P O L I T E C H N I K A W A R S Z A W S K A

DYSCYPLINA NAUKOWA INFORMATYKA TECHNICZNA I TELEKOMUNIKACJA/
DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

Rozprawa doktorska

mgr inż. Jan Karwowski

Aproksymacja stanu równowagi Stackelberga w grach wielokrokowych o sumie niezerowej z niepełną informacją z użyciem metod Monte Carlo

Promotor
prof. dr hab. inż. Jacek Mańdziuk

WARSZAWA 2022

Podziękowania

Powstanie tej rozprawy nie byłoby możliwe bez pomocy i wsparcia wielu osób. Chciałbym podziękować wszystkim, których wkład pomógł mi w pracach badawczych i w spisywaniu wyników.

Na początku dziękuję moim rodzicom za wszystko co dla mnie zrobili, od początku (którego nie pamiętam). Szczególnie dziękuję za wsparcie ze strony materialnej. Bez tego nie byłbym w stanie poświęcić wystarczająco dużo czasu na te prace.

Dziękuję prof. Jackowi Mańdziukowi, promotorowi, za zaproponowanie kierunku badań, w obszarze który był mi nieznany, a okazał się szalenie ciekawy i głęboki, za nadzór, współpracę i pomoc w trakcie prowadzenia badań, za poprawienie niezliczonej liczby błędów językowych w przygotowywanych publikacjach.

Dziękuję doktorowi Michałowi Okulewiczowi za porady dla początkującego naukowca, stymulujące rozmowy i dyskusje na temat sensu nauki.

Dziękuję wszystkim słuchaczom Seminarium Metod Inteligencji Obliczeniowej na Wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej za wysłuchiwanie moich nie zawsze dobrze przygotowanych i często długich prezentacji, dyskusję i sugestie, szczególne podziękowania należą się tu Michałowi Okulewiczowi, Adamowi Żychowskiemu, Karolowi Walędzikowi i Maciejowi Bartoszukowi.

Dziękuję wszystkim, którzy wspomogli proces redagowania tej rozprawy poprzez czytanie wstępnej wersji tekstu i sugerowanie poprawek: Michałowi Okulewiczowi, Karolinie Okrasie, Mai Kabus.

Dziękuję studentom wydziału Matematyki i Nauk Informacyjnych, którzy, często nieświadomie, stymulowali mój umysł w momentach zastoju.

Na koniec, choć z pewnością nie jest to najmniejsze podziękowanie, chciałbym podziękować Mai, która była chyba jedyną osobą na świecie, która wierzyła, że spisanie tego dokumentu jest możliwe.

Temu, który dał wszystko.

Dziękuję.

Streszczenie

Aproksymacja stanu równowagi Stackelberga w grach wielokrokowych o sumie niezerowej z niepełną informacją z użyciem metod Monte Carlo

Rozprawa opisuje dwie metody, będące oryginalnym wynikiem, które pozwalają przybliżać strategię lidera ze stanu Równowagi Stackelberga w grach wielokrokowych o sumie niezerowej z niepełną informacją. Hipotezą badawczą stawianą w rozprawie jest twierdzenie, że możliwe jest zbudowanie efektywnej metody poszukiwania Równowagi Stackelberga w tych grach w oparciu o próbkowanie Monte Carlo.

Gry wielokrokowe, to rodzaj gier, w których gracze mają wiele punktów decyzyjnych, w każdym z tych punktów muszą wykonać akcję. W teori gier często do reprezentacji gier wielokrokowych używa się postaci ekstensywnej – drzewa, gdzie węzłami są stany gry, a krawędziami ruchy możliwe do wykonania w tych stanach. Gry z niepełną informacją charakteryzują się tym, że gracze nie mają pełnej informacji o całym stanie gry, w szczególności o działaniach przeciwnika, a mogą obserwować jedynie projekcję stanu udostępniającą wybrane informacje. Dodatkowo, w tej rozprawie rozważane są tylko gry z doskonałą pamięcią, to znaczy takie, gdzie udostępniana graczowi informacja zawsze uwzględnia rozróżnienie stanów na podstawie wszystkiego co gracz dotąd zaobserwował, w tym wykonanych przez niego akcji. Suma niezerowa oznacza, że w grze dla dwóch graczy wypłaty otrzymane na koniec gry przez graczy nie muszą sumować się do zera.

Równowaga Stackelberga jest pojęciem z obszaru teorii gier. W Grze Stackelberga bierze udział dwóch asymetrycznych graczy, lider i naśladowca. Lider wybiera strategię mieszaną jako pierwszy, następnie upublicznia ją. Naśladowca wybiera swoją strategię znając już strategię lidera. Model Stackelberga zakłada pełną racjonalność naśladowcy, czyli zachowanie gdzie naśladowca zawsze wybierze strategię, która daje mu najlepszą możliwą wypłatę. Stanem Równowagi Stackelberga nazywamy układ strategii lidera i naśladowcy, gdzie strategia naśladowcy jest optymalną odpowiedzią na strategię lidera, a strategia lidera daje liderowi najwyższą możliwą wypłatę spośród wszystkich układów strategii spełniających warunek optymalnej odpowiedzi naśladowcy. Rozprawa wskazuje pozycje w literaturze, które mówią o praktycznym zastosowaniu Równowagi Stackelberga w sytuacjach związanych z interakcją pomiędzy siłami bezpieczeństwa (np. policją, strażą graniczną), a łamiącymi prawo (przemytnikami, terrorystami).

Po zdefiniowaniu potrzebnych pojęć z teorii gier zaprezentowane są istniejące w literaturze podejścia do obliczeniowego wyznaczania Równowagi Stackelberga. Duża część z istniejących w literaturze podejść jest dedykowana bardzo szczególnym podklasom wielokrokowych Gier Stackelberga o sumie niezerowej z niepełną informacją. Są to metody, które wykorzystują specyficzne cechy w strukturze gry, aby znacznie przyspieszyć obliczenia. Tych metod nie da się uogólnić na całą klasę gier rozważaną w rozprawie. Zaprezentowane są również metody z literatury dedykowane całej wspomnianej klasie gier. Wszystkie prezentowane metody, zarówno te specyficzne dla danej klasy, jak i ogólne wykorzystują programowanie liniowe jako istotny

element rozwiązania. Część metod konstruuje program liniowy, który wylicza stan równowagi, w przypadku części metod wyliczanych jest wiele programów liniowych, a sama strategia jest uzyskiwana z pomocą fragmentów działających poza programowaniem liniowym. Analiza metod z literatury wskazuje powtarzające się elementy, które można wykorzystać przy budowie nowych metod rozwiązujących Gry Stackelberga. Są to: technika przeglądania wszystkich strategii naśladowcy i dobierania do każdej z nich strategii lidera, metoda generowania kolumn, metoda podwójnej wyroczni. Oprócz samych metod wskazany jest też zbiór Search Games wykorzystywany przez niektóre prace opisujące te metody, który można wykorzystać do ewaluacji eksperymentalnej metod rozwiązujących Gry Stackelberga.

