {i in STOCK,ORDERS,PATTERNS[i]} integer >= 0; # relaksacja prętów we wzorcu
14 var usedPatterns {i in STOCK, PATTERNS[i]} integer >= 0;   # wykorzystane wzorce
15
16 # PROBLEM GŁÓWNY
17 minimize Cost:
18   sum {i in STOCK, j in PATTERNS[i]} stockCost[i] * usedPatterns[i,j];
19 subj to FillOrder {x in ORDERS}:
20   sum {i in STOCK, j in PATTERNS[i]} lfep[i,x,j] * usedPatterns[i,j] >= orderNum[x];
21 subj to StockLimit {i in STOCK}:
22   sum{ j in PATTERNS[i]} usedPatterns[i,j] <= stockNum[i];
```

Listing 2 przedstawia podproblem plecakowy z manualną relaksacją (13-17). Ponieważ inne proponowane w tym rozdziale modele będą w pewnym stopniu do siebie podobne (w szczególności problem główny), w następnych listingach pojawiać się będą tylko ich zasadnicze różnice.

Listing 3 prezentuje pierwszą część ogólnego algorytmu metody generacji kolumn (1), czyli inicjalizację problemu głównego i podproblemu. W pierwszym kroku wybierany jest solwer oraz jego opcje. Musi on być zdolny do rozwiązywania problemów liniowych, jak i mieszanych całkowitoliczbowych. Następnie deklarowane są problemy. Opcja *relax_integrality* odpowiada za ustalenie, czy ograniczenia całkowitoliczbowe obowiązują w problemie, zaś *presolve* wpływa na stopień uproszczenia problemu przed jego rozwiąza niem. Linie 9-22 odpowiadają za inicjalizację parametrów wstępnych. Jest tu np. tworzony zbiór wzorców początkowych O , wyznaczany operacją $\lfloor \frac{\sigma_k}{w_i} \rfloor$

Listing 2. Podproblem oraz dodatkowe parametry dla relaksacji manualnej

```

1 param relaxCost {ORDERS} default 0.00001; # koszt relaksacji
2 param price {ORDERS} default 0.0; # koszt prętów finalnych
3 param stockLength > 0; # długość pręta bazowego
4 var Use {ORDERS} integer >= 0; # użyte pręty we wzorze
5 var Relax {ORDERS} integer >= 0; # wartość relaksacji
6
7 # PROBLEM PLECAKOWY Z RELAKSACJĄ
8 maximize ReducedCost:
9   sum {x in ORDERS} (price[x] * Use[x] – Relax[x] * relaxCost[x]);
10 subj to LengthLimit:
11   sum {x in ORDERS} (orderLengths[x] * Use[x] – Relax[x]) <= stockLength;
12 subj to RelaxLimit {x in ORDERS}:
13   Relax[x] <= maxRelax[x]*Use[x];
14
15 -----
16 #DODATKOWE PARAMETRY WYKORZYSTYWANE W ALGORYTMIE
17 param relaxCostMultiplier default 1; # multiplikator koszta relaksacji [.dat]
```

Listing 3. Inicjalizacja problemów

```

1 problem Cutting_Opt: usedPatterns, Cost, FillOrder, StockLimit;
2   option relax_integrality 1;
3   option presolve 0;
4
5 problem Pattern_Gen: Use, Relax, ReducedCost, LengthLimit, RelaxLimit;
6   option relax_integrality 0;
7   option presolve 1;
8
9 param temp;
10 for {i in STOCK} {
11   let nPAT[i] := 0;
12
13   for {x in ORDERS} {
14     let temp := floor(stockLengths[i] / orderLengths[x]);
15     if temp > 0 then {
16       let nPAT[i] := nPAT[i] + 1;
17       let lfep[i,x,nPAT[i]] := temp;
18       let {x2 in ORDERS: x2 <> x} lfep[i,x2,nPAT[i]] := 0;
19       let {x2 in ORDERS} rfepl[i,x2,nPAT[i]] := 0;
20     }
21   }
22 };
```

Zasadnicza część algorytmu przedstawiona jest w listingu 4. Odpowiada ona krokom 2-5 algorytmu (1). Polecenie *solve Cutting_Opt* przekazuje problem główny do rozwiązyania solwerowi. Następnie zmienne dualne ograniczenia (9) przypisywane są jako ceny c_i prętów finalnych. Znając ceny można przystąpić do rozwiązywania podproblemu *Pattern_Gen*.

Odbiera się to w pętli iterującej po każdym pręcie bazowym. Nowe wzorce X poprawiające rozwiązanie problemu głównego są dodawane do zbioru definicji wzorców $lfep$ i zbioru wykorzystanych relaksacji $rfep$. W liniach 35-46 wyznaczany jest koszt relaksacji dla każdego typu pręta finalnego. Idea techniki wyznaczania kosztu relaksacji przedstawiona jest w podrozdziale 3.1.3. W linii 41 pojawia się parametr $relaxCostMultiplier$. Pozwala on skalować koszt relaksacji. Wartość tego parametru może być ustalana przez użytkownika, dając mu dodatkową kontrolę nad procesem optymalizacji.

Listing 4. Algorytm metody generacji kolumn

```

30 repeat
31 {
32     solve Cutting_Opt;
33     let {x in ORDERS} price[x] := FillOrder[x].dual;
34     for {i in STOCK} {
35         for {x in ORDERS} {
36             if relaxCostMultiplier = 1 then {
37                 let relaxCost[x] := 0.00001;
38             }
39             else {
40                 let minRelaxCost := price[x] / orderLengths[x];
41                 let relaxCost[x] := (minRelaxCost * 0.75) * relaxCostMultiplier;
42                 if relaxCost[x] < 0 then {
43                     let relaxCost[x] := 0;
44                 }
45             }
46         }
47         let stockLength := stockLengths[i];
48         let tolerance := stockCost[i]/10000;
49         solve Pattern_Gen;
50         if ReducedCost > tolerance + stockCost[i] – StockLimit[i] then {
51             let nPAT[i] := nPAT[i] + 1;
52             let {x in ORDERS} lfep[i,x,nPAT[i]] := Use[x];
53             let {x in ORDERS} rfep[i,x,nPAT[i]] := Relax[x];
54             let isNewPattern := 1;
55         }
56     }
57     if isNewPattern = 0 then {
58         break;
59     }
60     let isNewPattern := 0;
61 };
62 option Cutting_Opt.relax_integrality 0;
63 solve Cutting_Opt;

```

Aby nowy wzór cięcia poprawiał rozwiązanie spełniony musić być warunek w linii 50, który odpowiada równaniu (12). Warto w nim zwrócić uwagę, na dodaną niewielką tolerancję $tolerance = \frac{stockCost[i]}{10000}$, której uwzględnienie ma niwelować skutki niedokładności

numerycznych. Jej brak spowodowałby, że przy wyznaczeniu wzorca już występującego w zbiorze wzorców, algorytm mógłby przyjąć go jako poprawiającego rozwiązanie, tym samym zapętlając się w nieskończoność.

Jeżeli znaleziono choć jeden poprawny wzorzec, wraca się na początek algorytmu. W przeciwnym wypadku przechodzi się do ostatniego rozwiązyania problemu głównego, tym razem z ograniczeniami całkowitoliczbowymi.

3.1.2. Przykład

Tabela 3.1. Przykład zamówienia

pręty bazowe			pręty finalne		
Długość	ilość	koszt	Długość	ilość	max relaksacja
1200	100	1	200	20	0
			200	38	25
			300	35	20
			375	24	0

Załóżmy, że mamy do zaplanowania zamówienie o parametrach przedstawionych w tabeli 3.1. Pierwszym krokiem algorytmu jest inicjalizacja problemów. W tym przypadku oznacza to stworzenie wstępniego zbioru wzorców M i wypełnienie definicji wzorców O . Początkowe wzorce będą trywialne, to jest składające się z jednego rodzaju pręta finalnego. Długość pręta bazowego $\sigma = 1200$, a więc kolejne wzorce będą zawierać $\lfloor \frac{1200}{w_i} \rfloor$ prętów.

$$O = \begin{vmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{vmatrix} \quad (18)$$

Mając zdefiniowany podstawowy zbiór wzorców (18) możemy przystąpić do rozwiązyania problemu głównego w dziedzinie liczb rzeczywistych.

Iteracja pierwsza – problem główny: w wyniku optymalizacji otrzymano wektor P oraz wektor cen (zmiennych dualnych) C (19).

$$P = \begin{vmatrix} 3.333 \\ 6.333 \\ 8.75 \\ 8 \end{vmatrix} \quad C = \begin{vmatrix} 0.167 \\ 0.167 \\ 0.25 \\ 0.333 \end{vmatrix} \quad (19)$$

Iteracja pierwsza – podproblem: następnym krokiem jest rozwiązywanie podproblemu (13) w celu znalezienia nowego wzorca poprawiającego rozwiązanie. W wyniku optyma-

lizacji otrzymano rozwiązanie (20), nowy wzorzec X , wartości relaksacji R oraz wartość optymalizacji K :

$$X = \begin{vmatrix} 0 \\ 5 \\ 1 \\ 0 \end{vmatrix} \quad R = \begin{vmatrix} 0 \\ 100 \\ 0 \\ 0 \end{vmatrix} \quad K = 1.0833 \quad (20)$$

Nowo wyznaczony wzorzec uwzględnia zastosowanie relaksacji. Optymalnym rozwiązaniem jest skrócenie prętów $w_i = 200\text{cm}$ o $r_i/x_i = 20\text{cm}$. Rozwiązanie podproblemu dało wynik K większy od kosztu prętu bazowego $\gamma_1 = 1$, a ograniczenie ilości użytych prętów bazowych nie zostało naruszone więc $\alpha_1 = 0$. Oznacza to, że zastosowanie nowego wzorca poprawia rozwiązanie problemu głównego, dlatego algorytm jest kontynuowany. Wracamy więc do rozwiązywania problemu głównego i wyznaczenia nowych cen już z rozszerzonym o nowy wektor X zestawem wzorców. Przedstawione zostaną teraz pozostałe iteracje algorytmu prowadzące do końcowego rozwiązania.

Iteracja druga – problem główny:

$$O = \begin{vmatrix} 6 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 5 \\ 0 & 0 & 4 & 0 & 1 \\ 0 & 0 & 0 & 3 & 0 \end{vmatrix} \quad P = \begin{vmatrix} 3.333 \\ 0 \\ 6.85 \\ 8 \\ 7.6 \end{vmatrix} \quad C = \begin{vmatrix} 0.167 \\ 0.15 \\ 0.25 \\ 0.333 \end{vmatrix} \quad (21)$$

Iteracja druga – podproblem:

$$X = \begin{vmatrix} 0 \\ 2 \\ 3 \\ 0 \end{vmatrix} \quad R = \begin{vmatrix} 0 \\ 47 \\ 57 \\ 0 \end{vmatrix} \quad K = 1.049999 \quad (22)$$

Iteracja trzecia – problem główny:

$$O = \begin{vmatrix} 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 5 & 2 \\ 0 & 0 & 4 & 0 & 1 & 3 \\ 0 & 0 & 0 & 3 & 0 & 0 \end{vmatrix} \quad P = \begin{vmatrix} 3.333 \\ 0 \\ 0 \\ 8 \\ 3.385 \\ 10.54 \end{vmatrix} \quad C = \begin{vmatrix} 0.167 \\ 0.154 \\ 0.231 \\ 0.333 \end{vmatrix} \quad (23)$$

Iteracja trzecia – podproblem:

$$X = \begin{vmatrix} 0 \\ 3 \\ 1 \\ 1 \end{vmatrix} \quad R = \begin{vmatrix} 0 \\ 75 \\ 0 \\ 0 \end{vmatrix} \quad K = 1.0256 \quad (24)$$

Iteracja czwarta – problem główny:

$$O = \begin{vmatrix} 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 5 & 2 & 3 \\ 0 & 0 & 4 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 3 & 0 & 0 & 1 \end{vmatrix} \quad P = \begin{vmatrix} 3.333 \\ 0 \\ 0 \\ 9.571 \end{vmatrix} \quad C = \begin{vmatrix} 0.167 \\ 0.143 \\ 0.238 \\ 0.333 \end{vmatrix} \quad 5.905 \quad 6.286 \quad (25)$$

Iteracja czwarta – podproblem:

$$X = \begin{vmatrix} 6 \\ 0 \\ 0 \\ 0 \end{vmatrix} \quad R = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} \quad K = 1 \quad (26)$$

Znaleziony nowy wektor X jest identyczny z kolumną pierwszą macierzy O . Nie poprawia on rozwiązania, czwarta iteracja bowiem dała wynik $K = 1$, który to nie jest większy od kosztu pręta bazowego. Z utworzonym w ten sposób zestawem wzorców, przystępujemy do ostatecznego już rozwiązania problemu głównego tym razem jednak w dziedzinie liczb całkowitych. Jego wynikiem jest:

$$p = \begin{vmatrix} 4 \\ 0 \\ 0 \\ 6 \\ 0 \\ 10 \\ 6 \end{vmatrix} \quad (27)$$

Wartymi uwagi są wyniki w iteracji drugiej podproblemu 22. Długość wyznaczonego wzorca w tym kroku jest równa 1196cm, a więc wartości relaksacji nie są optymalne. Związane jest to ze złym doborem stałej λ . Jeżeli jej wartość jest za duża, algorytm może wstrzymywać się od stosowania relaksacji. Jeżeli zaś wartość stałej jest zbyt małą, relak-

sacja jest stosowana zbyt silnie, czego skutkiem są nie w pełni wykorzystane pręty jak w przykładzie.

3.1.3. Relaksacja automatyczna

Pewnym mankamentem modelu zaproponowanego w podrozdziale 3.1.1 jest to, że użytkownik musi być w stanie określić maksymalną relaksację dla każdego pręta finalnego. W praktyce nie zawsze będzie to oczywiste. Dużo przystępniejszy byłby model, który potrafiłby znaleźć interesujące zrelaksowane wzorce, wymagając jedynie minimalnego ukierunkowania przez użytkownika. Ten, jaki i kolejny podrozdział zaproponują rozwiązania, które byłyby bliższe takiemu założeniu.

Wiemy już, że wynikiem rozwiązania problemu plecakowego z relaksacją (13-17), można manipulować zmieniając parametr kosztu relaksacji λ . Teoretycznie można dobrą odpowiednią wartość tego parametru w taki sposób, że zastąpiłaby ograniczenie (15) nie pozwalając skracać długości w nieskończoność. Bez ograniczenia (15) nie byłoby potrzeby ustalania maksymalnej relaksacji. Można też pójść krok dalej. Dobierając taką λ , która nie pozwalałaby tworzyć wzorców trywialnych, to jest takich, które składają się z tego samego typu pręta finalnego. Wzorce nietrywialne niosą ze sobą największą wartość z uwagi na to, że nie da się ich w prosty sposób wyznaczyć.

Załóżmy, że próbujemy rozwiązać problem (13-17) dla zbioru prętów finalnych N składające się z jednego pręta o długości w w dziedzinie liczb rzeczywistych. Pomijając na razie ograniczenie (15), otrzymujemy problem maksymalizacji (28).

$$\max_{x,r} cx - r\lambda \quad \text{dla } x, r \geq 0 \quad (28)$$

$$wx - r = \sigma \quad (29)$$

Ponieważ $x \leq \frac{\sigma+r}{w}$ jest jedną górną granicą zmiennej, którą optymalizujemy, nierówność w drugim ograniczeniu może zostać zamieniona na równość. Po podstawieniu x i przekształceniu otrzymujemy wyrażenie (30).

$$\max_r \left(\frac{c}{w} - \lambda \right) r + \frac{c\sigma}{w} \quad (30)$$

Przybrało ono postać funkcji liniowej. Odczytujemy z niej, że dopóki $\frac{c}{w} - \lambda > 0$ wartość funkcji celu będzie rosła wraz ze wzrostem r . Taka sytuacja nie jest pożądana⁷, ponieważ prowadzi ona do tego, że r zawsze przyjmować będzie wartość maksymalną. Dążymy więc do $\frac{c}{w} - \lambda < 0$. Co ostatecznie daje nam nierówność (31).

$$\frac{c}{w} < \lambda \quad (31)$$

⁷ Ta sytuacja nie jest pożądana w przypadku relaksacji automatycznej. Dla relaksacji manualnej właśnie taka nierówność $\frac{c}{w} > \lambda$ powinna być spełniana, ponieważ docelowo odnaleźć chcemy największą wartość kosztu relaksacji, dla której r może osiągać dowolne wartości.

Dopóki λ będzie spełniać nierówność (31), relaksacja r będzie stosowana oszczędnie (ponieważ jej wzrost pogarszał będzie wynik funkcji celu). Tym samym znajdowane będą wzorce jak najbardziej poprawiające rozwiązanie problemu głównego, ale stosujące ograniczoną relaksację. Wymaga to jednak obliczania oddzielnej wartości λ dla każdego typu pręta finalnego.

Sprowadźmy wyrażenie optymalizacyjne (30) do postaci nierówności świadczącej o tym, czy nowo znaleziony wzorzec poprawia rozwiązanie problemu głównego. W tym celu podstawmy (30) pod lewą stronę nierówności (12). Otrzymamy nierówność (32).

$$\left(\frac{c}{w} - \lambda\right)r + \frac{c\sigma}{w} > \gamma - \alpha \quad (32)$$

Wyznaczmy wzór na λ przenosząc pozostałe zmienne na prawą stronę nierówności:

$$\begin{aligned} \frac{cr}{w} - \lambda r &> \gamma - \alpha - \frac{c\sigma}{w} \\ \lambda &< \frac{\alpha - \gamma + \frac{c(r+\sigma)}{w}}{r} \end{aligned} \quad (33)$$

Otrzymaliśmy ostatecznie nierówność (33). Na jej podstawie możemy wyznaczać parametr λ , dla którego powstaną (lub nie) nowe wzorce. Naszym celem jest uniemożliwienie tworzenia wzorców trywialnych. Zmienna r w nierówności (33) musi więc przyjmować wartość (34), która odpowiada relaksacji wymaganej do wyznaczenia kolejnego trywialnego wzorca zaraz po $\lfloor \frac{\sigma}{w} \rfloor$.

$$r = \lceil \frac{\sigma}{w} \rceil w - \sigma \quad (34)$$

Jeżeli więc parametr λ nie będzie spełniał nierówności (33) w której r przyjmuje wartość z równania (34) to wzorce trywialne nie będą uznawane za poprawiające rozwiązanie problemu głównego.

Model programowanie liniowego (35-40) oraz listing 5 przedstawiają podproblem plecakowy (13-17) dostosowany do aplikowania automatycznej relaksacji. Nowymi ograniczeniami, są (37) oraz (38). Pierwszy z nich zapewnia, że relaksacja nie będzie większa niż długość pręta finalnego. Teoretycznie powinno być to też zapewnione przez odpowiednie dobranie parametru λ_i . Drugą rolą tego graniczenia jest powiązanie zmiennej relaksacji r_i ze zmienną ilości użytych prętów x_i . Gdyby tego ograniczenia brakowało, relaksacja mogłaby być nakładana na pręty, które nawet nie zostały użyte we wzorcu. Równość (38) umożliwia wybieranie prętów, które mają podlegać automatycznej relaksacji. Użytkownik ustawia parametr binarny b_i w danych wejściowych.

$$\max_{x,r} \sum_{i=1}^N c_i x_i - r_i \lambda_i \quad (35)$$

przy ograniczeniach: $\sum_{i=1}^N w_i x_i - r_i \leq \sigma$ (36)

$$\forall i \in N \quad (w_i - 1)x_i \geq r_i \quad (37)$$

$$\forall i \in N \quad r_i = r_i b_i \quad (38)$$

$$\forall i \in N \quad (x_i \geq 0 \wedge r_i \geq 0) \quad (39)$$

$$\forall i \in N \quad x_i, r_i \in \mathbb{Z} \quad (40)$$

Listing 5. Problem plecakowy dla automatycznej relaksacji