Następnie prezentowany jest główny wkład autora rozprawy w dziedzinę. Pierwszym elementem jest rodzina gier z niepełną informacją rozgrywanych na grafach, która został wykorzystana do eksperymentalnej ewaluacji metod. W skład tej rodziny wchodzą trzy różne zbiory gier testowych. Drugim elementem jest opis dwóch metod do poszukiwania strategii lidera, która będzie dobrym przybliżeniem strategii lidera z Równowagi Stackelberga. Obie proponowane metody wykorzystują popularną metodę rozwiązywania gier z pełną informacją, na przykład gier planszowych, nazywaną Upper Confidence Bound applied to Trees (UCT). Metoda UCT w trakcie swojego działania buduje drzewo statystyk na temat ruchów, nazywane drzewem UCT. Pierwsza z metod, nazwana Mixed-UCT opiera się o wielokrotne uruchamianie metody UCT w grze dla jednego gracza, gdzie lider wybiera swoje ruchy, a ruchy naśladowcy pochodzą z wcześniej ustalonej strategii. Następnie statystyki zebrane w ten sposób w drzewie UCT są przekształcane w strategię mieszaną lidera. Ten proces jest powtarzany iteracyjnie, a po każdym uzyskaniu strategii lidera, aktualizowana jest ustalona strategia naśladowcy, przeciwko której rozgrywane są symulacje UCT, tak aby uwzględnić fakt, że strategia naśladowcy powinna być najlepszą odpowiedzią na strategię lidera. Metoda Mixed-UCT jest następnie uruchomiona na zbiorze testowym gier, które są bliskie sumie zerowej i porównana z metodami z literatury, które da się stosować do ogólnej klasy gier. Wyniki tych eksperymentów pokazują, ze Mixed-UCT jest znacznie szybsza od metod z literatury dla dużych gier testowych, a otrzymane strategie są tylko nieznacznie gorsze od strategii optymalnych. Potrzebuje też dużo mniej pamięci operacyjnej. Niestety Mixed-UCT nie działa zbyt dobrze dla gier o sumie dalszej od sumy zerowej. Druga metoda, nazwana O2-UCT, nie stosuje podejścia UCT bezpośrednio. Główna zasada działania tej metody to wielokrotne próbkowanie strategii naśladowcy, a następnie dobieranie do niej strategii lidera tak, aby spełniony był warunek, że strategia naśladowcy jest najlepszą odpowiedzią na tę strategię lidera, a w drugiej kolejności, żeby wypłata lidera była możliwie duża. W tej metodzie podejście UCT wykorzystywane jest do ukierunkowanego próbkowania strategii naśladowcy tak, aby preferować strategie dla których da się budować strategię lidera o dużej wypłacie. Sama metoda dobierania strategii lidera opiera się o koncepcję podwójnej wyroczni. Naprzemiennie poprawiana jest strategia lidera i poszukiwana jest najlepsza odpowiedź naśladowcy. W zależności od tego jaka odpowiedź została znaleziona, zmienia się kierunek poprawy strategii lidera. Metoda O2-UCT została przetestowana eksperymentalnie

na trzech zbiorach gier testowych i porównana z metodami z literatury. Dla wszystkich zbiorów testowych, dla dużych instancji gier testowych O2-UCT jest szybsza od metod z literatury. Wartości wypłat uzyskiwane przez strategie są bardzo bliskie optymalnym dla wszystkich gier, dla których udało się policzyć rozwiązania dokładne.

Wyniki eksperymentalne badające zaproponowane w tej rozprawie metody potwierdzają hipotezę badawczą postawioną w rozprawie.

Słowa kluczowe: Równowaga Stackelberga, UCT, MCTS

Abstract

Monte Carlo methods for approximation of Stackelberg Equilibium in sequential multi-act general sum games with imperfect information

This thesis describes two original methods that approximate leader's strategy of a Stackelberg Equilibrium in multi-act non-zero sum games with imperfect information. The research hypothesis checked in this thesis is that it is possible to construct a method based on Monte Carlo sampling that seeks Stackelberg Equilibrium in such games.

Multi-act games are a type of games where players encounter multiple decision points in the game and they have to act multiple times, once in every point. The popular representation of such games in game theory is extensive game — a tree where each node is a game state and arcs between nodes are actions possible to play in the given state. Imperfect information is a property that means that players in the game do not see the whole state of the game, in particular they may have limited information about opponents' actions. They can see only a projection of the state which reveals some information. In this is only games that satisfy additional property called perfect recall are considered. Perfect recall means that player can always tell the difference between two states where previous observations and actions of that player were different from each other. Non-zero sum game means that in two player game players' payoffs do not necessary sum up to 0.

Stackelberg Equilibrium is a game-theoretic concept. Stackelberg Game is played by two asymmetric players, the leader and the follower. The leader commits to a mixed strategy first and makes it public. The follower chooses a strategy already knowing leader's commitment. The assumption is that the follower is perfectly rational and chooses the strategy maximizing their payoff. The strategy profile is called Stackelberg Equilibrium where the follower's strategy is the best response to the leader's strategy and the profile maximizes leader's utility across all profiles satisfying the best response restriction. The literature review in the thesis cites papers that describe practical applications of Stackelberg Equilibrium in scenarios where interactions between law enforcement (police, border guards) and criminals (smugglers, terrorists).

After defining all required game-theoretic terms the thesis presents existing approaches do finding Stackelberg Equilibrium found in the literature. Majority of existing methods is dedicated to specific subclasses fo Stackelberg non-zero sum games with imperfect information. Those methods exploit specific properties of a game structure in given class to speed up computation. It is not possible to generalize such approaches to general class of games. Methods that are applicable to broad class of all games with the mentioned properties are also presented. All the presented methods, both game-specific and general ones employ linear programming as a base of the solutions. Some of the methods are composed of a linear program that defines the equilibrium and simply solve it, other methods solve many linear programs and use some additional procedures to obtain the final solution. During the analysis of the existing methods some recurring elements were noted, as they might be useful when building new methods for Stackelberg Equilibrium. Those include: browsing all follower's strategies and for each of the

strategies calculating a matching leader's strategy, a column generation method and a double-oracle method. Besides the methods a set of games called Search Games is also noted as it is used by several papers introducing new methods and it might be used to evaluate the newly constructed methods.

Next, the main author's contribution to the field is presented. The first part is a class of imperfect-information games played on graphs which is later used in experimental evaluation of the proposed methods. Within that class three separate sets of games were constructed. The second part is the description of the two new methods for seeking a leader's strategy that approximates the leader's strategy in Stackelberg Equilibrium. Both methods employ a popular approach to solving perfect information games, for instance board games, called Upper Confidence Bound applied to Trees (UCT). The UCT method, during execution, builds a tree that gathers statistics about moves in the game, called UCT tree. The first proposed method called Mixed-UCT, comprises running UCT to find moves in a single player game where the leader needs to choose which moves and the follower moves are sampled from some provided strategy. The statistics gathered in the UCT tree are then transformed into leader's mixed strategy. The process is repeated and after each repetition the follower's strategy which is used for game in UCT simulations is updated to accommodate the follower's best response for the current leader's strategy. Mixed-UCT is then evaluated on a test set containing games that are close to zero-sum games and the results are compared to the results of existing methods dedicated to the general class of games. The results show that Mixed-UCT is significantly faster than the existing methods for large instances of test games. The calculated strategies are only slightly worse than the optimal ones. It also needs less memory to perform calculations. Mixed-UCT does not perform well on games that are further from zero-sum. The second method, called O2-UCT, does not use UCT to directly find the leader's strategy. The main principle of the method is to sample follower's strategies and then find a leader's strategy such that the follower's strategy is the best response to it and, among all strategies satisfying that condition, try to find the strategy that maximizes leader's payoff. The UCT method is used to perform guided sampling of follower's strategies to process in a way that the strategies for which with leader's payoff was obtained are preferred. The method used to find the leader's strategy is based on double-oracle approach. In alternating fashion a leader's strategy is updated and the best follower's response is found. Based on which strategy is the best response the update direction of leader's strategy is chosen accordingly. The O2-UCT method was tested on three game sets and the results were compared to the existing methods. For all three game sets O2-UCT was the fastest for the large games instances. The payoff values obtained were very close to optimal ones in all cases where the optimal solutions were possible to calculate.

The experimental results that assess the proposed methods confirm the hypothesis stated in the beginning.

Keywords: Stackelberg Equilirbium, UCT, MCTS

Spis treści

Rozdział 1

Wprowadzenie

Rozprawa dotyczy aproksymacji Równowagi Stackelberga w grach wielokrokowych o sumie niezerowej z niepełną informacją i doskonałą pamięcią. Metody zaproponowane w tej rozprawie są uniwersalne, wymagają bardzo niewielu założeń co do struktury rozwiązywanej gry. Poszukiwanie Równowagi Stackelberga w wielu klasach gier, w tym w grach rozważanych w tej rozprawie jest problemem NP-trudnym. To motywuje potrzebę budowy metod heurystycznych, które będą znajdować dobre (choć często suboptymalne) rozwiązania w akceptowalnym czasie. Najważniejszym wynikiem rozprawy są dwie metody zbudowane na bazie heurystycznych metod przeszukiwania przestrzeni strategii graczy, które pozwalają znajdować strategie lidera dające wyniki bliskie strategiom optymalnym w Grach Stackelberga. Główną metodą badawczą stosowana w rozprawie jest eksperyment. W szczególności analiza efektywności proponowanych metod została przeprowadzona na odpowiednio przygotowanych zbiorach gier testowych. Badany obszar znajduje się na styku trzech pól: teorii gier, sztucznej inteligencji i optymalizacji matematycznej, przy czym metody proponowane w tej rozprawie odchodzą od trzeciego obszaru, technik optymalizacji matematycznej, które są stosowane w rozwiązaniach proponowanych w literaturze, na rzecz metod heurystycznych.

1.1 Teoria gier i idea Równowagi Stackelberga

Z matematycznego punktu widzenia, w najprostszym ujęciu, gra to zbiór zasad definiujący możliwe akcje i ich następstwa dla jednego lub więcej graczy – jednostek podejmujących decyzje. Działania wszystkich jednostek (graczy) są wykonywane w jednym środowisku i mogą ze sobą nawzajem oddziaływać. Dodatkowo, gra może być wzbogacona o element losowości, który daje efekty według znanego wszystkim graczom rozkładu prawdopodobieństwa. Sekwencja decyzji podjętych przez wszystkich graczy razem z wartościami ewentualnych zmiennych losowych definiują rozgrywkę, rozgrywka natomiast jest powiązana z wynikiem, nagrodą lub karą, dla każdego z graczy. Wynik gry każdego z graczy zwykle jest liczbą rzeczywistą. Z praktycznego punktu widzenia gra może być użyta jako model sytuacji rzeczywistych, gdy mamy do czynie-

nia z osobami podejmującymi decyzje i chcemy maksymalizować zyski wynikające z podjętych decyzji.

Teoria gier [65] jest obszarem badań, który zajmuje się budową i klasyfikacją modeli gier i definiowaniem w tym modelach stanów równowagi. Każda definicja stanu równowagi zawiera założenia dotyczące racjonalności i wiedzy graczy i wskazuje jakie układy zachowań graczy mogą wystąpić przy tych założeniach. Początkowo teoria gier służyła modelowaniu oddziaływań w ekonomii, jednak szybko znalazła zastosowanie również w innych dziedzinach. Jeszcze w XIX wieku Cournot analizował zachowania rynków i decyzje przedsiębiorstw z użyciem narzędzi matematycznych [25], natomiast abstrakcyjne opisy gier zaczęły się pojawiać w pierwszej połowie XX wieku, głównie za sprawą von Neumanna. Postęp w budowie aparatu teoretycznego został podsumowany w książce von Neumanna i Morgersterna [65], wydanej po raz pierwszy w 1944 roku. Dzięki modelowaniu na abstrakcyjnym poziomie akcji i stanów, a nie bytów związanych z konkretnym zastosowaniem, inne obszary szybko zaczęły wykorzystywać te narzędzia. Przykładem mogą być nauki polityczne i społeczne [19, 30], czy bezpieczeństwo publiczne [79, 37, 80]. Teoria gier jest też stosowana jako narzędzie wspomagające w obszarze sztucznej inteligencji, w systemach wieloagentowcyh [67], gdzie służy do wyliczania optymalnych zachowań poszczególnych agentów. Mnogość zastosowań powoduje, że rozwiązywanie problemów stawianych przez teorię gier, przyczynia się do postępu w wielu obszarach nauki. Co za tym idzie twierdzenia i algorytmy rozwijane w tym obszarze należy traktować jako uniwersalnie użyteczne.

Przykładem prostej gry jest sytuacja, gdzie w punkcie $(0,0)$ nieskończonej płaszczyzny stoi skrzynka. Dwie osoby chcą tę skrzynkę przesunąć. Jedna chce, aby skrzynka znalazła się możliwie daleko w kierunku osi x, a położenie y nie ma dla niej znaczenia. Druga osoba ma dokładnie odwrotne wymagania, to znaczy nie ma dla niej znaczenia położenie w osi x, ale chce, żeby przesunięcie wzdłuż osi y było możliwie duże. Będziemy nazywać tych graczy odpowiednio x i y. Obie osoby są wstanie ciągnąć skrzynkę z tą samą siłą, przez ten sam czas τ. Dla uproszczenia przyjmijmy, że ta siła powoduje przesuwanie się skrzynki z prędkością v. Każda z osób musi zdecydować, w którą stronę pociągnie skrzynkę. Modelując taką sytuację narzędziami teorii gier, każdemu z dwóch graczy przypisalibyśmy przestrzeń ruchów — przedział $[0,2\pi]$ reprezentujący kierunek (kąt) w jakim dany gracz pociągnie skrzynkę. Dla ustalenia uwagi przyjmijmy, że kąt 0 pokrywa się z kierunkiem osi x i rośnie przeciwnie do ruchu wskazówek zegara. Łatwo można pokazać, że, jeśli obaj gracze zdecydują się na kąt $\pi/4$, to przesunięcie rzutowane na każdą z osi wyniesie tyle samo, czyli $\sqrt{2}v\tau$. Jest to jednocześnie sytuacja maksymalizująca sumę przesunięć w osi x i y. Taka sytuacja może być stabilnym zachowaniem w modelu, pod warunkiem, że gracze ze sobą współpracują. Tak jednak być nie musi. Jeśli gracz x zdecyduje się wykorzystać fakt, że y chce współpracować i wybierze kąt 0, wtedy przesunięcie w kierunku x wzrośnie, kosztem spadku w kierunku y. Zatem, jeśli nie wiadomo, czy przeciwnik będzie współpracował, należałoby wybrać kąt odpowiednio 0, w celu maksymalizacji wartości x i $\pi/2$ w celu maksymalizacji y. Wtedy wynik każdego z graczy wyniesie $v\tau$.

Teoria gier dostarcza szerokie spektrum modeli, które można wykorzystać do odwzorowania sytuacji rzeczywistych. Często podjęcie decyzji jednorazowo, jak w przykładzie powyżej, może nie być wystarczające. W związku z tym rozważane są też gry wielokrokowe, gdzie gracze wykonują ruchy, obserwują ich efekty, a następnie wybierają kolejne ruchy, dysponując tą dodatkową wiedzą. Część takich gier zawiera również element niepełnej informacji. Niepełna informacja polega na tym, że gracze, obserwując sytuację, nie wiedzą wszystkiego, co definiuje stan gry. W przypadku gier karcianych mogą to być na przykład karty na ręce przeciwnika. Gracz wie ile kart ma przeciwnik, ale nie wie, jakie to karty. Te dwa elementy są bardzo przydatne w modelowaniu rzeczywistych interakcji między ludźmi lub systemami, gdzie wraz z postępem rozgrywki zyskujemy większą wiedzę o przeciwniku.

Równowaga Stackelberga [81, 54], którą przybliżają metody obliczeniowe prezentowane w tej rozprawie, w podstawowej wersji opisuje sytuację w grze dla dwóch graczy, gdzie jeden z graczy ma rolę lidera, a drugi naśladowcy. W porównaniu do stanu równowagi Nasha, gdzie stan równowagi definiowany jest symetrycznie dla wszystkich graczy, w przypadku Równowagi Stackelberga mamy do czynienia z asymetrią. Lider decyduje się na swoją strategię (zazwyczaj mieszaną) jako pierwszy i podaje ją do publicznej wiadomości. W tym momencie nie może już tej decyzji zmienić. Naśladowca podejmuje swoją decyzję znając już strategię lidera. Taka konstrukcja stanu równowagi była motywowana przez Stackelberga sytuacją, gdy jeden podmiot ma dominujący udział w rynku (lider) i decyzje przez niego podjęte są następnie obserwowane przez pozostałe podmioty (Naśladowców), które dostosowują się do niego.