```

39 maximize ReducedCost;
40   sum {x in ORDERS} (price[x] * Use[x] – Relax[x] * relaxCost[x]);
41 subj to LengthLimit:
42   sum {x in ORDERS} (orderLengths[x] * Use[x] – Relax[x]) <= stockLength;
43 subj to RelaxLimit {x in ORDERS}: #bar length cannot be shorter than 1
44   (orderLengths[x]-1) * Use[x] >= Relax[x];
45 subj to IsRelaxationAllowed {x in ORDERS}:
46   Relax[x] = Relax[x] * canBeRelaxed[x];

```

Algorytm stosujący automatyczną relaksację jest niemalże identyczny z tym przedstawionym w listingach 3 i 4. Jedyną różnicą jest obliczanie wartości kosztu relaksacji λ_i przed rozwiązaniem podproblemu. Wyznaczany jest on na podstawie nierówności (31) i (33). Listing 6 przedstawia tę procedure w kodzie AMPL.

Listing 6. Wyznaczanie kosztu relaksacji

```

51 for {x in ORDERS} {
52   let tempRelax := ceil(stockLengths[i]/orderLengths[x]) * orderLengths[x] – stockLengths[i];
53   let minRelaxCost := price[x] / orderLengths[x];
54   if tempRelax = 0 or allowBasicPatterns = 1 then {
55     let relaxCost[x] := minRelaxCost + 0.00001;
56   }
57   else {
58     let relaxCost[x] := ((price[x] * (tempRelax + stockLengths[i])) / orderLengths[x]
59     – stockCost[i] + StockLimit[i]) / tempRelax + 0.00001;
60     if relaxCost[x] <= minRelaxCost then {
61       let relaxCost[x] := minRelaxCost + 0.00001;
62     }
63   }
64   let relaxCost[x] := relaxCost[x] * relaxCostMultiplier;
65 };

```

Wartym uwagi w listingu 6 są instrukcje warunkowe. W danych wejściowych zmieniać

można wartość parametru binarnego *allowBasicPatterns*, który to decyduje czy zezwalamy na powstawanie wzorców trywialnych. Wartość kosztu relaksacji obliczona na podstawie nierówności (31) jest uznawana za minimalną. Nie dopuszczamy więc do sytuacji przypisania wartości z nierówności (33), jeżeli byłaby mniejsza od tej z (31).

3.1.4. Relaksacja krokowa

Ostatnim sposobem nakładania relaksacji zaproponowanym w tej pracy jest stopniowe wyznaczanie kolejnych poprawiających rozwiązywanie wzorców, które podlegają najmniej szemu możliwemu skróceniu. Z punktu widzenia użytkownika wyglądałoby to tak, że w pierwszej kolejności przedstawiane byłyby mu wzorce wyznaczone bez relaksacji, po czym miałby on możliwość generowania sukcesywnie wzorców z coraz to większą relaksacją. Proces dochodziłby końca w momencie, gdy wyznaczony zbiór cięć byłby satysfakcyjny.

Tak jak w przypadku poprzednich modeli, przystosowanie do tego typu relaksacji wymaga zmian w modelu podproblemu plecakowego. Funkcja celu w tym przypadku staje się ograniczeniem (42). W tej formie algorytmu koszt relaksacji nie występuje, dlatego cała nierówność przyjmuje postać warunku na poprawę rozwiązywania problemu głównego (12). Nową funkcją celu jest minimalizacja sumy relaksacji (41). Pozostałe ograniczenia przyjmują identyczną postać jak w przypadku modelu relaksacji automatycznej.

$$\min_r \sum_{i=1}^N r_i \quad (41)$$

$$\text{przy ograniczeniach: } \sum_{i=1}^N c_i x_i > \gamma - \alpha \quad (42)$$

$$\sum_{i=1}^N w_i x_i - r_i \leq \sigma \quad (43)$$

$$\forall i \in N \quad (w_i - 1)x_i \geq r_i \quad (44)$$

$$\forall i \in N \quad r_i = r_i b_i \quad (45)$$

$$\forall i \in N \quad (x_i \geq 0 \wedge r_i \geq 0) \quad (46)$$

$$\forall i \in N \quad x_i, r_i \in \mathbb{Z} \quad (47)$$

Kod algorytmu AMPL wykorzystujący metodę relaksacji krokowej jest najprostszy, ponieważ zawsze zakłada jednokrotne rozwiązywanie podproblemu i problemu głównego. Dzieje się tak, ponieważ po każdym przebiegu tego algorytmu, użytkownik musi otrzymać wynik i zdecydować czy chce generować wzorce dalej. Konsekwencją tego jest konieczność przekazywania pomiędzy uruchomieniami algorytmu zestawu wygenerowanych wzorców. Plik z danymi wejściowymi dla relaksacji krokowej jest najbardziej rozbudowany. Oprócz zbioru prętów bazowych i finalnych zawiera:

- indeks pręta bazowego, którego wzorzec cięcia będzie szukany

3. Relaksacja długości prętów finalnych w problemie cięcia materiału

- definicje wyznaczonych już wzorców
- relaksacje wyznaczonych już wzorców
- liczbę wzorców
- ceny prętów finalnych
- wartości zmiennej dualnej α_k (10)

Listing 7. Problem plecakowy dla automatycznej relaksacji

```
39 minimize RelaxSum:  
40     sum {i in ORDERS} Relax[i];  
41 subj to PatternCost:  
42     sum {i in ORDERS} price[i] * Use[i] >= minNewPatternCost;  
43 subj to LengthLimitR:  
44     sum {i in ORDERS} (orderLengths[i] * Use[i] – Relax[i]) <= stockLength;  
45 subj to RelaxLimit {i in ORDERS}:  
46     (orderLengths[i]–1) * Use[i] >= Relax[i];  
47 subj to IsRelaxationAllowed {i in ORDERS}:  
48     Relax[i] = Relax[i] * canBeRelaxed[i];
```

Listing 8. Algorytm relaksacji krokowej

```
20 param isFeasible;  
21 param i := stockItemToRelax;  
22 let isFeasible := 1;  
23 let minNewPatternCost := 0.0001 + stockCost[i] – prevStockLimit[i];  
24 let stockLength := stockLengths[i];  
25 solve Relaxed_Pattern_Gen;  
26  
27 if solve_result == "infeasible" or solve_result_num == 200 then  
28 {  
29     let isFeasible := 0;  
30 }  
31 else  
32 {  
33     let nPAT[i] := nPAT[i] + 1;  
34     let {x in ORDERS} lfep[i,x,nPAT[i]] := Use[x];  
35     let {x in ORDERS} rfep[i,x,nPAT[i]] := Relax[x];  
36 }  
37  
38 solve Cutting_Opt;  
39 let {x in ORDERS} price[x] := FillOrder[x].dual;
```

Zastosowanie algorytmu 8 musi być poprzedzone rozwiązaniem problemu cięcia materiału bez relaksacji. Można do tego wykorzystać model relaksacji manualnej (13-17) z zerowymi wartościami relaksacji lub użyć dedykowanego modelu. W pracy zdecydowano się na przygotowanie oddzielnego modelu, lecz nie będzie tu przedstawiony z uwagi na jego trywialność.

3.2. Minimalizacja odpadów

Do tej pory wszystkich omawianych w pracy algorytmów głównym celem była minimalizacja kosztu prętów bazowych (lub ich liczby, jeżeli jest tylko jeden typ pręta bazowego). Można jednak do problemu cięcia podejść od innej strony i za cel obrać minimalizację odpadów. Jako odpad rozumiemy różnicę długości pręta bazowego i sumy długości prętów finalnych pozyskanych z niego podczas cięcia. Mając do dyspozycji pręt bazowy o długości $\sigma = 1100$ i chcąc uzyskać z niego jedynie pręty finalne $w = 500$, stosując optymalny podział, otrzymamy na każdym przecie bazowym odpad $\beta = 100$.

Stosując takie podejście, trzeba być jednak przygotowanym na to, że rozwiązanie optymalne będzie generowało nadmiar niektórych prętów finalnych. W praktyce więc, użytkownicy decydujący się na użycie tego algorytmu powinni być w stanie magazynować niewykorzystane pręty, aby czerpać z niego największe korzyści.

Modele minimalizujące odpady różnią się od swoich odpowiedników minimalizujących koszt jedynie funkcjami celu problemu głównego i podproblemu (poza relaksacją krokową gdzie klasyczna funkcja celu przechodzi w ograniczenie). Ograniczenia (9-10) pozostają identyczne. Funkcja celu problemu głównego z minimalizacją odpadów przedstawiona jest w (48). Suma $\sum_{i=1}^N (o_{kij} w_i - q_{kij})$ jest równa długości wszystkich prętów finalnych użytych we wzorcu j pręta bazowego k , gdzie q_{kij} odpowiada wartości relaksacji. Odejmując wartość sumy od σ_k uzyskujemy odpad, który później jest mnożony przez liczbę użyć wzorca p_{kj} . Analogicznie jak w przypadku minimalizacji kosztów, ten sam problem główny będzie używany w podrozdziałach 3.2.1, 3.2.2 i 3.2.3.

$$\min_p \sum_{k=1}^S \sum_{j=1}^M (\sigma_k - \sum_{i=1}^N (o_{kij} w_i - q_{kij})) p_{kj} \quad (48)$$

Listing 9 przedstawia problem główny minimalizujący odpady sformułowany w języku AMPL. Wartymi uwagi są tu niezmienione względem listingu 1 ograniczenia *FillOrder* i *StockLimit*.

Listing 9. Problem główny minimalizujący odpady

```

22 minimize Waste:
23   sum {i in STOCK, j in PATTERNS[i]} (stockLengths[i] -
24     (sum{x in ORDERS} (lfep[i,x,j] * orderLengths[x] - rfep[i,x,j])) * usedPatterns[i,j];
25 subj to FillOrder {x in ORDERS}:
26   sum {i in STOCK, j in PATTERNS[i]} lfep[i,x,j] * usedPatterns[i,j] >= orderNum[x];
27 subj to StockLimit {i in STOCK}:
28   sum{j in PATTERNS[i]} usedPatterns[i,j] <= stockNum[i];

```

3.2.1. Relaksacja manualna

Funkcja celu podproblemu z relaksacją manualną przedstawiona jest w modelu (49) oraz listingu 10. Koszt prętów finalnych c_i jest wyznaczany w ten sam sposób, czyli przypisywana jest mu wartość zmiennej dualnej odpowiadającej ograniczeniu (9). Innymi słowy,

3. Relaksacja długości prętów finalnych w problemie cięcia materiału

koszt c_i jest równy przyrostowi odpadów, po zwiększeniu zamówienia na pręt finalny długości w_i o 1. Dlatego też, do długości pręta finalnego użytego we wzorcu dodawany jest odpad, który on potencjalnie generuje.

$$\min_{x,r} \sigma - \sum_{i=1}^N (w_i x_i - r_i + c_i x_i - r_i \lambda_i) \quad (49)$$

Listing 10. Model podproblemu minimalizującego odpady z relaksacją manualną

```
38 minimize ReducedCost:  
39   stockLength - sum{x in ORDERS}(orderLengths[x] * Use[x] - Relax[x] +  
40     price[x]*Use[x] - Relax[x]*relaxCost[x]);  
41 subj to LengthLimit:  
42   sum {x in ORDERS} (orderLengths[x] * Use[x] - Relax[x]) <= stockLength;  
43 subj to MaxRelax {x in ORDERS}:  
44   Relax[x] <= maxRelax[x] * Use[x];
```

Ostatnią różnicą pomiędzy minimalizacją odpadów a minimalizacją kosztów jest warunek poprawy rozwiązania problemu głównego pojawiający się w kodzie AMPL w linii 49. Aby wzorzec wyznaczony przez podproblem poprawiał wynik problemu głównego, funkcja celu (49) musi osiągnąć wartość mniejszą niż α_k .

Listing 11. Warunek poprawy rozwiązania problemu głównego

```
48 solve Pattern_Gen;  
49 if ReducedCost < StockLimit[i] - 0.1 then
```

3.2.2. Relaksacja automatyczna

Model AMPL relaksacji automatycznej z minimalizacją odpadów przedstawiony w listingu 12 od swojego odpowiednika z podrozdziału 3.1.3 różni się jedynie funkcją celu, która przybiera formę (49).

Listing 12. Model podproblemu minimalizującego odpady z relaksacją automatyczną

```
34 minimize ReducedCost:  
35   stockLength - sum{x in ORDERS}(orderLengths[x] * Use[x] - Relax[x] +  
36     price[x]*Use[x] - Relax[x]*relaxCost[x]);  
37 subj to LengthLimit:  
38   sum {x in ORDERS} (orderLengths[x] * Use[x] - Relax[x]) <= stockLength;  
39 subj to RelaxLimit {x in ORDERS}:  
40   (orderLengths[x]-1) * Use[x] >= Relax[x];  
41 subj to IsRelaxationAllowed {x in ORDERS}:  
42   Relax[x] = Relax[x] * canBeRelaxed[x];
```

Ważnym pytaniem jest jednak to, czy nierówności (31) i (33) wciąż pozwalają wyznaczyć odpowiednie koszta relaksacji. Stosując takie same podstawienia (29), lecz dla nowej

funkcji celu dostajemy:

$$\min_{x,r} \sigma - (wx - r + cx - r\lambda) \quad (50)$$

$$\min_r \sigma - (\sigma + r - r + c\frac{\sigma + r}{w} - r\lambda) \quad (51)$$

$$\min_r (\lambda - \frac{c}{w})r - \frac{c\sigma}{w} \quad (52)$$

Ponieważ (52) jest wyrażeniem minimalizacji w odróżnieniu od (30), a współczynnik kierunkowy funkcji liniowej jest odwrócony, koszt relaksacji powinien przybierać tę samą postać co w nierówności (31).

Przeprowadźmy jeszcze raz wyznaczenie nierówności blokującej tworzenie wzorców trywialnych:

$$\begin{aligned} (\lambda - \frac{c}{w})r - \frac{c\sigma}{w} &< \alpha \\ \lambda r - \frac{cr}{w} &< \alpha + \frac{c\sigma}{w} \\ \lambda &< \frac{\alpha + \frac{c(r+\sigma)}{w}}{r} \end{aligned} \quad (53)$$

Nierówność (53) nieznacznie różni się od swojego poprzednika (33). Pozbawiona jest ona parametru γ , który odpowiadał wcześniej za koszt prętów bazowych.

3.2.3. Relaksacja krokowa

Kroki, jakie trzeba podjąć w celu dostosowania relaksacji krokowej do minimalizacji odpadów, są takie same jak w przypadku poprzednich dwóch podrozdziałów. Wymagane jest dostosowanie funkcji celu i warunku poprawy rozwiązania problemu głównego. W przypadku relaksacji krokowej obydwie te zmiany dotyczą ograniczenia *PatternCost*.

Listing 13. Model podproblemu minimalizującego odpady z relaksacją automatyczną