W praktyce potrzebna jest budowa metod, które będą efektywnie znajdować układy strategii spełniające założenia konkretnych stanów równowagi. Dla niedużych, jednokrokowych gier często wystarczające są metody oparte o programowanie liniowe. Istotną zaletą takich metod jest fakt, że prosty sposób można zamienić definicję stanu równowagi na zbiór ograniczeń programu liniowego i udowodnić, że proponowany program faktycznie znajduje założony stan równowagi. Metody takie nie skalują się dobrze wraz z rozmiarem gry i w związku z tym rozwój zastosowań w dziedzinie jest ściśle uzależniony od rozwoju efektywnych metod pozwalających znajdować stany równowagi w rozbudowanych grach modelujących rzeczywiste zagadnienia. W przypadku Równowagi Stackelberga zostało udowodnione, że o ile dla gier jednokrokowych znalezienie Równowagi Stackelberga jest problemem wielomianowym, o tyle dla gier wielokrokowych z niepełną informacją problem poszukiwania Równowagi Stackelberga jest NP-trudny [55]. Omówienie trudności poszukiwania tej równowagi znajduje się w Rozdziale 2.6 niniejszej rozprawy. W dalszej części będzie stosowane określenie Gra Stackelberga oznaczające grę, w której poszukujemy Równowagi Stackelberga. W związku z tym, interesujące są sposoby znajdowania strategii przybliżających Równowagę Stackelberga.

Zastosowania Gier Stackelberga. Równowaga Stackelberga, podobnie jak wiele innych pojęć teorii gier, początkowo została zdefiniowana na użytek ekonomii i miała modelować sytuację duopolu, w którym jedna z firm podejmuje decyzje pierwsza. Duopol jest systemem,

gdzie na rynku obecne są dwie firmy, a bariera wejścia jest na tyle duża, że nie pojawiają się inni konkurenci. W przypadku rozważanym przez Stackelberga [81], jedna z firm decyduje się na produkcję i wypuszczenie na rynek danej ilości produktu oraz ustala jego cenę, wiedząc, że konkurencja dowie się jaka jest to ilość i dobierze swój plan produkcji tak, aby możliwie wykorzystać decyzję pierwszej firmy na swoją korzyść.

W późniejszym okresie stan Równowagi Stackelberga zyskał również popularność w innych obszarach. Szczególną rolę odegrał w zastosowaniach związanych z zabezpieczaniem, patrolowaniem i planowaniem kontroli, gdzie występuje silna asymetria między siłami obrońców, którzy działają w sposób ciągły i których zachowanie może być obserwowane oraz przestępcami, którzy mogą przygotowywać się do swoich działań prowadząc obserwację.

Pierwszym podejściem do użycia Równowagi Stackelberga w problemach tego typu jest praca z 1966, w której Maschler proponuje zastosowanie modelu bazującego na Równowadze Stackelberga do Gry Inspektora (The Inspector's Game) wprowadzonej w pracy [61]. Model ten opisuje sytuację, gdy pewna jednostka zdecydowała się podpisać umowę o przestrzeganiu pewnych zasad (na przykład, gdy państwo zdecydowało się podpisać układ pokojowy, w którym zobowiązuje się rozbroić część swojego wojska). Gra jest rozgrywana między dwiema stronami: inspektorem i organizacją, która potencjalnie chce naruszyć zasady układu. Inspektor wykonuje sekwencję kontroli, w trakcie których może zaobserwować jedno lub więcej zdefiniowanych w ramach gry zdarzeń niepokojących. W przypadku zaobserwowania takich zdarzeń inspektor może podjąć dalsze kroki i odkryć niezgodności z podpisanym paktem albo stwierdzić, że był to fałszywy alarm i wszystko jest w porządku. Gra ma strukturę niezerowej sumy, ponieważ niezależnie od tego czy alarm był fałszywy, czy nie, inspektor ponosi koszty, a strata drugiej strony powstaje tylko, gdy faktycznie istniały nieprawidłowości. Autor pracy proponuje właśnie Równowagę Stackelberga jako model optymalnych zachowań graczy. Co więcej, w ramach szerokiej analizy teoretycznej, praca przedstawia Twierdzenie 6.1, którego konsekwencją jest stwierdzenie, że w przypadku przedstawionej w pracy gry fakt ogłaszania strategii przez lidera (inspektora) może tylko poprawić wynik gracza w porównaniu z sytuacją, gdy strategia nie jest ujawniona. To zjawisko pokazuje jedną z ciekawych cech Równowagi Stackelberga: ujawnianie przez lidera jego strategii może być intuicyjnie odbierane jako utrudnienie, ale w niektórych modelach gier siła sprawcza lidera polegająca na narzuceniu konkretnej sytuacji naśladowcy może się okazać dużo większą korzyścią niż nieujawnianie strategii. Gry wzorowane na tym modelu lub rozszerzające go były umieszczane również w konkretnych obszarach, na przykład inspekcji Międzynarodowej Agencji Energii Atomowej (IAEA), weryfikującej wykorzystanie materiałów promieniotwórczych tylko do pozyskiwania energii [7]. Należy zwrócić uwagę na fakt, że w starszych pracach często używane są terminy Leadership Equilirbium i Leader Commitment do opisu sytuacji definiowanej przez Równowagę Stakelberga, gdzie lider podejmuje decyzję o strategii, a naśladowca zna tę strategię przed podjęciem decyzji.

Innym obszarem, nie wywodzącym się bezpośrednio z teorii gier, jest obszar Przerywania Sieci (ang. Network Interditicion, czasami też Graph Interdiction) [80]. Jest to szeroki obszar

modelowania oddziaływania dwóch przeciwstawnych sił, w którym jedna strona chce znaleźć optymalną drogę w sieci (grafie) połączeń, a druga strona chce zablokować możliwość zrealizowania celów pierwszej strony. Przykładem są działania straży granicznej, która próbuje uniemożliwić przemytnikom znalezienie drogi przez granicę, bądź po prostu przez patrolowanie obszaru [93], bądź poprzez stosowanie specjalistycznego sprzętu, wykrywającego np. materiały radioaktywne [63]. W innym wariancie, jedna ze stron zarządza siecią połączeń, a druga próbuje te połączenia zniszczyć, na przykład zarządca infrastruktury przesyłu prądu przeciwko świadomemu sabotażyście lub katastrofom naturalnym [26]. Podobny mechanizm został zastosowany w algorytmie mającym za zadanie kontrolować pożary poprzez wysyłanie jednostek straży pożarnej w kierunkach najbardziej intensywnego rozprzestrzeniania się ognia [70, 71]. Do definiowania optymalnych zachowań aktorów w tym obszarze często używana jest Równowaga Stackelberga. Rozważane problemy mogą mieć zarówno naturę statyczną, gdy chcemy zablokować jednorazową próbę ataku poprzedzoną dobrym rozpoznaniem [72], jak i sekwencyjną, gdzie wraz z postępem rozgrywki przeciwnik zyskuje częściową wiedzę o naszych działaniach obronnych [14, 15]. Jednym z ciekawszych modeli, jest praca, gdzie program budowy bomby atomowej przez wrogie państwo jest opisany skierowanym grafem acyklicznym, a celem jest uderzenie w proces w tym grafie, który spowoduje możliwie duże opóźnienie w konstrukcji bomby [22]. Prace w tym obszarze, w tym prace wymienione powyżej, korzystają z metod silnie wykorzystujących strukturę problemu w postaci sieci do znalezienia Równowagi Stackelberga.