```

34 minimize RelaxSum;
35   sum {x in ORDERS} Relax[x];
36   subj to PatternCost:
37     stockLength - sum{x in ORDERS}(orderLengths[x] * Use[x] - Relax[x] +
38     price[x]*Use[x]) <= minNewPatternCost;
39   subj to LengthLimit:
40     sum {x in ORDERS} (orderLengths[x] * Use[x] - Relax[x]) <= stockLength;
41   subj to RelaxLimit {x in ORDERS}:
42     (orderLengths[x]-1) * Use[x] >= Relax[x];
43   subj to IsRelaxationAllowed {x in ORDERS}:
44     Relax[x] = Relax[x] * canBeRelaxed[x];

```

3.2.4. Model dodatkowy

W pracy przygotowano do testów jeszcze jeden model optymalizacyjny (54-61). Potrafi on wyznaczać nowe optymalne wzorce w podproblemie bez potrzeby iteracji po prętach bazowych. Jest on odmianą modelu relaksacji manualnej minimalizującej odpady, lecz podobną wersję można było przygotować na podobieństwo innych omawianych modeli. Pojawia się w nim nowa zmienna η_k oraz ograniczenie (56), których celem jest zapewnienie, że tylko jeden pręt bazowy zostanie wybrany jako podstawa wzorca. Dodatkowo w funkcji celu uwzględnia się warunek na poprawienie rozwiązań problemu głównego. W przypadku relaksacji manualnej minimalizującej odpady jest to jedynie parametr α_k .

W testach przeprowadzonych w dalszej części pracy porównana zostanie szybkość modelu z dodatkową zmienną η (54-61) i modelu z iteracją po prętach bazowych (49).

$$\min_{x,r,\eta} \sum_{k=1}^S (\sigma_k - \alpha_k) \eta_k - \sum_{i=1}^N (w_i x_i - r_i + c_i x_i - r_i \lambda) \quad (54)$$

$$\text{przy ograniczeniach: } \sum_{i=1}^N w_i x_i - r_i \leq \sum_{k=1}^S \sigma_k \eta_k \quad (55)$$

$$\sum_{k=1}^S \eta_k = 1 \quad (56)$$

$$\forall i \in N \quad r_i \leq m_i x_i \quad (57)$$

$$\forall i \in N \quad (x_i \geq 0 \wedge r_i \geq 0) \quad (58)$$

$$\forall k \in S \quad \eta_k \geq 0 \quad (59)$$

$$\forall i \in N \quad x_i, r_i \in \mathbb{Z} \quad (60)$$

$$\forall k \in S \quad \eta_k \in \mathbb{Z} \quad (61)$$

4. System wspierający cięcie

W tym rozdziale przedstawiona zostanie autorska aplikacja internetowa, przygotowana w ramach niniejszej pracy magisterskiej. Została ona zaprojektowana jako narzędzie ułatwiające użytkownikom korzystanie z wcześniej omówionych algorytmów. Wykonanie aplikacji realizuje pośrednio kluczowe cele pracy poprzez:

- **Implementację algorytmów cięcia** – logika biznesowa aplikacji (backend⁸) w głównej mierze skupia się na wykorzystaniu opracowanych w rozdziale 3 modeli.
- **Wsparcie testów badawczych** – ważną częścią aplikacji są testy integracyjne, które automatyzują proces weryfikacji oraz porównywania wyników generowanych przez modele programowania liniowego.
- **Praktyczne zastosowanie** – warstwa aplikacji odpowiedzialna za interakcję z użytkownikiem (frontend⁹) pozwala w intuicyjny sposób definiować własne problemy cięcia.

W dalszej części rozdziału zdefiniowano założenia funkcjonalne aplikacji, wymieniono wykorzystane technologie oraz opisano jej strukturę. Przedstawiono również przykłady ilustrujące jej użycie przez potencjalnego użytkownika.

4.1. Założenia funkcjonalne i niefunkcjonalne systemu

W celu prawidłowego zaprojektowania architektury systemu konieczne jest szczegółowe określenie jego wymagań funkcjonalnych. W przypadku aplikacji wspierającej cięcie są to:

1. **Definiowanie własnych problemów cięcia** – użytkownik aplikacji ma możliwość wypełniania zbioru prętów bazowych, jak i zbioru prętów finalnych. Poszczególne atrybuty przedmiotów powinny być edytowalne tylko, jeśli aktualnie wybrany algorytm z nich korzysta. Wprowadzone przez użytkownika dane powinny być weryfikowane, a użytkownik informowany, gdy są one nieprawidłowe.
2. **Wybór celu optymalizacji** – Interfejs graficzny powinien umożliwiać wybór między minimalizacją kosztu a minimalizacją odpadu.
3. **Wybór stosowanej relaksacji** – Interfejs graficzny powinien pozwalać na wybór pomiędzy relaksacją manualną, automatyczną i krokową.
4. **Rozwiązywanie problemu cięcia** – Aplikacja powinna rozwiązywać problemy cięcia wprowadzone przez użytkownika, za pomocą metod opisanych w rozdziale 3.
5. **Prezentacja wyników** – Po rozwiązaniu problemu, użytkownik może obejrzeć zwrócone przez algorytm wzorce cięcia. Prezentowane winny być również ogólne statystki,

⁸ backend – warstwa serwerowa aplikacji odpowiedzialna za logikę biznesową i przetwarzanie danych.

⁹ frontend – warstwa interfejsu użytkownika, odpowiedzialna za pobieranie i wyświetlanie danych oraz przekazywanie ich do backendu.

4. System wspierający cięcie

takie jak łączny koszt użytych pretów bazowych, suma odpadów czy całkowita relaksacja.

6. **Import i eksport problemów** – Użytkownik powinien być w stanie zapisywać utworzone przez niego problemy cięcia, jak i mówić je wczytywać do systemu.
7. **Eksport rozwiązań** – Rozwiązanie w postaci wzorców powinno być eksportowalne np. w formacie CSV.
8. **Walidacja rozwiązań** – System powinien reagować informacją zwrotną w przypadku gdy problem zadany przez użytkownika jest nierozwiązywalny.

Sformułować można też pewne wymagania niefunkcjonalne systemu, które spełniać powinna aplikacja przeznaczona dla użytkowników Internetu. Będą to:

1. **Wydajność** – Interfejs użytkownika powinien cechować się wysoką responsywnością.
2. **Skalowalność** – Metody wywoływane po stronie serwera powinny być zrównoleglone, co pozwoli na łatwe skalowanie przepustowości przy wzroście liczby użytkowników.
3. **Kompatybilność** – Korzystanie z systemu powinno być możliwe przy użyciu wszystkich popularnych przeglądarek internetowych, jak i urządzeń mobilnych.
4. **Bezpieczeństwo** – Dostęp do zasobów serwera powinien być ograniczony dla użytkowników zewnętrznych aby zapewnić ochronę przed atakami. Komunikacja z serwerem powinna odbywać się po protokole HTTPS.

4.2. Wykorzystane technologie

Aby zrealizować postawione wymagania, konieczne było zastosowanie odpowiednich technologii, które zapewniłyby wysoką wydajność, skalowalność i elastyczność aplikacji.

4.2.1. Frontend

Do stworzenia warstwy interfejsu użytkownika użyto frameworku¹⁰ **React**. Cechuje się on dobrą responsywnością[16] i często używany jest do budowania aplikacji jednostronnicowych¹¹. Taką też formę przyjmie projektowany system, gdyż zapewnia ona większą płynność i efektywniejsze wykorzystanie zasobów[17].

Jako język programowania frontendu wybrano **TypeScript**. Pozwala on zwiększyć czytelność kodu i jego bezpieczeństwo, poprzez wczesne wyłapywanie błędów programisty.

Do tworzenia zaawansowanych formularzy i tabel wykorzystano bibliotekę komponentów **Mantine** oraz **Mantine React Table**. Skróciły one znacznie czas tworzenia aplikacji i sprawiły, że stała się bardziej interaktywna i przyjazna użytkownikowi. Biblioteka **zustand** została użyta do zarządzania stanem aplikacji.

Do budowy i uruchamiania frontendu wykorzystano **Vite.js**. Narzędzie to umożliwia automatyczne odwzorowywanie zmian w kodzie, co przyspiesza cykl rozwoju. Jako śro-

¹⁰ framework – szkielet do budowy aplikacji. Zestaw bibliotek i komponentów, które mogą być przez programistę rozbudowywane i dostosowywane w celu zbudowania aplikacji.

¹¹ jednostronicowa aplikacja internetowa (ang. *Single Page Application*) posiada tylko jeden plik html i nie wymaga przeładowywania strony w trakcie użytkowania

dowisko uruchomieniowe frontendu wybrano **Node.js**. Zarządzanie zależnościami oraz automatyzację procesu budowy umożliwia **npm** (*Node Package Manager*).

4.2.2. Backend

Warstwa serwerowa aplikacji została oparta na technologii **ASP.NET** z użyciem języka **C#**. Wykorzystanie tego środowiska znacznie ułatwia implementację wielu kluczowych funkcji w aplikacjach backendowych, takich jak:

- **Kontrolery zapytań HTTP** – dzięki ASP.NET łatwo jest zdefiniować kontrolery, które obsługują zapytania HTTP (GET, POST, PUT, DELETE), zapewniają podstawową walidację typów i zwracają odpowiedzi w odpowiednim formacie.
- **Serializacja i deserializacja** – framework oferuje wbudowaną obsługę serializacji i deserializacji danych w popularnych formatach, takich jak JSON, co umożliwia łatwą wymianę danych między klientem a serwerem.
- **Warstwa pośrednia** – ASP.NET udostępnia gotowe elementy warstwy pośredniej takie jak autoryzacja, logowanie czy też obsługa błędów.
- **Zrównoleglenie przetwarzania danych** – kontrolery ASP.NET domyślnie uruchamiają przetwarzanie zapytań w oddzielnych wątkach. Technika wstrzykiwania zależności ułatwia zarządzanie cyklem życia obiektów, co też pośrednio pomaga w stosowaniu programowania równoległego.
- **Testy** – ASP.NET wspiera tworzenie testów jednostkowych i integracyjnych. W projektowanym systemie do pisania testów użyto biblioteki **xUnit**.

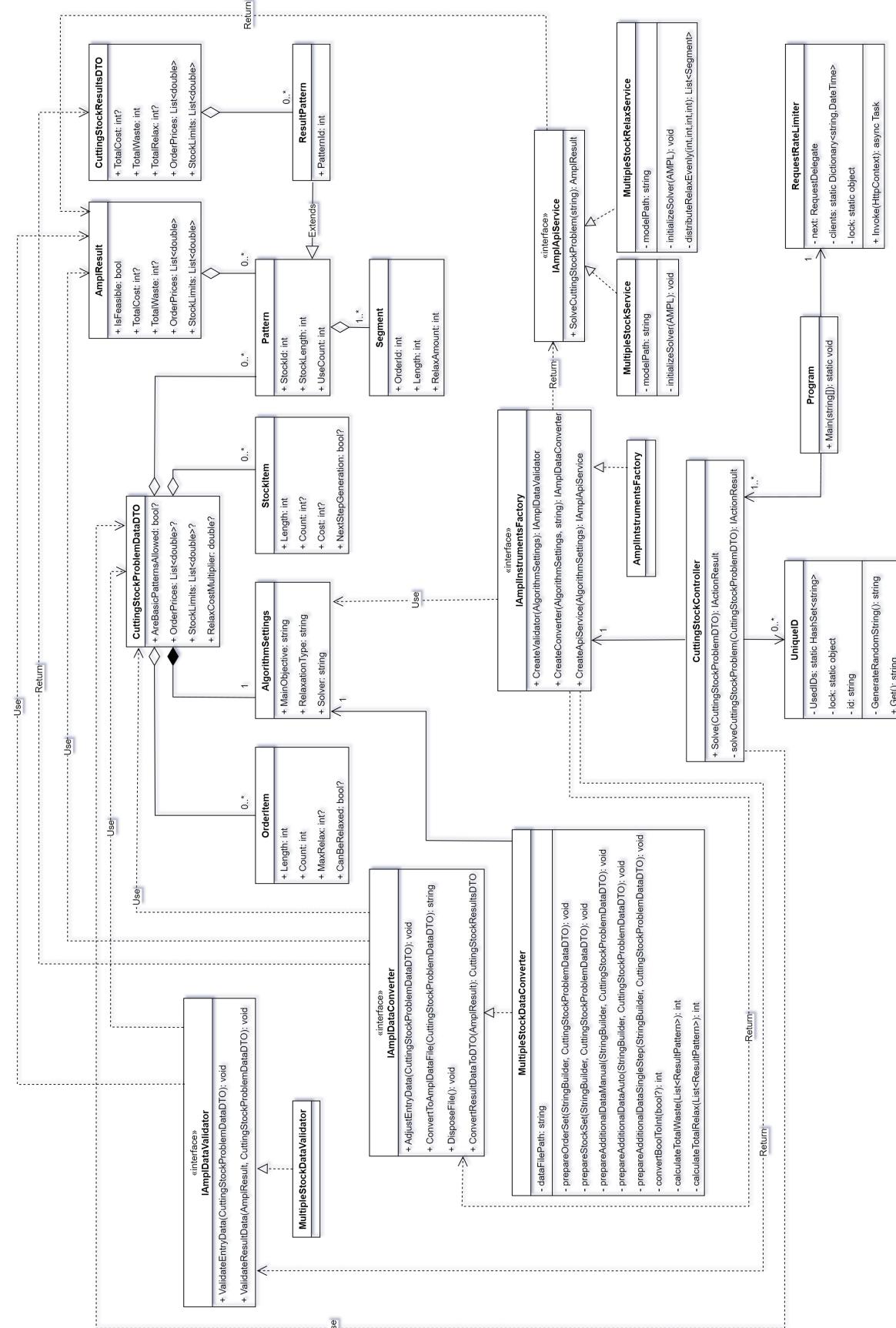
Do komunikacji z **AMPL** wykorzystano **AMPL API**, które obsługuje też język **C#**. Za jego pomocą serwer może wołać metody **AMPL** i odczytywać jego wyniki i wartości zmiennych.

4.3. Struktura klas

Kod aplikacji pisany był zgodnie z paradygmatem programowania obiektowego. Wyóżnić więc można kilka najważniejszych klas i komponentów w systemie. Po stronie serwera będą to:

- `CuttingStockController` – kontroler odpowiedzialny za przyjmowanie danych wejściowych problemu cięcia, rozwiązywanie go i zwracanie wyniku.
- `MultipleStockDataConverter` – klasa przygotowująca dane dostarczone z frontendu do formatu obsługiwanej przez **AMPL** i przciwnie.
- `MultipleStockDataValidator` – validator danych wejściowych, jak i wyjściowych.