W ostatnich latach pojawił się obszar Gier Obronnych (ang. Security Games) [79], który zajmuje się używaniem narzędzi teorii gier wprost do modelowania zagadnień bezpieczeństwa narodowego i zabezpieczania obiektów użyteczności publicznej. Obszar ten pojawił się wraz ze wzrostem wydatków na obronność po zamachach 11. września 2001. Gry Obronne są rozwijane przede wszystkim przez zespół Teamcore pod kierunkiem Milinda Tambe. Metody budowane w tym obszarze często korzystają z rozwiązań opartych o Równowagę Stackelberga i często są nazywane Grami Obronnymi Stackelberga (ang. Stackelberg Security Games). Za sukces tego obszaru odpowiada przede wszystkim szereg wdrożonych rozwiązań. Najważniejsze wdrożenia obejmują: planowanie patroli służby US Federal Air Marshals [48], planowanie punktów kontrolnych na lotnisku w Los Angeles (IATA LAX) [37], planowanie tras patroli straży wybrzeża (US Coast Guard) [1]. Poza zagadnieniami związanymi z bezpieczeństwem, również z użyciem narzędzi teorii gier zostały wdrożone metody przeciwdziałania kłusownictwu [31], czy planowanie tras kontrolerów biletów w komunikacji miejskiej [92]. Publikacje w obszarze Gier Obronnych obejmują też metody, które w chwili obecnej nie zostały wdrożone, ale proponują metodę działającą w grach modelujących zagadnienie wzorowane na rzeczywistym, na przykład ochrona tankowców przed atakami piratów [90], zapobieganie nielegalnym wycinkom lasów [38], prewencyjne patrolowanie ulic miejskich [86] czy zabezpieczenie imprez masowych [91]. Badania w obszarze Gier Obronnych są skupione na budowaniu rozwiązań skutecznych w modelu konkretnego zagadnienia. Prace publikowane w tym obszarze stanowią dobry przegląd potencjalnych technik i mechanizmów pozwalających przyspieszyć obliczenie

Równowagi Stackelberga w grach modelujących zadane zastosowanie. Część z tych mechanizmów można również próbować wykorzystywać przy budowie metod ogólnych, część jest ściśle związana ze strukturą danej podklasy gier. Jednym z takich mechanizmów jest dekompozycja – obserwacja, że stany i ruchy w grach modelujących rzeczywiste zagadnienia składają się z pewnych składowych, które można rozważać niezależnie. Na przykład w przypadku punktów kontrolnych w obrębie lotniska każdy patrol może być wyposażony w różne rodzaje sprzętu skanującego. Obszary chronione przez patrole mogą się pokrywać. W efekcie, zamiast rozważać przypisania do konkretnych punktów kontrolnych, możemy analizować pokrycie obszarów, co znacznie zmniejsza przestrzeń ruchów w grze. Ponadto w większości prac z tego obszaru rozważane gry są jednokrokowe. Techniki i uproszczenia stosowane w Grach Obronnych są przedyskutowane w Rozdziale 3 niniejszej rozprawy.

Mimo stosowanych uproszczeń, to właśnie te prace przyczyniły się do wzrostu zainteresowania Równowagą Stackelberga w ostatnich 10 latach. Pojawiły się prace dotyczące strategii patrolowania i poszukiwania używające Równowagi Stackelberga do modelowania zachowań graczy [9, 8]. W ostatnich 5 latach zaczęły się pojawiać prace dotyczące obliczania Równowagi Stackelberga w ogólnych klasach gier, nie specjalizowane do żadnych zastosowań [18, 17, 16, 87, 95]. Z punktu widzenia algorytmów proponowanych w tej rozprawie, to właśnie te ostatnie metody są najbardziej istotne, ponieważ działają w ogólnych klasach gier, w tym niektóre z nich w wielokrokowych grach o sumie niezerowej. W Rozdziale 4 zawarte są porównania metod proponowanych w rozprawie z przytoczonymi metodami z literatury.

Motywacja konieczności budowy lepszych metod. Podsumowując aktualny stan literatury związany z Grami Stackelberga, można zaobserwować, że ogromna większość prac skupia się na grach jednokrokowych, w szczególności cechę tę mają wszystkie wdrożenia w Grach Obronnych Stackelberga. Tylko niewielka liczba prac rozważa gry wielokrokowe. Przyczyną tej sytuacji jest fakt, że w literaturze jest bardzo niewiele metod dedykowanych grom wielokrokowym i będących w stanie wykorzystać ich strukturę w problemie poszukiwania Równowagi Stackelberga. W związku z tym konieczne jest zaproponowanie nowych podejść, które umożliwią praktyczne rozwiązywanie takich modeli i pozwolą na ich wdrożenie.

1.2 Metody Monte Carlo i inne metaheurystyki

Informatyka jest stosunkowo młodą dyscypliną naukową, która zaczęła się dynamicznie rozwijać od drugiej połowy XX wieku. Obszarem, który od początku towarzyszył rozwojowi informatyki,jest sztuczna inteligencja. Celem badań w tym obszarze jest budowa systemów komputerowych, które będą w stanie uczyć się i rozwiązywać zadania, bez wcześniejszego zaprogramowania pod kątem konkretnego problemu. XXI wiek przyniósł bardzo szeroki rozwój sztucznej inteligencji, wraz ze wzrostem dostępnej mocy obliczeniowej coraz więcej zadań może być rozwiązywanych przez komputery.

W przypadku sztucznej inteligencji stosowanej do gier planszowych i karcianych popularnym podejściem jest Monte Carlo Tree Search (MCTS) [23], które opiera się na losowym przeszukiwaniu drzewa stanów gry i w kolejnych próbkowaniach stopniowym przechodzeniu od losowego przeszukiwania do przeszukiwania preferującego okolice najlepszych znalezionych rozwiązań wykorzystując informacje zebrane podczas wcześniejszych próbkowań. Podejście okazało się skuteczne w wielu grach, jak na przykład grze planszowej Havannah [85] czy brydżu [33]. Poza grami, MCTS okazał się również skutecznym podejściem w innych problemach związanych z podejmowaniem decyzji, między innymi planowaniu zadań w projekcie, w warunkach niedeterminizmu [89], jak i bardziej ogólnie w rozwiązywaniu decyzyjnych procesów Markova [76]. Najbardziej znaczącym sukcesem metod bazujących na MCTS w ostatnich latach jest seria podejść zapoczątkowanych przez Alpha Go [77], która była w stanie jako pierwsza pokonać graczy Go na poziomie mistrzowskim. W kolejnych latach zostały opublikowane prace rozszerzające tę koncepcję, między innymi Alpha Go Zero [78], będąca rozwinięciem Alpha Go, która względem swojej poprzedniczki nie wykorzystuje zapisów historycznych partii w procesie nauki i najnowsza, MuZero [74], która jest w stanie, bez wiedzy ekspertów uczyć się grać w różne gry, nie tylko Go.

Sukcesy metod MCTS w obszarze gier motywują wykorzystanie właśnie tego podejścia jako bazy metod proponowanych w rozprawie.

1.3 Cel i zakres rozprawy

Celem rozprawy jest budowa heurystycznych metod do aproksymacji Równowagi Stackelberga w grach wielokrokowych z niepełną informacją o sumie niezerowej.

Przez aproksymację stanu Równowagi Stackelberga rozumiemy poszukiwanie strategii lidera, która przy założeniu, że naśladowca zagra optymalną odpowiedź, daje liderowi wynik bliski wynikowi w stanie równowagi. Formalne definicje Równowagi Stackelberga, klasy gier z niepełną informacją o sumie niezerowej i postawionego problemu optymalizacyjnego przedstawione są w Rozdziale 2 rozprawy. Tak postawiony problem poszukiwania Równowagi Stackelberga we wspomnianej klasie gier jest problemem NP-trudnym [55]. Skupienie się na strategii lidera wynika z faktu, że obliczania strategii naśladowcy jest łatwe względem ustalonej strategii lidera. Wyliczenie strategii lidera w większości przypadków jest trudne.

Problem budowy metod ogólnych wpisuje się w nurt badań podstawowych. Przedstawione w tej rozprawie wyniki badań są odniesione jedynie do zbiorów gier testowych i nie były przygotowywane pod kątem wdrożenia. W opinii autora rozprawy budowa takich metod jest niezbędna, aby w przyszłości umożliwić wykorzystanie dużo bardziej złożonych, a zatem i bardziej precyzyjnych modeli w rzeczywistych sytuacjach.

Główną częścią rozprawy jest projektowanie i budowa dwóch autorskich metod heurystycznych, które z użyciem metaheurystyk bazujących na przeszukiwaniu Monte Carlo są w stanie znajdować przybliżone strategie lidera w Grach Stackelberga oraz weryfikacja skuteczności

tych metod.