- `MultipleStockRelaxService` – klasa wywołująca metody **AMPL** i odczytująca zwracane przez niego zmienne i parametry.
- `RequestRateLimiter` – element warstwy pośredniej uniemożliwiający hostom nadmierne wysyłanie zapytań do serwera.

4. System wspierający cięcie



Rysunek 4.1. Diagram klas warstwy serwerowej

Najważniejsze komponenty po stronie warstwy użytkownika to:

- `LayoutWithHeaderSidebar` – Dzieli stronę na pasek nagłówka i część główną.
- `MainContent` – Część główna aplikacji. Składa się z panelu ustawień, tabel do wprowadzania danych i tabel wyników.
- `SettingsPanel` – panel ustawień. Służy do wyboru algorytmu. Wyświetla też jego opis.
- `DynamicTable` – komponent renderujący tabelę z możliwością dynamicznego dodawania i usuwania wierszy.
- `StockTable` oraz `OrderTable` – tabele umożliwiające edycję prętów bazowych i prętów finalnych.
- `PatternTable` oraz `OutputVariablesTable` – tabele prezentujące wyniki.

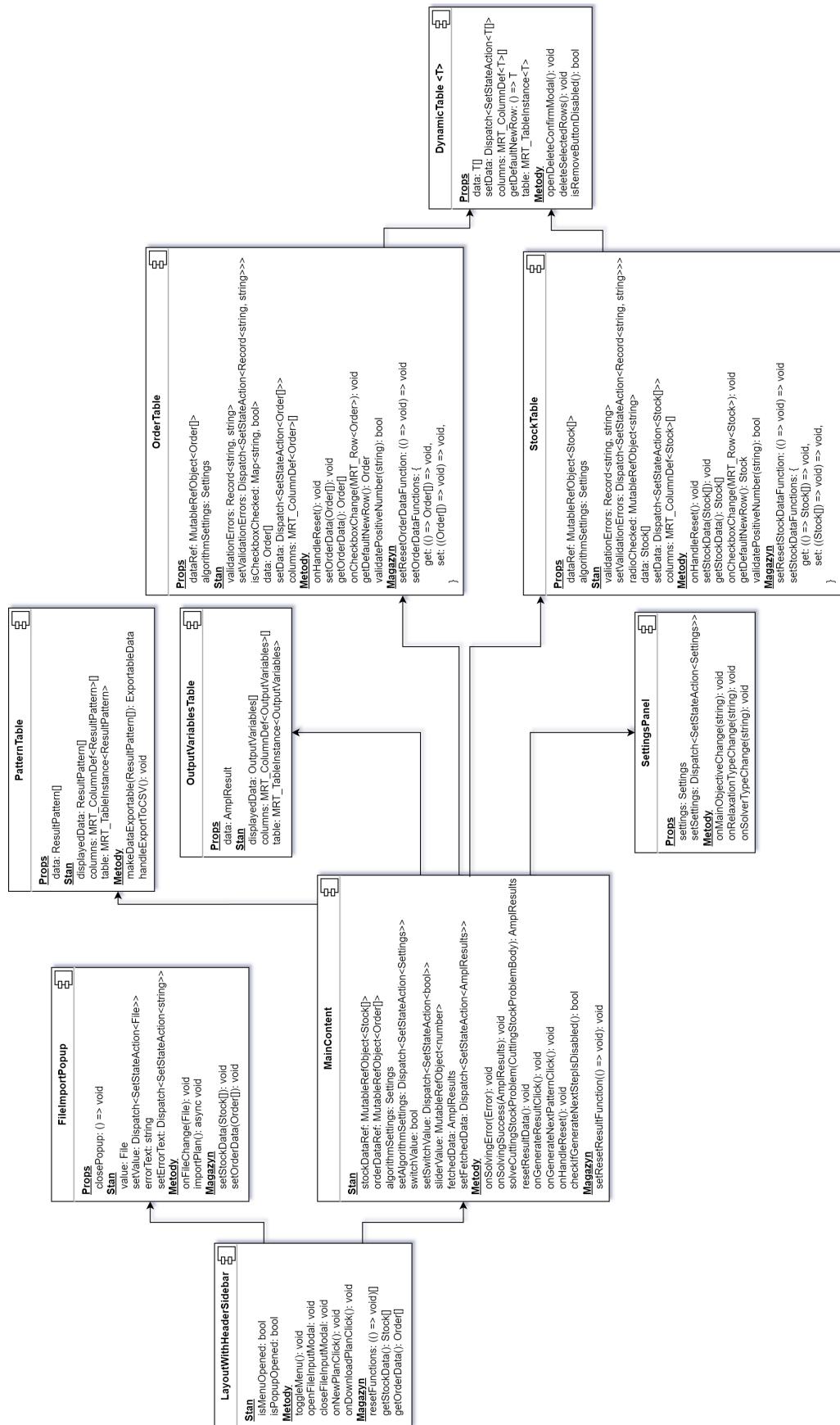
Powstało też kilka hooków¹² odpowiedzialnych za zarządzanie magazynem stanu¹³ i wysyłanie zapytań HTTP do backendu. Są nimi:

- `useStockStore` – Przechowuje zbiór ustawionych prętów bazowych.
- `useOrderStore` – Przechowuje zbiór ustawionych prętów finalnych.
- `useResetStore` – Przechowuje funkcje resetujące stan okna.
- `useSolveCuttingStockProblem` – Przesyła definicję problemu cięcia do serwera. Zwraca rozwiązanie lub informuje o zaistniałym błędzie.

¹² hook – technika pozwalająca ingerować w działanie aplikacji poprzez przechwytywanie wywołań, funkcji i komunikatów. W React hooki pozwalają zarządzać stanem komponentów funkcyjnych.

¹³ magazyn w React przechowuje stan globalny aplikacji. W pracy do zarządzania magazynem wykorzystano bibliotekę `zustand`.

4. System wspierający cięcie



Rysunek 4.2. Diagram komponentów warstwy interfejsu użytkownika

4.4. Przykłady użycia

Na poniższych zrzutach ekranu 4.3, 4.4 i 4.5 przedstawiono kolejne kroki, jakie podejmuje użytkownik korzystający z systemu w celu rozwiązania problemu cięcia.

The screenshot shows the Steel Cutting System interface with several components:

- Top Bar:** Includes "New Plan" (blue icon), "Import Plan" (red box, step 1), "Download Plan" (grey icon), and a search bar.
- Algorithm Settings (Left Column):**
 - Minimize:** Cost (radio button selected).
 - Relaxation type:** Manual (radio button selected).
 - Solver:** Cbc (radio button selected).
- Order Table (Top Right):** Shows a table header with columns: Length \downarrow , Count \uparrow , and Total Relax \uparrow . Below it says "No records to display". Buttons include "Add new" and sorting arrows.
- Stock Table (Middle Right):** Shows a table header with columns: Length \uparrow , Count \uparrow , and Cost \downarrow . Below it says "No records to display". Buttons include "Add new" and sorting arrows.
- Result Generation (Bottom Center):** A blue button labeled "Generate Result" is visible. Below it, a section titled "Relax Cost Multiplier" has a dropdown menu with "Cbc" selected.
- Pattern Table (Bottom Right):** Shows a table header with columns: Pattern \uparrow , Stock Length \uparrow , and Count \uparrow . Below it says "No records to display". Buttons include "Add new" and sorting arrows.

Rysunek 4.3. Wybór algorytmu i wprowadzenie danych

4. System wspierający cięcie

The screenshot shows the Steel Cutting System interface with the following components:

- Algorithm Settings:**
 - Minimize: Cost (radio button selected)
 - Vaste (radio button)
 - Relaxation type: Manual (radio button selected)
 - Auto
 - Single Step
 - Solver: Cbc (radio button selected)
 - HIGHS
 - CPLEX
- Stock Table:**

	Length ↑ :	Count ↑ :	Cost ↑ :
1200	100	1	

Add new 10 < 1-1 of 1 >
- Order Table:**

	Length ↑ :	Count ↑ :	Max Relax ↑ :
	200	20	Max Relax
	200	38	25
	300	35	20
	375	24	Max Relax

Add new 10 < 1-4 of 4 >
- Algorithm Description:**

This model assumes that the maximum relaxation value will be provided in the input data. The solution found will include relaxed lengths only if they improve the final result. Relaxation values will be applied sparingly to avoid generating additional waste.
- Results:**

Cost ↑ :	Waste ↑ :	Total Relax ↑ :
---	---	---

No records to display

Rysunek 4.4. Przesłanie problemu do backendu

Algorithm Settings

Minimize: Cost Waste

Relaxation type: Manual Auto Single Step

Solver: Cbc HiGHS CPLEX

Algorithm Description

This model assumes that the maximum relaxation value will be provided in the input data. The solution found will include relaxed lengths only if they improve the final result. Relaxation values will be applied sparingly to avoid generating additional waste.

Stock Table	Order Table																																								
<table border="1"> <thead> <tr> <th></th> <th>Length ↑↓</th> <th>Count ↑↓</th> <th>Cost ↑↓</th> </tr> </thead> <tbody> <tr> <td>1200</td> <td>100</td> <td>1</td> <td></td> </tr> <tr> <td>Add new</td> <td>10</td> <td>< ></td> <td>1-1 of 1</td> </tr> <tr> <td>Generate Result</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Relax Cost Multiplier</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		Length ↑↓	Count ↑↓	Cost ↑↓	1200	100	1		Add new	10	< >	1-1 of 1	Generate Result				Relax Cost Multiplier				<table border="1"> <thead> <tr> <th></th> <th>Length ↑↓</th> <th>Count ↑↓</th> <th>Max Relax ↑↓</th> </tr> </thead> <tbody> <tr> <td>200</td> <td>200</td> <td>20</td> <td>Max Relax</td> </tr> <tr> <td>300</td> <td>200</td> <td>38</td> <td>25</td> </tr> <tr> <td>375</td> <td>200</td> <td>35</td> <td>20</td> </tr> <tr> <td>Add new</td> <td>10</td> <td>< ></td> <td>1-4 of 4</td> </tr> </tbody> </table>		Length ↑↓	Count ↑↓	Max Relax ↑↓	200	200	20	Max Relax	300	200	38	25	375	200	35	20	Add new	10	< >	1-4 of 4
	Length ↑↓	Count ↑↓	Cost ↑↓																																						
1200	100	1																																							
Add new	10	< >	1-1 of 1																																						
Generate Result																																									
Relax Cost Multiplier																																									
	Length ↑↓	Count ↑↓	Max Relax ↑↓																																						
200	200	20	Max Relax																																						
300	200	38	25																																						
375	200	35	20																																						
Add new	10	< >	1-4 of 4																																						
<table border="1"> <thead> <tr> <th>Stock</th> <th>Waste ↑↓</th> <th>Total Relax ↑↓</th> </tr> </thead> <tbody> <tr> <td>26</td> <td>450</td> <td>1450</td> </tr> </tbody> </table>	Stock	Waste ↑↓	Total Relax ↑↓	26	450	1450	<table border="1"> <thead> <tr> <th>Pattern</th> <th>Stock Length ↑↓</th> <th>Count ↑↓</th> <th></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1200</td> <td>4</td> <td>200</td> </tr> <tr> <td>2</td> <td>1200</td> <td>6</td> <td>182 (+18) 181 (+19) 280 (+20) 375</td> </tr> <tr> <td>3</td> <td>1200</td> <td>10</td> <td>180 (+20) 280 (+20) 280 (+20)</td> </tr> <tr> <td>4</td> <td>1200</td> <td>6</td> <td>375 375 375</td> </tr> </tbody> </table>	Pattern	Stock Length ↑↓	Count ↑↓		1	1200	4	200	2	1200	6	182 (+18) 181 (+19) 280 (+20) 375	3	1200	10	180 (+20) 280 (+20) 280 (+20)	4	1200	6	375 375 375														
Stock	Waste ↑↓	Total Relax ↑↓																																							
26	450	1450																																							
Pattern	Stock Length ↑↓	Count ↑↓																																							
1	1200	4	200																																						
2	1200	6	182 (+18) 181 (+19) 280 (+20) 375																																						
3	1200	10	180 (+20) 280 (+20) 280 (+20)																																						
4	1200	6	375 375 375																																						

Rows per page: 10 < > 1-4 of 4

Rysunek 4.5. Eksport wyników do CSV

4. System wspierający cięcie

Zrzuty ekranu 4.6, 4.7 i 4.8 pokazują wszystkie pozostałe formy, jakie przyjmować może interfejs użytkownika. Zależny jest on od aktualnie stosowanego algorytmu.

The screenshot displays the Steel Cutting System interface with the following sections:

- Algorithm Settings:**
 - Minimize: Cost (selected)
 - Relaxation type: Manual (selected)
 - Solver: Cbc (selected)
- Stock Table:**

	Length ↑↓	Count ↑↓
1200	100	

Add new
- Order Table:**

	Length ↑↓	Count ↑↓	Max Relax ↑↓
200	20	Max Relax	
200	38	25	
300	35	20	
375	24	Max Relax	

10 < 1-4-of 4 >
- Generate Result:**

Relax Cost Multiplier: 0
- Cost Table:**

Cost ↑↓	Waste ↑↓	Total Relax ↑↓
0	0	1800
- Results Table:**

Pattern ↑↓	Stock Length ↑↓	Count ↑↓
1	1200	4
2	1200	24
3	1200	3

Rows per page: 10 1-3 of 3

Rysunek 4.6. Przykładowe rozwiązań z minimalizacją kosztów

Algorithm Settings

Minimize:	<input checked="" type="radio"/> Cost	<input type="radio"/> Waste	
Relaxation type:	<input type="radio"/> Manual	<input checked="" type="radio"/> Auto	<input type="radio"/> Single Step
Solver:	<input checked="" type="radio"/> Cbc	<input type="radio"/> HiGHS	<input type="radio"/> CPLEX

Algorithm Description

This model attempts to determine the relaxation value automatically. It is up to the user to decide which items can be subject to relaxation. Under the 'Generate Result' button, there is a switch that allows for enabling or disabling the generation of basic patterns. By basic patterns, we mean patterns consisting of a single type of final item.

Stock Table

	Length ↑↓	Count ↑↓	Cost ↑↓
1200	100	1200	
1000	100	1000	
			Add new

Order Table

	Length ↑↓	Count ↑↓	Can be relaxed
	158	38	
	200	20	
	261	45	
	310	35	
	450	15	
	500	10	
			Add new
			1-6 of 6 < >

Results

Cost ↑↓	Waste ↑↓	Total Relax ↑↓
37400	0	7459

Rysunek 4.7. Przykładowe rozwiązanie z relaksacją automatyczną

4. System wspierający cięcie

The screenshot shows the Steel Cutting System interface with the following sections:

- Algorithm Settings**:
 - Minimize: Cost (selected)
 - Relaxation type: Single Step (selected)
 - Solver: Cbc (selected)
- Stock Table**:

	Length ↑↓	Count ↑↓	Cost ↑↓	Generate next step
1200	100	1200	0	Generate next step
1000	100	1000	0	
- Order Table**:

	Length ↑↓	Count ↑↓	Can be relaxed
158	38	38	>
200	20	20	>
261	45	45	>
310	35	35	>
450	15	15	>
500	10	10	>
- Buttons**:
 - Add new
 - Generate Result
 - Generate Next Pattern
 - 1-6 of 6
- Algorithm Description**:

This model determines increasingly larger relaxation values, starting from the optimal solution without relaxation. It is up to the user to decide which items can be subject to relaxation. The user also selects a base item for which the next pattern will be determined. The generation of the next pattern is performed by pressing the 'Generate Next Pattern' button.
- Cost Table**:

Cost ↑↓	Waste ↑↓	Total Relax ↑↓
44513	219	60

Rysunek 4.8. Przykładowe rozwiązanie z relaksacją krokową

Na zrzutach ekranu 4.9 pokazano wygląd aplikacji podczas korzystania z niej na urządzeniu mobilnym.

Algorithm Settings

Minimize:	<input checked="" type="radio"/> Cost	Relaxation type:	<input checked="" type="radio"/> Manual	<input type="radio"/> Auto
<input type="radio"/> Waste	<input type="radio"/> Single Step	Solver:	<input checked="" type="radio"/> Cbc	<input type="radio"/> HIGHS
Algorithm Description This model assumes that the maximum relaxation value will be provided in the input data. The solution found will include relaxed lengths only if they improve the final result. Relaxation values will be applied sparingly to avoid generating additional waste.				

Stock Table

	Length ↑↓	Count ↑↓	Cost ↑↓
<input type="checkbox"/>	1200	100	1200
<input type="checkbox"/>	1000	100	1000

Add new 10 < >

Generate Result

Relax Cost Multiplier 0

Order Table

	Length ↑↓	Count ↑↓	Max Relax ↑↓
<input type="checkbox"/>	158	38	Max Relax
<input type="checkbox"/>	200	20	Max Relax
<input type="checkbox"/>	261	45	Max Relax
<input type="checkbox"/>	310	35	Max Relax
<input type="checkbox"/>	450	15	Max Relax
<input type="checkbox"/>	500	10	Max Relax

Add new 10 < >

	Cost ↑↓	Waste ↑↓	Total Relax ↑↓
	44800	451	0

Stock Table

	Length ↑↓	Count ↑↓	Cost ↑↓
<input type="checkbox"/>	1200	100	1200
<input type="checkbox"/>	1000	100	1000

Add new 10 < >

Generate Result

Relax Cost Multiplier 0

Order Table

	Length ↑↓	Count ↑↓	Max Relax ↑↓
<input type="checkbox"/>	158	38	Max Relax
<input type="checkbox"/>	200	20	Max Relax
<input type="checkbox"/>	261	45	Max Relax
<input type="checkbox"/>	310	35	Max Relax
<input type="checkbox"/>	450	15	Max Relax
<input type="checkbox"/>	500	10	Max Relax

Add new 10 < >

	Cost ↑↓	Waste ↑↓	Total Relax ↑↓
	44800	451	0

Summary

Pattern ↑↓	Stock Length ↑↓	Count ↑↓	0	1
1	1200	9	158	158
2	1200	8	261	310
3	1200	2	261	450
4	1000	11	158	261
5	1000	4	200	200

Rows per page 5 < >

Rysunek 4.9. Wygląd aplikacji w urządzeniu mobilnym Samsung Galaxy A51

4.5. Testy aplikacji

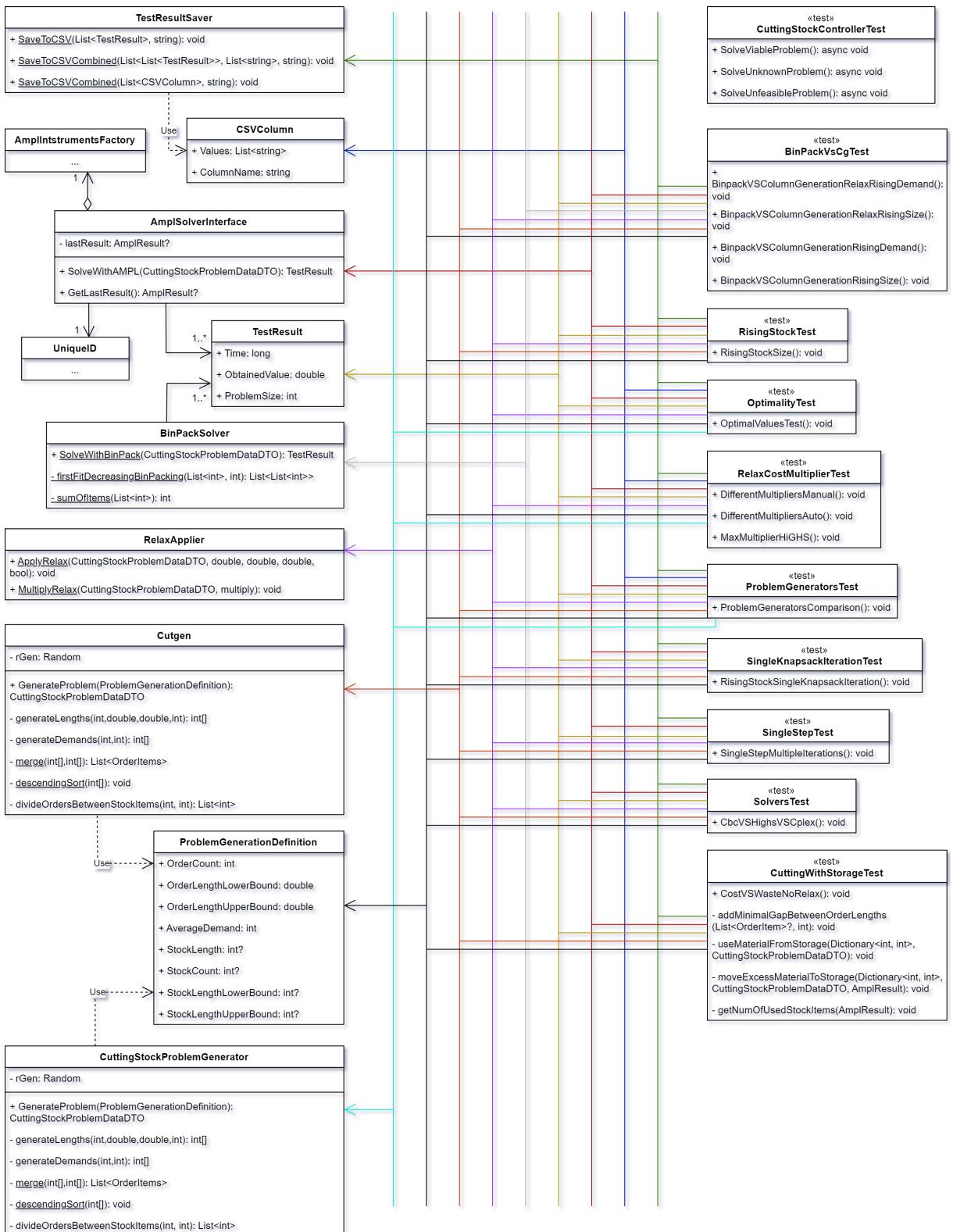
W oddzielnym projekcie testowym, korzystającym z narzędzia xUnit, opracowano testy integracyjne backendu aplikacji internetowej. Miały one za zadanie:

1. Zweryfikować działanie kontrolera zapytań HTTP.
2. Zweryfikować działanie oprogramowania pośredniego ograniczającego liczbę zapytań od poszczególnych hostów.
3. Przetestować działanie interfejsu AMPL.
4. Przetestować efektywność algorytmów (dokładniej o tym w rozdziale 5)

Wszystkie testy ujęte zostały w dziewięciu klasach, które to grupują je według ich charakteru. Klasami tymi są:

- CuttingStockControllerTest – Testy integracyjne kontrolera zapytań HTTP. Weryfikują zwracane przez niego wartości i obsługę błędów. Dodatkowo sprawdzane jest działanie RequestRateLimitera.
- ProblemGeneratorsTest – Porównanie dwóch generatorów problemów cięcia. Więcej o tym w podrozdziale 5.1.
- OptimalityTest – Weryfikowanie uzyskiwanych wyników przy użyciu różnych modeli względem rozwiązania optymalnego. Więcej o tym w podrozdziale 5.2.
- RelaxCostMultiplierTest – Badanie wpływu zmiany kosztu relaksacji na ostateczny wynik. Więcej o tym w podrozdziale 5.2.
- RisingStockTest – Badania wpływu ilości prêtów bazowych na czas wykonywania zadania. Więcej o tym w podrozdziale 5.2.2.
- SingleKnapsackIterationTest – Porównanie modelu dodatkowego (54-61) z modelem iterującym po prêtach bazowych. Więcej o tym w podrozdziale 5.2.2.
- SingleStepTest – Testy relaksacji krokowej. Więcej o tym w podrozdziale 5.2.4.
- CuttingWithStorageTest – Test mający na celu wskazać która minimalizacja, odpadów czy kosztów, jest bardziej opłacała, gdy użytkownik ma zdolność magazynowania prêtów. Więcej o tym w podrozdziale 5.2.5.
- BinPackVsCgTest – Porównanie wyników działania metody generacji kolumn z algorymem pakowania pojemników. Więcej o tym w podrozdziale 5.2.6.
- SolversTest – Porównanie działania trzech popularnych solwerów programowania liniowego i programowania całkowitoliczbowego mieszanego. Więcej o tym w podrozdziale 5.2.7.

Testy interfejsu użytkownika nie były automatyzowane z uwagi na nieduży poziom jego skomplikowania. Przeprowadzane zostały one manualnie.



Rysunek 4.10. Uproszczony diagram klas testów.

4.6. Proces instalacji

Budowanie systemu jest w znacznym stopniu zautomatyzowane przez platformę programistyczną **Visual Studio 2022**. Proces komplikacji opisany w tym podrozdziale zakłada, że będzie ona dostępna. Konsekwencją tego jest to, że instrukcja przeznaczona będzie tylko użytkowników systemu **Windows**. Dla innych systemów należałoby wykorzystać alternatywne narzędzia wspierające komplikację .NET.

4.6.1. Pobranie zależności

Pierwszym krokiem do zbudowania aplikacji będzie pozyskanie kodu źródłowego. Jest on dostępny w repozytorium kodu github pod linkiem:

<https://github.com/JAJA1706/SteelCuttingSystem>

Do pobrania zależności frontendu wykorzystane jest narzędzie npm. Instalator tego narzędzia, jak i środowiska uruchomieniowego Node.js pobrać można ze strony:

<https://nodejs.org/en/download/> Po zainstalowaniu npm, należy przejść w konsoli do folderu "*SteelCuttingSystem/steeltocutoptimizer.client*", a następnie wywołać komendę "**npm install**". Pobierze ona wszystkie wymagane przez frontend zależności.

Kolejnym krokiem będzie pobranie i zainstalowanie środowiska AMPL, wraz z jego solwerami. Ich instalator można otrzymać po uprzednim zalogowaniu pod linkiem:

<https://portalAMPL.com/user/AMPL/request/AMPLce>. Rozwiązywanie większych problemów (ponad 500 zmiennych) wymagać będzie rejestracji AMPL na komputerze. Pod powyższym linkiem znaleźć można instrukcję jak tego dokonać. Jest to niemniej jednak krok opcjonalny. Podczas instalacji AMPL należy upewnić się, że zaznaczona będzie opcja dodania ścieżek do solwerów i aplikacji jako zmiennych środowiskowych. Jest to domyślne ustawienie.

W folderze "*SteelCuttingSystem/SteelCutOptimizer.Server/lib*" znaleźć się muszą biblioteki AMPL API przeznaczone dla języka C#. W celu uproszczenia procesu instalacji owe biblioteki zostały już dołączone do kodu źródłowego. Podczas budowy backendu, Visual Studio przenosi zawartość folderu "*SteelCuttingSystem/SteelCutOptimizer.Server/lib*" jak i folderu "*SteelCuttingSystem/AMPL*" w miejsce, w którym powstaje działająca aplikacja, aby miała ona do nich bezpośredni dostęp.

Budowanie całej aplikacji odbywa się przy użyciu solucji "*SteelCuttingSystem/SteelCutOptimizer.sln*". Skupia ona trzy projekty, frontendu, backendu i testów. Przed komplikacją należy upewnić się, czy Visual Studio przystosowane jest do pracy z projektami .NET. W instalatorze Visual Studio zaznaczone powinny być następujące obciążenia: **Opracowywanie zawartości dla platformy ASP.NET, Programowanie aplikacji klasycznych dla platformy .NET**.

4.6.2. Uruchomienie

Po skompletowaniu wszystkich zależności otworzyć można solucję *SteelCutOptimizer.sln* i przystąpić do procesu komplikacji wszystkich projektów. Po udanej komplikacji

najprostszym sposobem na uruchomienie aplikacji jest skonfigurowanie opcji uruchamiania jako "Wiele projektów startowych" i zaznaczenie jako aktywnych projektów *SteelCutOptimizer.Server* i *steelcutoptimizer.client*. Alternatywnym sposobem będzie manualne uruchomienie frontendu przy użyciu komendy "**npm run dev**" (uprzednio znajdującej się w folderze "*SteelCuttingSystem/steelcutoptimizer.client*") oraz backendu za pomocą visual studio przy korzystaniu z opcji uruchamiania "Pojedynczy projekt startowy" i profilu uruchamiania o nazwie *https*.

Frontend aplikacji domyślnie działa na porcie 5173. Aby więc otworzyć okno interfejsu użytkownika systemu cięcia, należy przejść w dowolnej przeglądarce internetowej na adres: <https://localhost:5173/>

5. Testy algorytmów

5.1. Przygotowanie danych

Przeprowadzenie licznych testów modeli programowanie liniowego, wymagało użycia generatora problemów cięcia, który mając zdefiniowane parametry określające:

- liczbę typów prętów bazowych
- minimalną długość prętów bazowych
- maksymalną długość prętów bazowych
- liczbę typów prętów finalnych
- minimalną część długości pręta bazowego będącą długością pręta finalnego
- maksymalną część długości pręta bazowego będącą długością pręta finalnego
- średnie zapotrzebowanie prętów finalnych