Kolejnym celem jest zaproponowanie zbiorów gier, na których można wykonać porównanie skuteczności różnych metod aproksymujących strategię lidera w stanie Równowagi Stackelberga. Budowa takiego zbioru jest konieczna, ponieważ metody prezentowane dotąd w literaturze testowane są na niedużych, specyficznych dla danej pracy, zbiorach gier, co bardzo utrudnia porównania różnych metod pomiędzy sobą. Następnie zaproponowane metody są porównane z metodami z literatury pod kątem czasu obliczeń, jak i jakości uzyskiwanych strategii. Tak przeprowadzona analiza umożliwia ocenę jakości znajdowanych strategii przez proponowane metody.

Główna część rozprawy poprzedzona jest rozdziałem teoretycznym wprowadzającym teorię gier i przeglądem istniejących metod rozwiązujących Gry Stackelberga.

1.4 Hipoteza badawcza i szczegółowe cele badawcze

Główna hipoteza badawcza stawiana w rozprawie sformułowana jest w następujący sposób:

> *Możliwe jest wykorzystanie metod Monte Carlo do efektywnego aproksymowania strategii lidera w wielokrokowych Grach Stackelberga o sumie niezerowej z niepełną informacją, które dają niewiele gorszą wartość oczekiwaną wypłaty lidera, przy optymalnej odpowiedzi naśladowcy, w porównaniu z wartością oczekiwaną wypłaty lidera w stanie równowagi.*

Lista szczegółowych celów badawczych jest następująca:

1. Budowa dwóch metod aproksymacji stanu Równowagi Stackelberga.
   (a) Metody zapewniającej dużą szybkość obliczeń.
   (b) Metody wolniejszej, ale dającej bardziej precyzyjne wyniki.
2. Porównanie szybkości działania i jakości uzyskiwanych wyników:
   (a) proponowanych metod między sobą,
   (b) proponowanych metod z podejściami istniejącymi w literaturze.
3. Zaproponowanie dwóch rodzin gier wielokrokowych o sumie niezerowej i niepełnej informacji do testowania metod.

1.5 Oryginalne wyniki rozprawy

W ramach rozprawy zaprezentowane są następujące wyniki powstałe w ramach realizacji postawionych celów badawczych, które są oryginalnym wkładem w dziedzinę:

- Metoda Mixed-UCT, zbudowana na bazie metody Upper Confidence bounds applied to Trees (UCT) [50], która aproksymuje strategię lidera w wielokrokowych Grach Stackelberga z niepełną informacją o sumie niezerowej z dodatkowym ograniczeniem na strukturę zbiorów informacyjnych opisanym w dalszej części rozprawy.

- Metoda O2-UCT, która aproksymuje strategię lidera w wielokrokowuch Grach Stackelberga z niepełną informacją o sumie niezerowej, bez żadnych dodatkowych ograniczeń na strukturę gry.

- Generator gier testowych imitujących budynki wraz z wygenerowanym zbiorami testowymi do oceny metod aproksymujących strategie.

Dodatkowym oryginalnym wynikiem, który nie jest bezpośrednio związany z postawioną hipotezą badawczą, jest modyfikacja metody SMOS (Stackelberg Model of the Oil-Siphoning problem) [90]. Oryginalna metoda SMOS działa dla pewnej rodziny gier wielokrokowych o sumie zerowej, a proponowana modyfikacja umożliwia jej stosowanie dla gier o sumie niezerowej. Modyfikacja ta jest wspólną pracą autora rozprawy i Filipa Grajka, w momencie opracowywania metody studenta studiów magisterskich. Modyfikacja ta wnosi istotny wkład w rozdział niniejszej rozprawy, który opisuje istniejące podejścia do Gier Stackelberga. Pokazuje ona, jak wzrasta poziom skomplikowania algorytmu przy przejściu do sumy niezerowej i jak trudna jest modyfikacja metod specyficznych do danego modelu gry.

1.6 Opublikowane wyniki rozprawy

Elementy tej rozprawy były opublikowane wcześniej w następujących pracach naukowych:

- Praca konferencyjna (C CORE 2018): Jan Karwowski i Jacek Mańdziuk. *A New Approach to Security Games*. W: *Artificial Intelligence and Soft Computing - 14th International Conference, ICAISC 2015, Zakopane, Poland, June 14-28, 2015, Proceedings, Part II*. Red. Leszek Rutkowski i in. T. 9120. Lecture Notes in Computer Science. Springer, 2015, s. 402–411. ISBN: 978-3-319-19368-7. DOI: 10.1007/978-3-319-19369-4_36 — metoda próbowania w grach z niepełną informacją będąca podstawą metody Mixed-UCT

- Praca konferencyjna (A CORE 2018): Jan Karwowski i Jacek Mańdziuk. *Mixed Strategy Extraction from UCT Tree in Security Games*. W: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. Red. Gal A. Kaminka i in. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, s. 1746–1747. ISBN: 978-1-61499-671-2. DOI: 10.3233/978-1-61499-672-9-1746 — wprowadzenie metody Mixed-UCT

- Praca konferencyjna (C CORE 2018): Jan Karwowski i Jacek Mańdziuk. *The Impact of the Number of Averaged Attacker's Strategies on the Results Quality in Mixed-UCT*. W: *Artificial Intelligence and Soft Computing - 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11-15, 2017*. Red. Leszek Rutkowski i in. T. 10246. Lecture Notes in Computer Science. Springer, 2017, s. 477–488. ISBN: 978-3-319-59059-2. DOI: 10.1007/978-3-319-59060-8_43 — analiza wpływu długości historii strategii atakującego, jednego z parametrów metody, na szybkość i skuteczność jej działania.

- Praca konferencyjna (A* CORE 2018) Jan Karwowski, Jacek Mańdziuk, Adam Żychowski, Filip Grajek i Bo An. *A Memetic Approach for Sequential Security Games on a Plane with Moving Targets*. W: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, s. 970–977. ISBN: 978-1-57735-809-1 — w sekcji 4. tej pracy została przedstawiona modyfikacja znanej z literatury metody SMOS [90] do gier o sumie niezerowej.

- Praca w czasopiśmie: Jan Karwowski i Jacek Mańdziuk. *A Monte Carlo Tree Search approach to finding efficient patrolling schemes on graphs*. W: *European Journal of Operational Research* 277.1 (2019), s. 255–268. DOI: 10.1016/j.ejor.2019.02.017 — zawierająca pełny opis metody Mixed-UCT wraz z rozszerzoną ewaluacją eksperymentalną względem wcześniejszych prac konferencyjnych.

- Rozszerzony abstrakt na konferencji (A* CORE 2018) Jan Karwowski i Jacek Mańdziuk. *Stackelberg Equilibrium Approximation in General-Sum Extensive-Form Games with Double-Oracle Sampling Method*. W: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. Red. Edith Elkind, Manuela Veloso, Noa Agmon i Matthew E. Taylor. International Foundation for Autonomous Agents i Multiagent Systems, 2019, s. 2045–2047. ISBN: 978-1-4503-6309-9 — wprowadzenie metody O2-UCT.

- Praca konferencyjna (A* CORE 2020) Jan Karwowski i Jacek Mańdziuk. *Double-Oracle Sampling Method for Stackelberg Equilibrium Approximation in General-Sum Extensive-Form Games*. W: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.02 (2020), 2054–2061. ISSN: 2159-5399. DOI: 10.1609/aaai.v34i02.5578 — zawierająca opis metody O2UCT.

1.7 Układ rozprawy

Rozdział 2 formalnie definiuje używane pojęcia teorii gier, w tym Równowagę Stackelberga. Oprócz definicji rozdział zawiera dowód NP-trudności poszukiwania Równowagi Stackelberga w klasie gier rozważanej w rozprawie oraz obserwacje i twierdzenia, które są podstawą budowy efektywnych metod obliczających Równowagę Stackelberga.

Rozdział 3 opisuje metody poszukiwania Równowagi Stackelberga znane z literatury. W tym rozdziale opisujemy techniki wykorzystywane w metodach dedykowanych grom modelującym konkretne zastosowanie, które pozwalają przyspieszyć obliczenia oraz wskazujemy, w jaki sposób utrudniają one budowę ogólnych metod. Ograniczenia te motywują konieczność budowy zupełnie nowych metod aproksymujących strategię lidera w Grach Stackelberga. W dalszej części rozdziału opisujemy metody dedykowane ogólnej klasie gier, które posłużą jako punkt odniesienia przy ocenie skuteczności autorskich metod.

Rozdział 4 opisuje część stanowiącą większość oryginalnego wkładu rozprawy – dwie metody heurystyczne aproksymujące strategię lidera w Stanie Równowagi Stackelberga. Rozdział ten przedstawia decyzje projektowe podjęte w trakcie projektowania metod, następnie szczegóły ich działania. Prezentowana jest też ewaluacja eksperymentalna tych metod. Na początku rozdziału prezentowane są zbiory testowe, zaproponowane przez autora rozprawy, które zostały wykorzystane do ewaluacji metod, następnie prezentowana jest pierwsza z metod, *Mixed-UCT*, która jest próbą zastosowania wprost metody Upper Confidence Bound applied to Trees (UCT) do poszukiwania strategii lidera ze stanu Równowagi Stackelberga. W drugiej kolejności prezentowana jest metoda *O2-UCT*, która odchodzi od pomysłu stosowania UCT bezpośrednio do poszukiwania strategii i zamiast tego stosuje schemat metody zainspirowany podejściami do Gier Stackelberga znanymi z literatury, a metodę UCT wykorzystuje tylko jako narzędzie do realizacji konkretnych fragmentów rozwiązania. Każdy z opisów proponowanych metod zawiera opis eksperymentów, w których proponowana metoda porównywana jest z metodami z literatury i w której oceniana jest jej skuteczność.

Rozprawę kończy Rozdział 5, który podsumowuje wyniki eksperymentalne, wskazuje mocne i słabe strony proponowanych metod. Następnie, na podstawie wniosków wyciągniętych z eksperymentów, stwierdzamy prawdziwość postawionej hipotezy badawczej. W tym rozdziale wskazujemy również dalsze kierunki badań i plany na rozszerzenia i ulepszenia metod proponowanych w tej rozprawie.

Rozdział 2

Użyte pojęcia teorii gier i definicja Równowagi Stackelberga

Ten rozdział wprowadza znane z literatury formalne definicje gier i stanów równowagi rozważanych w rozprawie oraz prezentuje obecny stan wiedzy na temat trudności obliczania i aproksymacji stanu Równowagi Stackelberga. Przedstawione tu definicje pochodzą z wielu źródeł powstałych na przestrzeni kilkudziesięciu lat i nie są przytoczone w oryginalnym brzmieniu, a dostosowane do notacji i nomenklatury stosowanej w tej rozprawie. Rozdział wprowadza kilka powszechnie używanych sposobów przedstawiania strategii w grach, które będą użyte dalej, w Rozdziale 3 przedstawiającym znane z literatury metody obliczania strategii lidera w Stanie Równowagi Stackelberga oraz w Rozdziale 4 prezentującym oryginalny wkład autora rozprawy.

Rozważanym wariantem stanu Równowagi Stackelberga jest Silna Równowaga Stackelberga (Strong Stackelberg Equilibium) [54] opisana szczegółowo w dalszej części tego rozdziału. W rozważaniach teoretycznych gry będą opisywane z użyciem postaci ekstensywnej (extensive form game) [53, 52]. Po przytoczeniu odpowiednich definicji gier, zostanie formalnie zdefiniowany stan Równowagi Stackelberga. Na koniec tej rozdziału zostaną wprowadzone definicje, które nie są wykorzystywane przy rozważaniu metod będących przedmiotem rozprawy, ale są niezbędne do przedstawienia rozwiązań występujących w literaturze.

Głównym przedmiotem rozprawy są gry Stackelberga dla dwóch graczy (jeden lider i jeden naśladowca). Podane reprezentacje gier będą w wielu przypadkach definiowane dla dowolnej liczby graczy, jednak dla ustalenia uwagi można przyjąć, że zbiór wszystkich graczy $N = \{l, f\}$ — odpowiednio lider i naśladowca (ang. follower, stąd indeks dolny f).

2.1 Podstawowe definicje

Poniżej są przedstawione powszechnie znane definicje gier, które rozwiązywane są przez metody omówione w rozprawie w tej rozprawie, zarówno definicje, które dotyczą metod będących przedmiotem rozprawy, jak i definicje potrzebne do omówienia prac innych autorów z obszaru

badań. Teoria gier jest narzędziem używanym w wielu różnych obszarach i jedną z konsekwencji tego rozrzucenia jest zróżnicowana terminologia określająca te same byty, zarówno w literaturze polskojęzycznej, jak i anglojęzycznej. W związku z tym definicje często będą podawały kilka nazw bytu. Zawsze pierwsza z podanych nazw będzie nazwą, która jest używana w rozprawie.

Każda gra, niezależnie od szczegółów definicji modelu, będzie składała się z [11]:

- skończonego zbioru graczy, oznaczanego symbolem $\mathcal{N}$
- zbiorów akcji, które gracze mogą wykonać w konkretnym momencie gry. Co do zasady zbiór akcji może być skończony, przeliczalny lub nieprzeliczalny. W przypadku, gdy wszystkie zbiory akcji są skończone, mówimy o grze skończonej, w przeciwnym przypadku mówimy o grze nieskończonej. W tej rozprawie będą występować wyłącznie gry skończone.
- sposobu przebiegu rozgrywki — jakie kolejni gracze wykonują akcje i co wiedzą na temat stanu gry. Rozgrywką będziemy nazywali stan końcowy gry [1] związany z wykonaniem przez wszystkich graczy wszystkich akcji wymaganych przez zasady gry,
- sposób przeprowadzenia rozgrywki i zbiory akcji będą definiować zbiory strategii prostych gracza. Dla każdego gracza $i \in \mathcal{N}$ będzie zdefiniowany jeden zbiór strategii prostych, oznaczany zwykle Π_i.
- funkcji wypłat, które będą definiować wynik uzyskany przez gracza na koniec gry. Wypłata gracza zawsze będzie liczbą rzeczywistą. Wypłata i-tego gracza, w momencie gdy gracze zdecydowali się zagrać odpowiednio strategie proste $\pi_1, \ldots \pi_n$, będzie oznaczana symbolem $u_i(\pi_1, \ldots \pi_n)$.

Symbole $\mathcal{N}$, Π_i, u_i w dalszej treści rozprawy będą oznaczały byty opisane powyżej, o ile wyraźnie nie zaznaczono inaczej. Ścisłe definicje poszczególnych klas gier są przytoczone w dalszej części tego rozdziału. Każdą z tych gier można będzie opisać z użyciem krotki $(\mathcal{N}, \{\Pi_i\}_{i\in\mathcal{N}}, \{u_i\}_{i\in\mathcal{N}})$.

Poniżej przedstawiona jest ogólna charakteryzacja gier uwzględniająca cechy gier będących przedmiotem rozprawy.

Definicja 2.1. *Grę Γ nazywamy* grą o sumie zerowej *[65, Rozdział 11.4], jeśli dla każdej możliwej rozgrywki, suma wypłat wszystkich graczy wynosi 0.*

Definicja 2.2. *Grę Γ nazywamy* grą o sumie niezerowej*, (stosowana jest również nazwa gra o sumie ogólnej lub gra o zmiennej sumie), jeśli nie jest spełniony warunek sumy zerowej.*

[1] Gry są sformułowane w taki sposób, że stan końcowy jednoznacznie definiuje wszystkie wykonane akcje.

Uwaga. Mimo, że definicje 2.1 i 2.2 są spełnione dla rozłącznych klas gier, w tej rozprawie klasa gier o sumie niezerowej będzie oznaczać klasę zawierającą zarówno gry o sumie niezerowej, jak i gry o sumie zerowej. W literaturze stosuje się też czasami termin *gry o sumie ogólnej*.

Uwaga 2. Możemy wyróżnić szczególną klasę gier, o sumie stałej, to znaczy takich, gdzie dla każdej gry istnieje pewna stała $k \in \mathbb{R}$, taka że suma wypłat graczy dla każdej rozgrywki wynosi k. Ta klasa gier nie jest istotnie ciekawsza od klasy gier o sumie zerowej [11]. Każdą taką grę można sprowadzić do gry o sumie zerowej poprzez odjęcie od wypłat graczy wartości $k/|\mathcal{N}|$. W związku z tym wszędzie tam, gdzie jest mowa o grach o sumie niezerowej należy myśleć przede wszystkim o grach, gdzie sumy wypłat graczy różnią się pomiędzy rozgrywkami.