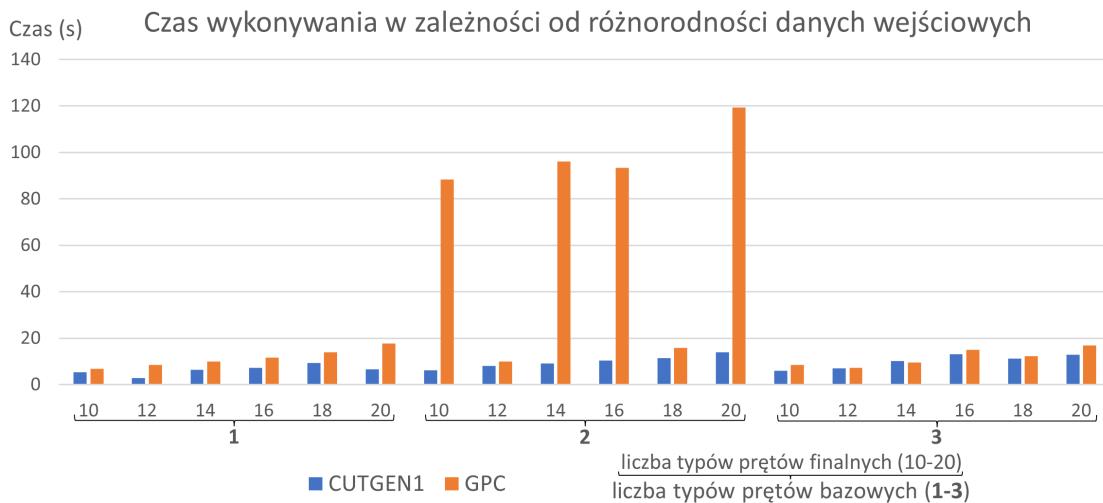
był w stanie tworzyć losowe problemy cięcia.

Ostatecznie wykorzystano dwie różne implementacje generatorów. Pierwszy z nich, o nazwie **CUTGEN1**, opierał się na algorytmie zaproponowanym w artykule [18]. Oryginalna implementacja autorów artykułu napisana była w języku Fortran. Powstał jednak jej port w języku Java [19], który to na potrzeby tej pracy został zaimplementowany przez autora w języku C# i rozwinięty o możliwość generowania różnych typów prętów bazowych.

CUTGEN1 posiada jednak jedną wadę. Nie jest on w stanie podać rozwiązania optymalnego wygenerowanego problemu. Aby być w stanie dokładniej określić efektywność przygotowanych w pracy modeli, autor zaimplementował drugi generator pozbawiony tej wady. Jego zasada działania jest następująca. Mając przygotowany zestaw prętów bazowych, po kolei dzieli każdego z nich na krótsze odcinki. Stają się one zamówieniem, rozwiązanie optymalne zaś równe jest liczbie użytych prętów bazowych. Generator nazwany został **GPC** (Generator problemów cięcia).

Na wykresie 5.1 przedstawiono porównanie czasu rozwiązywania problemów wygenerowanych przez CUTGEN1 i GPC. Generatory otrzymywały na wejściu takie same parametry. Ich zadaniem było zwrócenie problemów o różnej liczbie typów prętów bazowych (1-3) i różnej liczbie typów prętów finalnych (10-20). Następnie problemy te rozwiązywane były identycznym modelem programowania liniowego, w tym przypadku był to model z relaksacją manualną. Wartość maksymalnej relaksacji i indeksy prętów poddawanych skróceniu także były losowane z określonych przedziałów.

Z analizy wykresu 5.1 wynika, że problemy wygenerowane za pomocą GPC w przeważającej części przypadków wymagają dłuższego czasu na rozwiązanie. Niepokojąco wyglądać też mogą cztery odbiegające znacznie wartości. Solwery programowania liniowego uruchamiane były dla wszystkich testów z warunkiem na rozwiązanie problemu maksymalnie w 240 sekund. Przekroczenie tego progu skutkowało zwróceniem najlepszego osiągniętego do tej pory rozwiązania. Większość testów przeprowadzana była też w seriach, których



Rysunek 5.1. Porównanie generatorów problemu cięcia

wynik później był uśredniany. W przypadku porównania generatorów były to 3 przebiegi na jedną wartość przedstawioną na wykresie 5.1. Wywnioskować więc można, że na 54 problemy wygenerowane przez GPC, 4 z nich były zbyt trudne do rozwiązania przez stosowany w teście solwer¹⁴ w zadanej ramie czasowej.

Ponieważ CUTGEN1 jest generatorem sprawdzonym, tworzącym regularniejsze problemy od GPC, to jego zdecydowano się wykorzystać w większości testów. Jedynie w kolejnym badaniu 5.2.1, w którym to informacja o rozwiązaniu optymalnym będzie potrzebna, użyty zostanie GPC.

5.2. Testy modeli programowania liniowego

Testy przedstawione w tym podrozdziale zostały utworzone we framework'u testowym xUnit zintegrowanym w pełni z Visual Studio w języku C#. Interfejsem do komunikacji z AMPL były metody zaimplementowane jako backend aplikacji internetowej z rozdziału 4. Przeprowadzone badania pełniły więc też rolę testów integracyjnych.

5.2.1. Efektywność relaksacji manualnej i automatycznej

Wykres 5.2 prezentuje skuteczność metody generacji kolumn w rozwiązywaniu problemu cięcia. Parametry wejściowe generatora problemów to:

- liczba typów prętów bazowych: 1-3
- minimalna długość prętów bazowych: 900
- maksymalna długość prętów bazowych: 1200
- liczba typów prętów finalnych: 10-20
- minimalna długość pręta finalnego: 10% pręta bazowego
- maksymalna długość pręta finalnego: 80% pręta bazowego

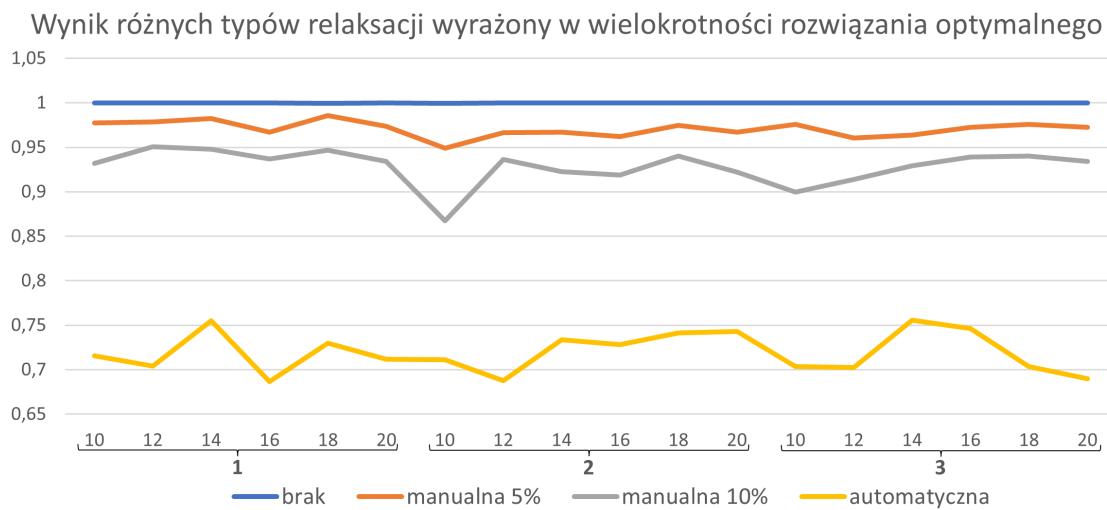
¹⁴ W tym przypadku był to solwer Cbc, jednak ustalone, że np. HiGHS z tymi trudnymi wariantami był w stanie sobie poradzić.

5. Testy algorytmów

- średnie zapotrzebowanie prętów finalnych: 100

Parametry relaksacji to:

- procent relaksowanych prętów: 50%
- minimalna wartość relaksacji manualnej: 0% pręta finalnego
- maksymalna wartość relaksacji manualnej: 20% pręta finalnego



Rysunek 5.2. Efektywność różnych typów relaksacji

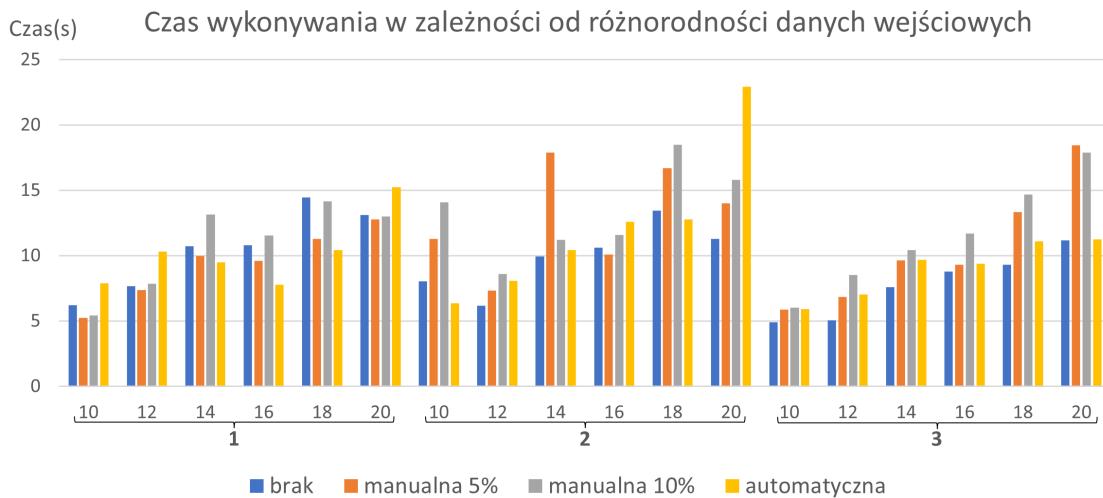
Bez stosowania żadnej relaksacji metoda generacji kolumn zwracała wyniki generujące średnio 0.001% nadwyżki względem rozwiązania optymalnego. Warto jednak mieć na uwadze to, że im większe jest zamówienie pod względem liczby zamówionych prętów finalnych, tym nadwyżka w punktach procentowych względem optimum będzie malała. Związana jest ona bowiem tylko z dodaniem ograniczeń całkowitoliczbowych w ostatnim kroku metody generacji kolumn (Alg. 1).

Ten sam problem rozwiązyany bez relaksacji, trafiał także do modelów z relaksacją. W przypadku relaksacji "manualna 10%" wartości maksymalne relaksacji były jedynie podwajane (nielosowane) względem "manualna 5%". Dzięki temu można dobrze zaobserwować nieproporcjonalność wyniku do stosowanej relaksacji. Podwojenie jej nie oznacza uzyskania dwa razy lepszego wyniku.

Zastosowanie relaksacji manualnej na poziomie 5% skutkowało uzyskaniem rozwiązań średnio 2,7% lepszych od optymalnego bez relaksacji. Podwojenie relaksacji podwyższyło ten wynik do 8,4%.

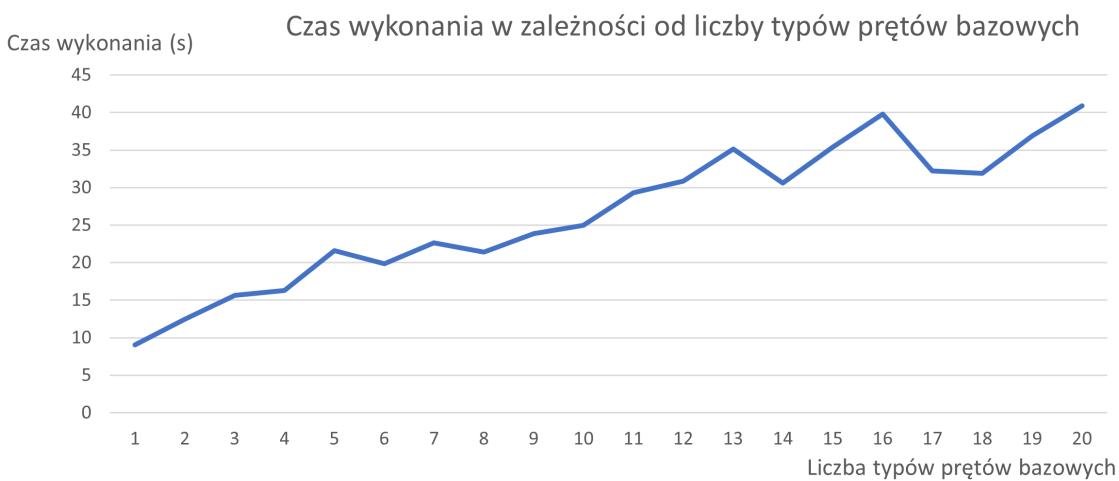
Zastosowanie relaksacji automatycznej skutkowało wynikami dużo lepszymi niż te uzyskane dla średniej relaksacji 10%. Oznacza to, że algorytm zakładał skrócenie prętów o miary przewyższające te, które są możliwe w praktyce. Nie oznacza to jednak, że technika ta nie ma zastosowania. Użytkownik korzystający z niej nie musi bowiem decydować się na użycie wszystkich wygenerowanych przez nią wzorców, jak to odbywało się podczas testów.

5.2.2. Wydajność relaksacji manualnej i automatycznej



Rysunek 5.3. Wydajność algorytmu dla różnych typów relaksacji

Z wykresu 5.3 wynika, że czas rozwiązywania problemu cięcia metodą generacji kolumn zwiększa się wraz z liczbą typów prętów bazowych i prętów finalnych. Co więcej, stosowanie relaksacji również wpływa na czas wykonania. Średnio wydłużał się on o 20% podczas stosowania relaksacji manualnej i o 10% podczas automatycznej. Spowodowane jest to większą liczbą zmiennych w modelu z relaksacją niż bez niej.



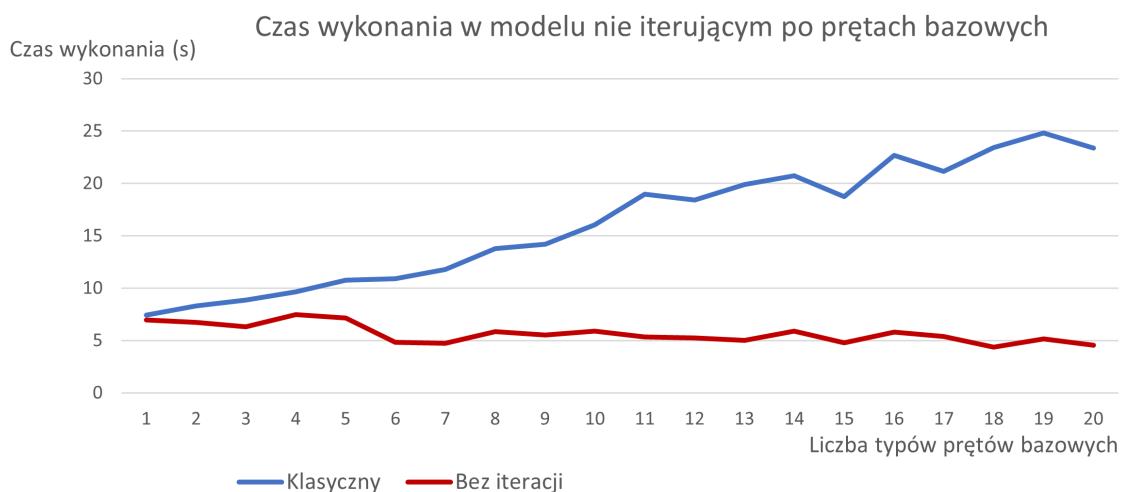
Rysunek 5.4. Wydajność metody generacji kolumn w zależności od liczby prętów bazowych

Wykres 5.4 ilustruje zależność czasu wykonywania od liczby typów prętów bazowych. W problemach generowanych do tego testu liczba typów prętów finalnych była stała i wynosiła 20. Wzrost czasu wykonywania spowodowany jest głównie koniecznością wielokrotnego rozwiązywania podproblemu plecakowego. Do wniosku takiego można dojść, także przyglądając się wykresowi 5.5. Przedstawia on wykorzystanie w praktyce modelu

5. Testy algorytmów

(54-61), który to umożliwia zastosowanie generacji kolumn bez wielokrotnego iterowania po prętach bazowych. Czas wykonywania w jego przypadku okazuje się w ogóle nie zależeć od liczby typów prętów bazowych.

Wykres 5.4 przedstawiał czas wykonania dla problemów minimalizacji kosztu, zaś 5.5 dla problemów minimalizacji odpadów. Porównując ze sobą te diagramy, wywnioskować można, że problemy minimalizacji kosztu są bardziej złożone obliczeniowo niż te minimalizujące odpady.

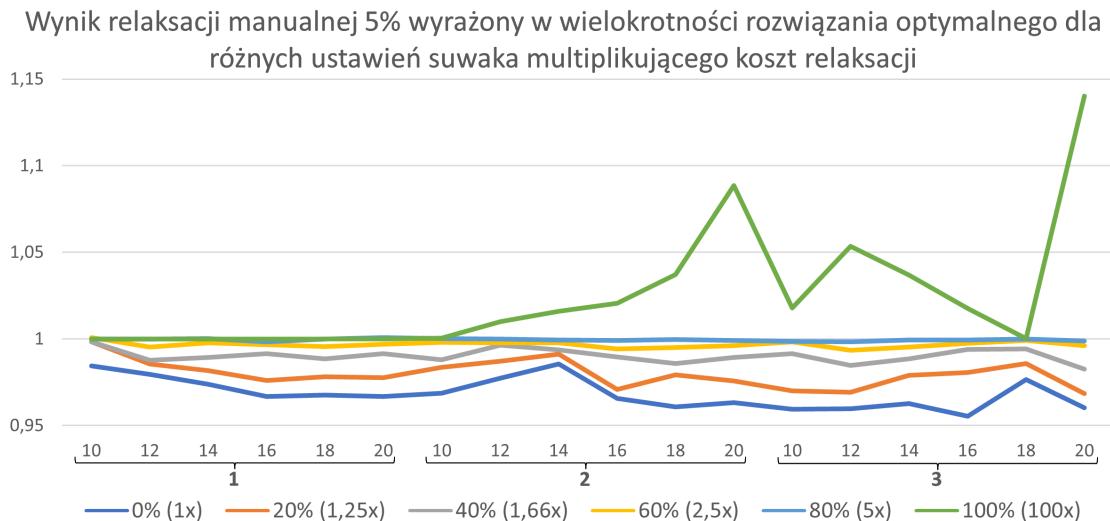


Rysunek 5.5. Wydajność modelu (54-61) nie iterującego po prętach bazowych porównana z modelem klasycznym (49)

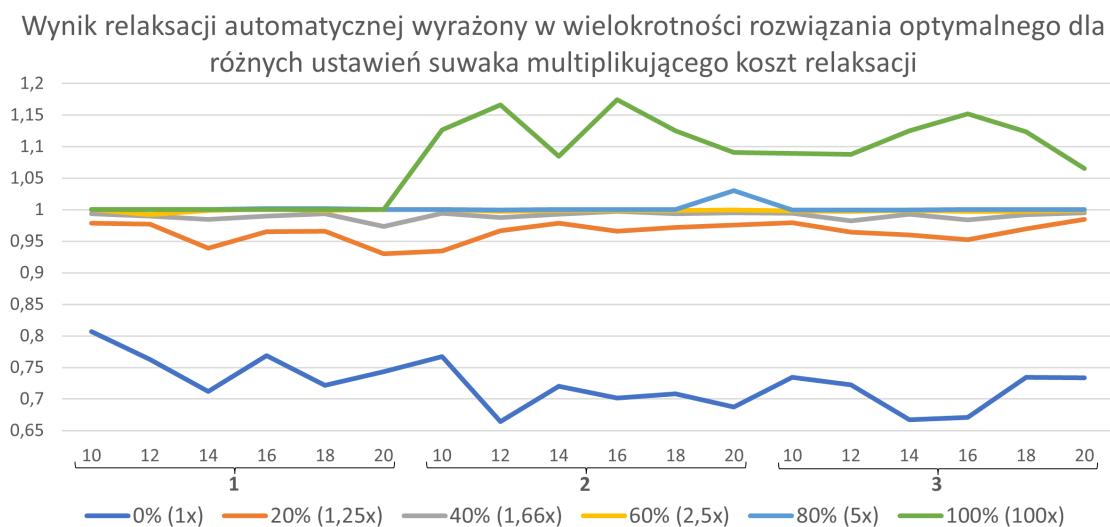
W teorii wzrost liczby prętów bazowych wiąże się z potęgowym wzrostem możliwych rozwiązań, zaś wzrost liczby prętów finalnych z wykładniczym. Wykresy przedstawione w tym podrozdziale świetnie demonstrowają efekt stosowanie metody generacji kolumn, gdyż czasy wykonywania algorytmów nie rosły w tak znacznym tempie.

5.2.3. Modyfikacja kosztu relaksacji

Użytkownik aplikacji jest w stanie skalować w pewnym zakresie wartość kosztu relaksacji dla algorytmów relaksacji manualnej i automatycznej. Dokonuje tego, ustawiając w odpowiedni sposób suwak multiplikujący koszt relaksacji. Suwak może przyjąć 100 różnych pozycji, zwracając przy tym wartości z przedziału (0-0.99). Wysokość multiplikacji wyznaczana jest ze wzoru $1/(1-z)$, gdzie z to wartość zwrócona przez suwak. Tym samym przedział możliwych do uzyskania multiplikacji to (1x-100x). Poniższy test pokazać ma, w jaki sposób zmieniają się wyniki uzyskiwane podczas korzystania z relaksacji manualnej i automatycznej dla sześciu różnych pozycji suwaka. Przedstawiają je wykresy 5.6 oraz 5.7.



Rysunek 5.6. Wyniki relaksacji manualnej w zależności od ustawienia suwaka multiplikującego wartość kosztu relaksacji.



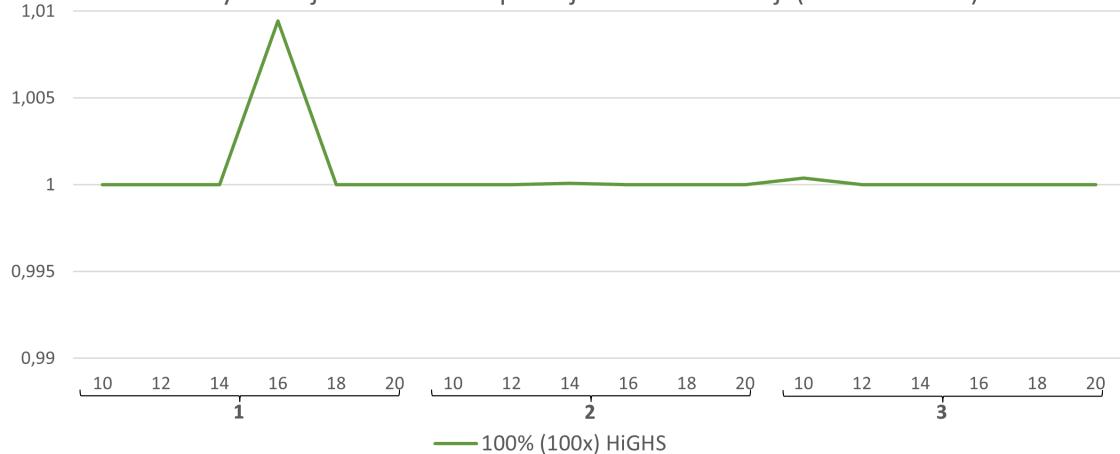
Rysunek 5.7. Wyniki relaksacji automatycznej w zależności od ustawienia suwaka multiplikującego wartość kosztu relaksacji.

Zauważać można, że za pomocą suwaka użytkownik może w łatwy sposób kontrolować poziom nakładania relaksacji. Jest to w szczególności istotne podczas korzystania z relaksacji automatycznej, którą bez suwaka ciężko byłoby ukierunkować. Niepokojące mogą być jednak wyniki uzyskane dla wysokich wartości mnożnika. Dalekie są one od rozwiązania optymalnego. Powodem tego jest nieprawidłowe działanie (lub skonfigurowanie) solwera Cbc. Zbyt wcześnie rezygnuje on z rozwiązywania problemu, gdyż zakłada, że znalezienie lepszego rozwiązania będzie trudne lub niemożliwe. W teorii zaś powinien on po prostu rozwiązać problem bez korzystania z relaksacji. W przypadku użycia innych solwerów np. HiGHS, taki problem nie występuje, co pokazane jest na wykresie 5.8.

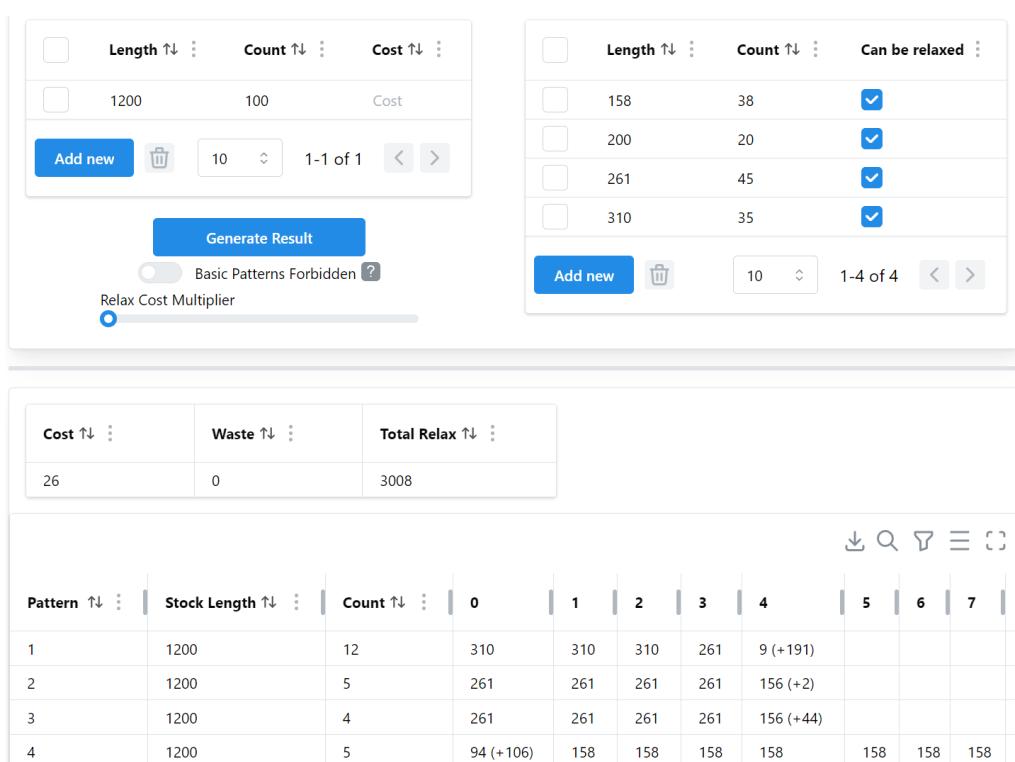
5. Testy algorytmów

Wszystkie testy przeprowadzone poza niniejszym podrozdziałem korzystały z domyślnej wartości multiplikatora, to jest 1x.

Wynik relaksacji automatycznej wyrażony w wielokrotności rozwiązania optymalnego dla maksymalnej wartości multiplikacji kosztu relaksacji (solwer HiGHS)



Rysunek 5.8. Wyniki relaksacji automatycznej w zależności od maksymalnej multiplikacji kosztu relaksacji dla solwera HiGHS



Rysunek 5.9. Relaksacja automatyczna z podstawowym ustawieniem suwaka

The screenshot displays a software interface for pattern relaxation. At the top left is a table for 'Length ↑' and 'Count ↑' with a row for '1200' and '100'. Below it is a 'Generate Result' button and a slider for 'Relax Cost Multiplier' set to 1.2x. To the right is another table for 'Length ↑' and 'Count ↑' with rows for 158, 200, 261, and 310, all marked as 'Can be relaxed'. Below these are two summary tables: one for 'Cost ↑' and 'Waste ↑' showing values 28 and 219 respectively, and another for 'Total Relax ↑' showing a total of 201. At the bottom is a detailed table showing the count of each pattern length (1200) across six categories (0, 1, 2, 3, 4, 5, 6), with some values including multipliers like '(+7)'.

Length ↑	Count ↑	Cost ↑
1200	100	Cost

Length ↑	Count ↑	Can be relaxed
158	38	<input checked="" type="checkbox"/>
200	20	<input checked="" type="checkbox"/>
261	45	<input checked="" type="checkbox"/>
310	35	<input checked="" type="checkbox"/>

Cost ↑	Waste ↑	Total Relax ↑
28	219	201

Pattern ↑	Stock Length ↑	Count ↑	0	1	2	3	4	5	6
1	1200	7	310	310	310	261			
2	1200	7	310	310	194 (+6)	193 (+7)	193 (+7)		
3	1200	1	261	261	261	261			
4	1200	8	261	261	261	261	156 (+2)		
5	1200	5	252 (+9)	158	158	158	158	158	158

Rysunek 5.10. Relaksacja automatyczna z multiplikatorem 1.2x

This screenshot shows the same software interface as Rysunek 5.10, but with a different relaxation multiplier. The 'Relax Cost Multiplier' slider is now set to 2x. The results show a significant reduction in waste and cost. The 'Total Relax ↑' table shows a total of 59. The detailed table at the bottom shows a higher count of pattern 1 (12) and a lower count of pattern 5 (1).

Length ↑	Count ↑	Cost ↑
1200	100	Cost

Length ↑	Count ↑	Can be relaxed
158	38	<input checked="" type="checkbox"/>
200	20	<input checked="" type="checkbox"/>
261	45	<input checked="" type="checkbox"/>
310	35	<input checked="" type="checkbox"/>

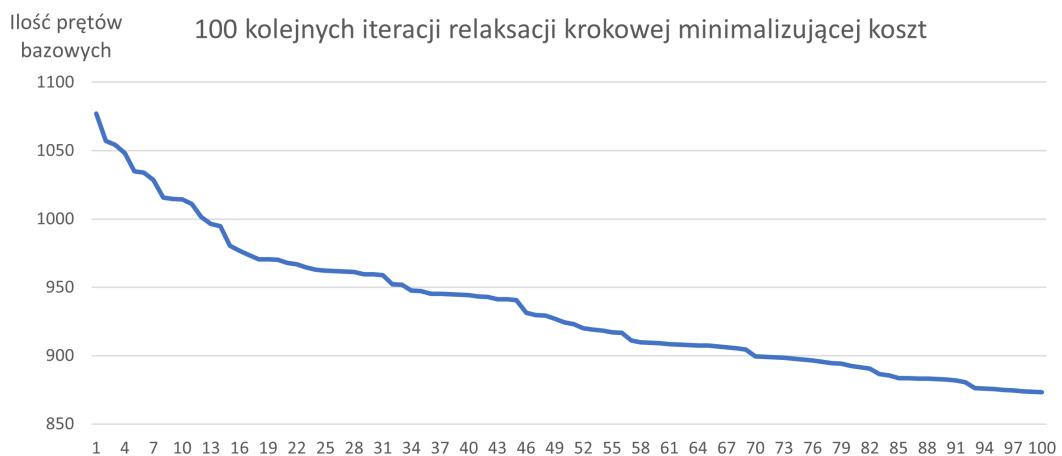
Cost ↑	Waste ↑	Total Relax ↑
29	202	59

Pattern ↑	Stock Length ↑	Count ↑	0	1	2	3	4	5	6
1	1200	12	310	310	310	261			
2	1200	7	261	261	261	261	156 (+2)		
3	1200	5	252 (+9)	158	158	158	158	158	158
4	1200	4	200	200	200	200	200	200	
5	1200	1	158	158	158	158	158	158	158

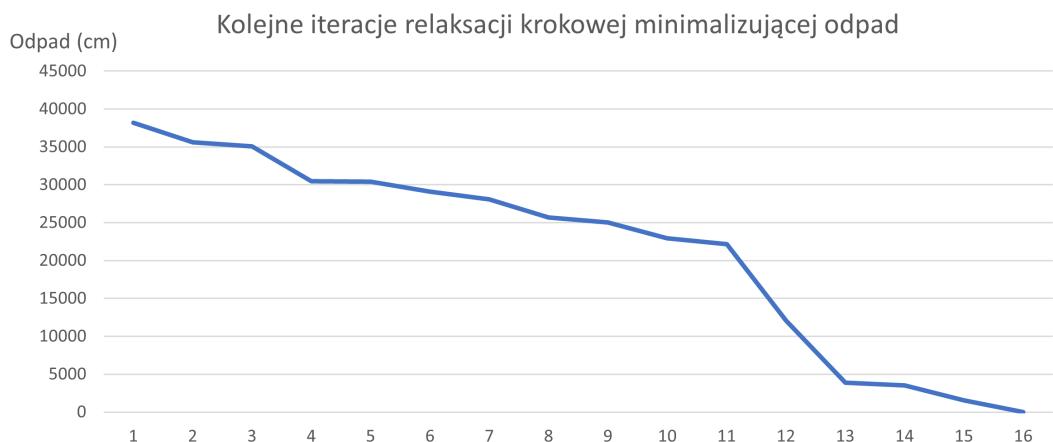
Rysunek 5.11. Relaksacja automatyczna z multiplikatorem 2x

5.2.4. Testy relaksacji krokowej

Ponieważ relaksacji krokowej nie da się bezpośrednio porównać z innymi stosowanymi metodami, przedstawione zostaną jedynie jej dwa przykładowe przebiegi dla minimalizacji kosztu i minimalizacji odpadu. Ich koniec następuje w momencie osiągnięcia optimum lub wykonania stu iteracji.



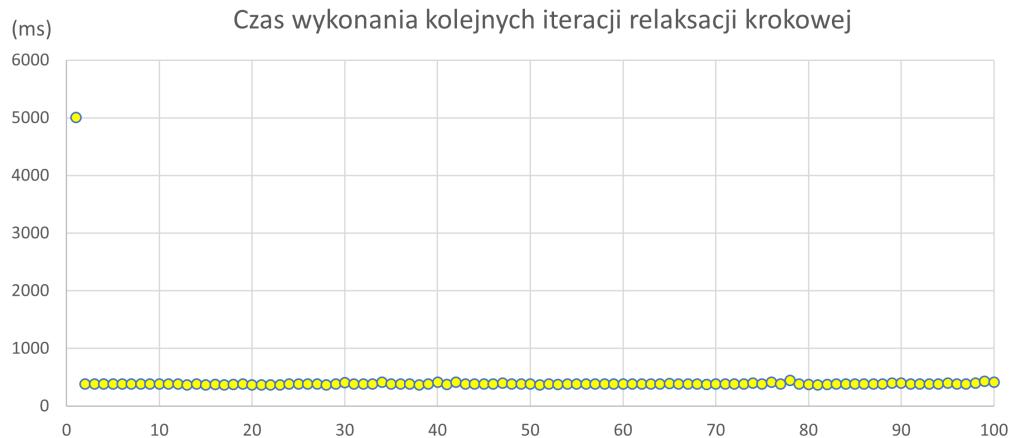
Rysunek 5.12. Przykładowy przebieg relaksacji krokowej minimalizującej koszt.



Rysunek 5.13. Przykładowy przebieg relaksacji krokowej minimalizującej odpad.

Zgodnie z wykresem 5.13 minimalizacja odpadu była w stanie osiągnąć optimum w szesnastu krokach. Tego samego nie można powiedzieć o minimalizacji kosztu, która zakończyła działanie po stu iteracjach. Kolejny raz można wyciągnąć wniosek, że minimalizacja odpadów jest procesem mniej złożonym obliczeniowo.

Z wykresu punktowego 5.14 odczytać można, że czas rozwiązywania pojedynczych iteracji relaksacji krokowej jest stały i znikomy względem czasu potrzebnego do wstępniego rozwiązania problemu bez relaksacji.



Rysunek 5.14. Czasy wykonywania poszczególnych iteracji relaksacji krokowej.

5.2.5. Problem cięcia z możliwością magazynowania prętów

Gdy użytkownik zamierza rozwiązać problem cięcia tylko jeden raz w długim przedziale czasowym lub gdy nie ma możliwości magazynowania prętów finalnych, powinien zdecydować się na użycie jednego z modeli minimalizującego koszt. Zapewni mu to najlepsze na daną chwilę wzorce cięcia. W przeciwnym wypadku użytkownik powinien rozważyć zastosowanie minimalizacji odpadów.



Rysunek 5.15. Problem cięcia z możliwością magazynowania.

Wykres 5.15 obrazuje wyniki testu, w którym nadmiernie wyprodukowane pręty były zapamiętywane. W przypadku gdy w którymś z kolejnych zamówień pojawiał się pręt, który był przechowywany, jego liczba z magazynu była odejmowana od tej z zamówienia. Aby zwiększyć prawdopodobieństwo pojawienia się podobnych prętów, ich długości w tym teście były zaokrąglane do wielokrotności 100.

5. Testy algorytmów

Zgodnie z oczekiwaniami, na początku testu wyniki minimalizacji kosztu były korzystniejsze. Po pewnym czasie, to jest około 30 zamówieniach, minimalizacja odpadu nadrobiła straty i dawała wyniki lepsze od konkurencyjnych. Przykład ten pokazuje, że nie zawsze preferowaną metodą będzie minimalizacja kosztu.

5.2.6. Porównanie z algorytmem pakowania pojemników

Do tej pory w pracy wszystkie testy przeprowadzane były z udziałem metody generacji kolumn. Aby sprawdzić, jak prezentuje się ona na tle innych algorytmów, zaimplementowano algorytm pakowania do pojemników z heurystyką pierwszego dopasowania malejącego omówiony w podrozdziale 2.4.

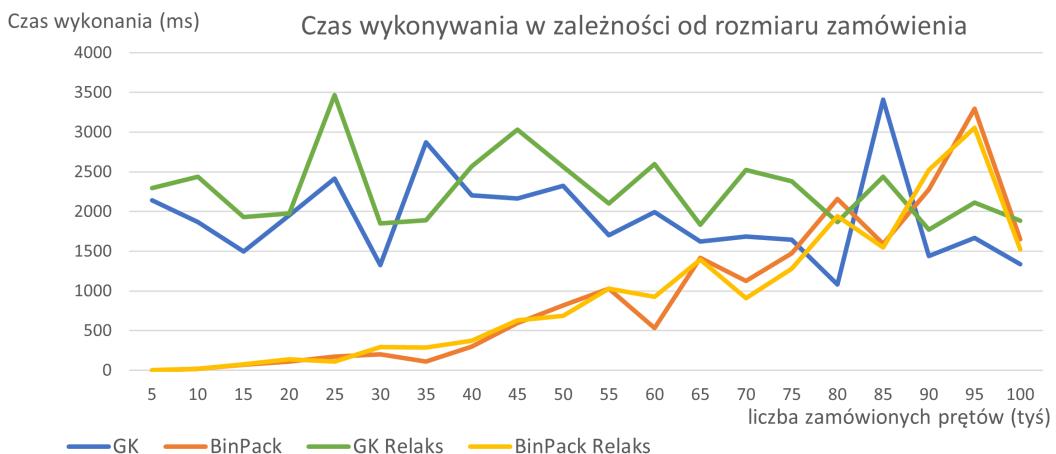
Przypadki testowe porównujące metodę generacji kolumn i algorytm pakowania zakładały wygenerowanie problemów o jednym typie pręta bazowego, lecz o zmieniającym się rozmiarze zamówienia. Zwiększało się ono pod względem sumy zamówionych prętów lub pod względem różnorodności typów prętów finalnych. Relaksacja nakładana była losowo tak jak w przypadku innych testów. Jej działanie w algorytmie pakowania było jednak uproszczone. Wartość maksymalna relaksacji była od razu traktowana jako skrócenie pręta o jej wartość. Powstawały przez to nadmierne odpadki, lecz minimalizacja i tak odbywała się po koszcie, więc nie zmieniało to rozwiązania.

Algorytm pakowania pojemników nie jest tak skomplikowany, jak metoda generacji kolumn, co jest jego dużą zaletą. Wymaga on jednak iteracji po każdym pręcie finalnym, więc rozmiar zamówienia pod względem sumy zamówionych prętów finalnych będzie miał na niego wpływ. Ten fakt ukazuje wykres 5.16. Wraz ze wzrostem liczby zamówionych prętów, czas wykonania algorytmu pakowania rósł, zaś metody generacji kolumn utrzymywał się na stałym poziomie.

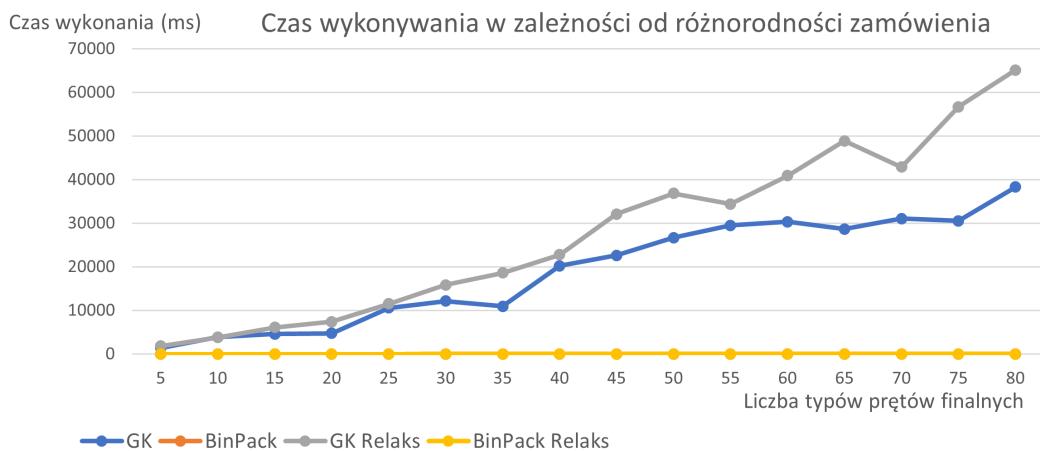
Odwrotna sytuacja pojawia się, gdy rośnie liczba typów prętów finalnych, a rozmiar zamówienia jest stały. Wtedy to czas wykonania metody generacji kolumn znaczaco wzrasta. Sytuacje tę przedstawia wykres 5.17.

Na wykresach 5.18 i 5.19 przyjrzeć się można różnice w wynikach końcowych porównywanych technik. W znacznej większości przypadków to metoda generacji kolumn dawała lepsze rozwiązania, lecz nie odbiegały one istotnie od tych uzyskanych przez algorytm pakowania. Zdarzały się też przypadki, w których to algorytm pakowania zwrócił rozwiązania takie same jak generacja kolumn lub nawet lepsze.

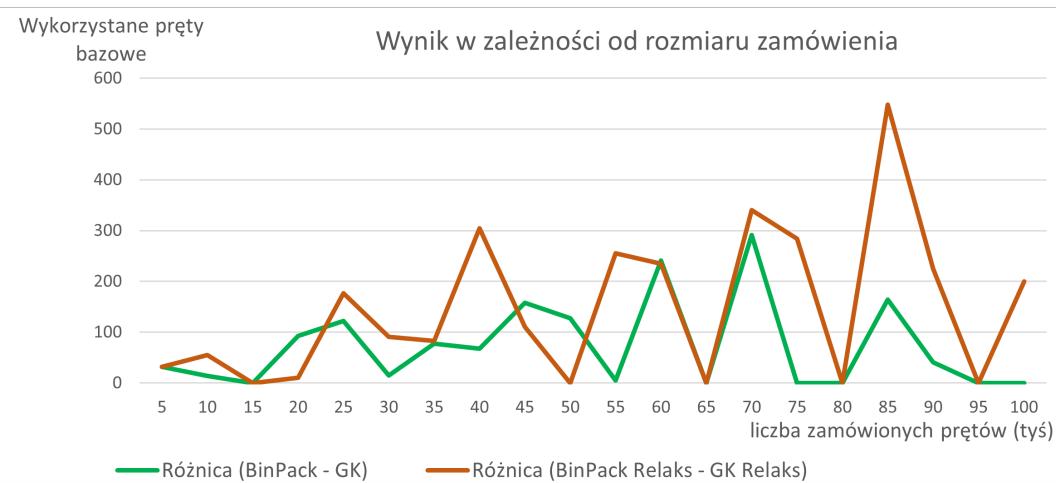
Algorytm pakowania z heurystyką First-Fit Decreasing dzięki swojej prostocie i szybkości w przypadku małych zamówień może być bardzo dobrym narzędziem do rozwiązywania problemu cięcia z jednym typem pręta bazowego. Jednakże atuty metody generacji kolumn takie jak możliwość operowania na większej liczbie prętów bazowych, niewrażliwość na wielkość zamówienia oraz większa precyzja wyników, sprawiają, że jest ona bardziej uniwersalnym i wszechstronnym rozwiązaniem.



Rysunek 5.16. Czas wykonywania algorytmów w zależności od rozmiaru zamówienia.

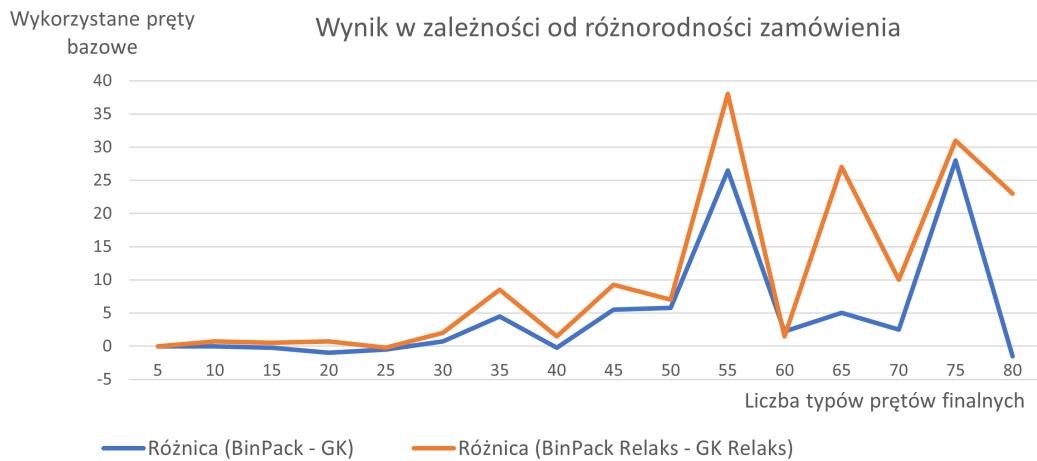


Rysunek 5.17. Czas wykonywania algorytmów w zależności od liczby typów prętów finalnych.



Rysunek 5.18. Wyniki algorytmów w zależności od rozmiaru zamówienia.

5. Testy algorytmów



Rysunek 5.19. Wyniki algorytmów w zależności od liczby typów pretów finalnych.

5.2.7. Porównanie solwerów

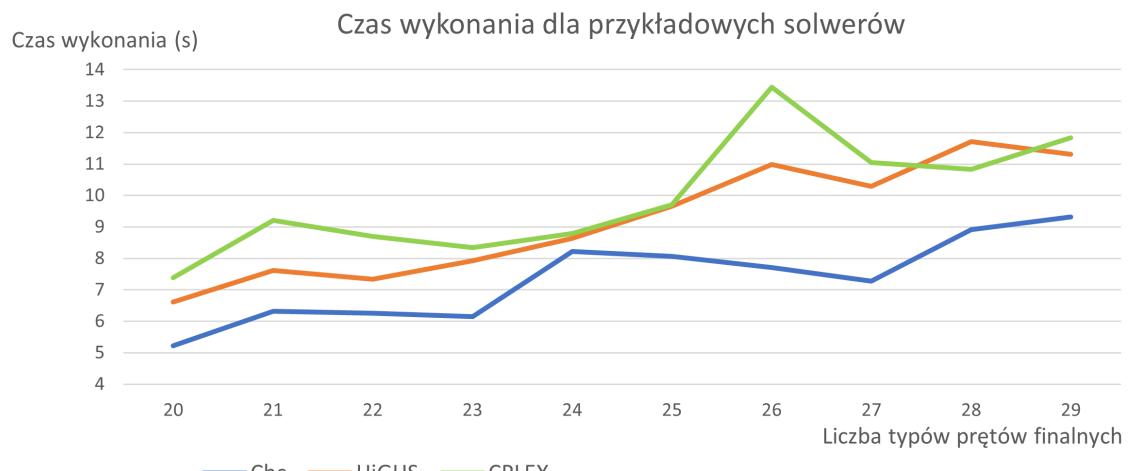
Ponieważ język AMPL pozwala w prosty sposób wybierać solwer, który rozwiązać ma dany problem, ostatni przedstawiony test w tym rozdziale będzie porównywał wyniki otrzymane przy użyciu trzech różnych solwerów dla tego samego problemu cięcia. Użyte w teście solwery to:

- Cbc (ang. Coin-or branch and cut) – otwartoźródłowy solwer, wykorzystujący standardowe techniki optymalizacyjne mieszanego programowania całkowitoliczbowego. Do rozwiązywania problemów liniowych wykorzystuje solwer Clp. Oba solwery są częścią projektu COIN-OR.
- HiGHS – nowszy od Cbc solwer otwartoźródłowy, rozwiązuje problemy liniowe, całkowitoliczbowe i kwadratowe. Często uważany w badaniach jako najszybszy solwer otwartoźródłowy [20].
- CPLEX – solwer komercyjny, obsługujący największy zakres problemów z wymienionych tu solwerów. Znany ze swojej wydajności.

Wykres 5.20 przedstawia porównanie wydajności solwerów. Jego wyniki są odwrotne z oczekiwaniemi. Okazało się, że problem cięcia najszybciej rozwiązywany był przez solwer Cbc. Jeżeli chodzi o precyzję końcowego wyniku, to różnica w niej była niemalże niezauważalna. O ile poszczególne rozwiązani różniły się nieznacznie od siebie, tak suma wykorzystanych pretów bazowych we wszystkich przypadkach testowych wynosiła odpowiednio:

1. Cbc: 51133
2. HiGHS: 51132
3. CPLEX: 51132

Warte zwrócenia uwagi jednak jest to, że pojawiały się w poprzednich testach problemy, których solwer Cbc nie był w stanie rozwiązać w zadany mu czasie (rys. 5.1) 240 sekund. Te same problemy były już rozwiązywane przez solwery HiGHS i CPLEX.



Rysunek 5.20. Porównanie wydajności solwerów

6. Wnioski

Celem niniejszej pracy magisterskiej było zbadanie algorytmów rozwiązywających problem cięcia z relaksacją. Aby tego dokonać, należało najpierw wnikliwie przeanalizować temat problemu cięcia w literaturze. Poznać najpopularniejsze metody jego rozwiązywania i wybrać jedną, która zostanie rozszerzona o możliwość relaksacji. Zdecydowano się wykorzystać metodę generacji kolumn, z uwagi na jej efektywność i duży potencjał. Następnie zaproponowano siedem autorskich modeli programowania liniowego i programowania całkowitoliczbowego, które rozwijały klasyczną metodę generacji kolumn o możliwość skracania prętów. Oferowały one minimalizację:

1. Kosztu z:
 - a) Relaksacją manualną
 - b) Relaksacją automatyczną
 - c) Relaksacją krokową
2. Odpadów z:
 - a) Relaksacją manualną
 - b) Relaksacją automatyczną
 - c) Relaksacją krokową
 - d) Relaksacją manualną bez iteracji po prętach bazowych

W ramach pracy zaimplementowana została aplikacja internetowa, tytułowy system wspierający zakup i cięcie stali. Pozwala ona jej użytkownikom definiować własne problemy cięcia i korzystać ze wszystkich opracowanych modeli programowania liniowego.

Przeprowadzono liczne testy, których celem było zweryfikowanie efektywności i wydajności modeli, a przede wszystkim wpływu relaksacji na uzyskane wyniki. Rezultaty testów są zadowalające. Wszystkie przygotowane modele mogą znaleźć zastosowanie w praktycznych problemach. Relaksacja manualna, gdy jest się w stanie określić granice relaksacji. Relaksacja automatyczna, gdy wyznaczenie granic jest trudne. Relaksacja krokowa, gdy użytkownik chce przejrzeć wszystkie dostępne możliwości.

Wszystkie zaproponowane w tej pracy rozwiązania są w pełni otwarte na dalsze możliwości rozwoju. Przykładowymi modułami, które można byłoby poddać modyfikacji, są:

1. **Aplikacja internetowa** – aplikacja może zostać rozwinięta o obsługę profili użytkowników, którzy mieliby możliwość zapisywania utworzonych problemów na swoim koncie i przeglądania historii ich rozwiązywania. Poprawie mogłyby ulec komponenty obrazujące wyniki, aby były bardziej czytelne. Więcej statystyk z procesu rozwiązywania problemu mogłyby być prezentowanych użytkownikowi.
2. **Algorytmy rozwiązywania problemu cięcia** – zastosowanie bardziej zaawansowanych technik, wspomnianych w podrozdziałach 2.2 i 2.3, pozwoliłoby przybliżyć się do optymalnych rozwiązań problemu cięcia.

3. **Rozszerzanie modeli programowania liniowego** – do modeli problemów można byłoby wprowadzić kolejne ograniczenia lub parametry, np. na szerokość piły tnącej.
4. **Metody relaksacji** – dalsze badania mogą być prowadzone nad nowymi metodami relaksacji. Duży potencjał ma relaksacja automatyczna. Zastanowić się można nad innymi sposobami obliczania kosztu relaksacji niżeli te przedstawione w tej pracy.

Udało się zrealizować wszystkie postawione we wstępie cele pracy. Idea relaksacji długości prętów, czy też innych ciętych przedmiotów, niesie ze sobą duży potencjał. Zdecydowanie należałoby skupić na tej technice uwagę w przyszłych badaniach, dla których praca ta może posłużyć jako punkt wyjścia.

Bibliografia

- [1] L. V. Kantorovich, „Mathematical Methods of Organizing and Planning Production”, *Management Science*, t. 6, s. 366–422, 1939. adr.: <https://api.semanticscholar.org/CorpusID:62611375>.
- [2] K. L. V. i Z. V. A, *Calculation of Rational Cutting of Stock*. Lenizdat, Leningrad., 1951.
- [3] R. Gilmore i R. Gomory, „A Linear Programming Approach to the Cutting Stock Problem I”, *Oper Res*, t. 9, sty. 1961. DOI: 10.1287/opre.9.6.849.
- [4] R. Silva i R. Schouery, „A Branch-and-Cut-and-Price Algorithm for Cutting Stock and Related Problems”, *Revista Eletrônica de Iniciação Científica em Computação*, t. 22, s. 31–40, czer. 2024. DOI: 10.5753/reic.2024.4646.
- [5] M. Berberler i U. Nuriyev, „A New Heuristic Algorithm for the One-Dimensional Cutting Stock Problem”, *Applied and Computational Mathematics*, t. 9, s. 19–30, sty. 2010.
- [6] H. Vasques da Silva, F. Lemos, A. Cherri i S. De Araujo, „Arc-flow formulations for the one-dimensional cutting stock problem with multiple manufacturing modes”, *RAIRO - Operations Research*, t. 57, sty. 2023. DOI: 10.1051/ro/2023001.
- [7] M. Gradišar, G. Resinović i M. Kljajić, „A hybrid approach for optimization of one-dimensional cutting”, *European Journal of Operational Research*, t. 119, nr. 3, s. 719–728, 1999, ISSN: 0377-2217. DOI: [https://doi.org/10.1016/S0377-2217\(98\)00329-4](https://doi.org/10.1016/S0377-2217(98)00329-4). adr.: <https://www.sciencedirect.com/science/article/pii/S0377221798003294>.
- [8] J. Bisschop, *AIMMS - Optimization Modeling*. Lulu.com, 2006.
- [9] J. Santos i N. Nepomuceno, „Computational Performance Evaluation of Column Generation and Generate-and-Solve Techniques for the One-Dimensional Cutting Stock Problem”, *Algorithms*, t. 15, s. 394, paź. 2022. DOI: 10.3390/a15110394.
- [10] P. Panciatici, M. Campi, S. Garatti i in., „Advanced optimization methods for power systems”, w *2014 Power Systems Computation Conference*, 2014, s. 1–18. DOI: 10.1109/PSCC.2014.7038504.
- [11] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh i P. Vance, „Branch-and-Price: Column Generation for Solving Huge Integer Programs”, *Operations Research*, t. 46, s. 33, czer. 1998. DOI: 10.1287/opre.46.3.316.
- [12] J. M. V. de Carvalho, „Exact solution of bin-packing problems using column generation and branch-and-bound”, *Ann. Oper. Res.*, t. 86, s. 629–659, 1999. adr.: <https://api.semanticscholar.org/CorpusID:39550379>.
- [13] D. S. Johnson, „Near-Optimal Bin Packing Algorithms”, 1973.
- [14] G. Dósa, „The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leq 11/9OPT(I) + 6/9$ ”, w *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, B. Chen, M. Paterson i G. Zhang, red., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 1–11, ISBN: 978-3-540-74450-4.

6. Bibliografia

- [15] J. Fang, Y. Rao, Q. Luo i J. Xu, „Solving One-Dimensional Cutting Stock Problems with the Deep Reinforcement Learning”, *Mathematics*, t. 11, s. 1028, lut. 2023. DOI: 10.3390/math11041028.
- [16] P. Rawat i A. N. Mahajan, „ReactJS: A Modern Web Development Framework”, 2020. adr.: <https://api.semanticscholar.org/CorpusID:249010550>.
- [17] J. P. Michael Mikowski, *Single Page Web Applications. Programowanie aplikacji internetowych z JavaScript*. Helion, 2015.
- [18] T. Gau i G. Wäscher, „CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem”, *European Journal of Operational Research*, t. 84, nr. 3, s. 572–579, 1995, Cutting and Packing, ISSN: 0377-2217. DOI: [https://doi.org/10.1016/0377-2217\(95\)00023-J](https://doi.org/10.1016/0377-2217(95)00023-J). adr.: <https://www.sciencedirect.com/science/article/pii/037722179500023J>.
- [19] V. Balabanov, *CUTGEN1-J*, <https://github.com/akavrt/cutgen>, 2013.
- [20] M. Parzen, J. Hall, J. Jenkins i T. Brown, „Optimization solvers: the missing link for a fully open-source energy system modelling ecosystem”, maj 2022. DOI: 10.5281/zenodo.6409432.

Spis rysunków

1.1	Problem cięcia materiału	7
1.2	Zastosowanie relaksacji	7
2.1	Model problemu cięcia w postaci sieci przepływów[6]	15
4.1	Diagram klas warstwy serwerowej	36
4.2	Diagram komponentów warstwy interfejsu użytkownika	38
4.3	Wybór algorytmu i wprowadzenie danych	39
4.4	Przesłanie problemu do backendu	40
4.5	Eksport wyników do CSV	41
4.6	Przykładowe rozwiązanie z minimalizacją kosztów	42
4.7	Przykładowe rozwiązanie z relaksacją automatyczną	43
4.8	Przykładowe rozwiązanie z relaksacją krokową	44
4.9	Wygląd aplikacji w urządzeniu mobilnym Samsung Galaxy A51	45
4.10	Uproszczony diagram klas testów.	47
5.1	Porównanie generatorów problemu cięcia	51
5.2	Efektywność różnych typów relaksacji	52
5.3	Wydajność algorytmu dla różnych typów relaksacji	53
5.4	Wydajność metody generacji kolumn w zależności od liczby prętów bazowych	53
5.5	Wydajność modelu (54-61) nie iterującego po prętach bazowych porównana z modelem klasycznym (49)	54
5.6	Wyniki relaksacji manualnej w zależności od ustawienia suwaka multiplikującego wartość kosztu relaksacji.	55
5.7	Wyniki relaksacji automatycznej w zależności od ustawienia suwaka multiplikującego wartość kosztu relaksacji.	55
5.8	Wyniki relaksacji automatycznej w zależności od maksymalnej multiplikacji kosztu relaksacji dla solwera HiGHS	56
5.9	Relaksacja automatyczna z podstawowym ustawieniem suwaka	56
5.10	Relaksacja automatyczna z multiplikatorem 1.2x	57
5.11	Relaksacja automatyczna z multiplikatorem 2x	57
5.12	Przykładowy przebieg relaksacji krokowej minimalizującej koszt.	58
5.13	Przykładowy przebieg relaksacji krokowej minimalizującej odpad.	58
5.14	Czasy wykonywania poszczególnych iteracji relaksacji krokowej.	59
5.15	Problem cięcia z możliwością magazynowania.	59
5.16	Czas wykonywania algorytmów w zależności od rozmiaru zamówienia.	61
5.17	Czas wykonywania algorytmów w zależności od liczby typów prętów finalnych.	61
5.18	Wyniki algorytmów w zależności od rozmiaru zamówienia.	61
5.19	Wyniki algorytmów w zależności od liczby typów prętów finalnych.	62
5.20	Porównanie wydajności solwerów	63