Definicja 2.3. *Grą* jednokrokową *[11, Rozdział 2.4] będziemy nazywać grę, w której każdy z graczy wybiera akcję dokładnie raz, a wszyscy gracze podejmują swoją decyzję jednocześnie.*

Definicja 2.4. *Grą* wielokrokową *[11, Rozdział 2.5] (inna używana nazwa to gra sekwencyjna), będziemy nazywać grę, gdzie gracze nie wybierają ruchów jednocześnie lub któryś z graczy kilkukrotnie wybiera ruch w grze.*

Definicja 2.5. *Grą* z pełną informacją *[53, Sekcja 3.], (także grą z doskonałą informacją), będziemy nazywać grę, gdzie gracz w momencie podejmowania decyzji o ruchu ma pełną wiedzę o stanie gry i ruchach wykonanych przez innych graczy.*

Definicja 2.6. *Grą* z niepełną informacją*, (także grą z niedoskonałą informacją), będziemy nazywać grę, która nie spełnia definicji* pełnej informacji*, tzn. gdzie gracz w momencie podejmowania decyzji o ruchu może nie mieć pełnej wiedzy o stanie gry.*

Definicja 2.7. *Grą* niekooperacyjną *[11] będziemy nazywać grę, w której gracze nie mają żadnych możliwości komunikacji przed rozgrywką lub w trakcie rozgrywki, poza informacjami ujawnianymi przez zasady gry.*

Przykładem gry z pełną informacją są takie gry planszowe jak warcaby, szachy, czy go, gdzie bierki czy kamienie na planszy stanowią cały opis stanu gry i są przez cały czas widoczne dla wszystkich. Kanonicznym przykładem gier z niepełną informacja są gry karciane takie jak poker czy brydż, gdzie gracz wie jedynie jakie karty ma na ręce oraz ma pewną informację o zagraniach innych graczy, na przykład licytacji, w pokerze również o liczbie wymienionych kart, czy kartach znajdujących się na stole.

2.2 Reprezentacja gier jednokrokowych

Najprostszym modelem gry, który warty jest rozważenia, jest jednokrokowa dla dwóch graczy ($|\mathcal{N}| = 2$) o sumie zerowej. Najbardziej rozpowszechnioną formą reprezentacji takiej gry jest postać normalna, nazywana też macierzową lub strategiczną. Ruchy graczy są definiowane odpowiednio przez wiersze i kolumny macierzy, a wypłaty przez elementy tej macierzy.

Definicja 2.8. *Grą dwuosobową o sumie zerowej w postaci normalnej [65, Rozdział 12.] nazywamy krotkę* $(\mathcal{N}, A_c, A_r, M)$*, gdzie:* $\mathcal{N} = \{r,c\}$ *— zbiór graczy (*r *— gracz wierszowy,* c *— gracz kolumnowy),* A_c *— skończony zbiór akcji gracza* c*,* A_r *— skończony zbiór akcji gracza* r*,* $M \in \mathbb{R}^{|A_r|\times|A_c|}$*, Wiersze macierzy są etykietowane elementami zboru* A_r*, a kolumny elementami zbioru* A_c*. Wartość funkcji wypłaty gracza kolumnowego to odpowiedni element macierzy* M $u_c : A_r \times A_c \to \mathbb{R} = M(a_r a_c)$*, wartość wypłaty gracza kolumnowego jest liczbą przeciwną do wypłaty gracza wierszowego:* $u_r : A_r \times A_c \to \mathbb{R} = -M(a_r a_c)$.

Uwaga. Macierz M jest jedynym elementem niezbędnym do opisania gry. Pozostałe elementy krotki są obecne w definicji tylko dla poprawy czytelności notacji.

Ważnym elementem gry w tak zdefiniowanej postaci normalnej jest, to że gracze wybierają ruchy jednocześnie, nie ma sytuacji, gdy jeden z graczy wie jak zagrał przeciwnik przed podjęciem swojej decyzji.

Przykładem prostej gry, którą można reprezentować w postaci normalnej jest gra, w której każdy z graczy ma monetę. W tajemnicy przed przeciwnikiem gracze odwracają swoją monetę reszką lub orłem do góry, a następnie odsłaniają monetę. W przypadku, gdy obie monety są odwrócone tym samym symbolem do góry, pierwszy gracz dostaje jeden punkt, a drugi traci jeden punkt. W przeciwnym wypadku wynik gry jest odwrotny. Grę tę można opisać następującą macierzą:

	O	R
O	1	-1
R	-1	1

Reprezentację tę można łatwo uogólnić na gry, które nie mają własności sumy zerowej — suma wyników graczy w każdej z rozgrywek może być inna.

Definicja 2.9. *Grą dwuosobową o sumie niezerowej [11, Rozdział 2.2.] w postaci normalnej nazywamy krotkę* $(\mathcal{N}, A_c, A_r, M_r, M_c)$*, gdzie:* $N = \{r,c\}$ *— zbiór graczy,* A_c *— skończony zbiór akcji gracza* c*,* A_r *— skończony zbiór akcji gracza* r*,* M_c *— macierz wypłat gracza* c*,* M_r *— macierzy wypłat gracza* r*. W obu macierzach wiersze i kolumny etykietowane tak, jak w przypadku sumy zerowej.*

Jedyną różnicą w stosunku do sumy zerowej są wartości wypłat graczy. Tym razem wynikiem gracza r jest odpowiednie element macierzy M_r, a wynikiem gracza c odpowiedni element gracza M_c.

2.3 Pojęcia strategii prostej i strategii mieszanej

W tym rozdziale zostaną przedstawione podstawowe pojęcia związane ze strategiami graczy. Jeśli nie zaznaczono inaczej, są to definicje wprowadzone przez Von Neumanna [65] przełożone na język oznaczeń używany w tej pracy.

Definicja 2.10. *Strategią prostą, (także strategią podstawową lub czystą, ang. pure strategy) gracza $i \in \mathcal{N}$ będziemy nazywać wybór przez gracza dokładnie jednej akcji w grze w postaci normalnej. Zbiór strategii prostych gracza i będziemy oznaczać symbolem Π_i, a strategie (elementy tego zbioru) zwykle symbolem π_i.*

Definicja 2.11. *Strategią mieszaną gracza $i \in \mathcal{N}$ będziemy nazywać rozkład prawdopodobieństwa nad zbiorem strategii prostych tego gracza. Zbiór strategii mieszanych gracza i będziemy oznaczać Δ_i, a pojedynczą strategię mieszaną δ_i.*

Definicja 2.12. *Profilem strategii (prostych lub mieszanych) będziemy nanazywać krotkę strategii, po jednej dla każdego z graczy biorących udział w grze. Dla profilów strategii prostych będziemy stosować oznaczenie $\Pi = \times_{i \in \mathcal{N}} \Pi_i$ i odpowiednio dla profilów strategii złożonych $\Delta = \times_{i \in \mathcal{N}} \Delta_i$*

Definicja 2.13. *Wypłatą gracza i dla zadanego profilu strategii mieszanych δ będziemy nazywać wartość oczekiwaną funkcji wypłaty u_i dla strategii będących zmiennymi losowymi o rozkładach zadanych przez strategie mieszane δ_i wchodzące w skład tego profilu.*

Dopuszczając się nadużycia notacji, będziemy stosować symbol $u_i(\delta)$ lub $u_i(\delta_1, \delta_2, \ldots, \delta_n)$ do oznaczenia tej wartości oczekiwanej.

Intuicje i przykłady strategii mieszanych są podane w Podrozdziale 2.5 wprowadzającej stany równowagi w dalszej części tego rozdziału.

2.4 Reprezentacja gier sekwencyjnych (wielokrokowych)

W tym podrozdziale opisujemy typowe sposoby reprezentacji gier sekwencyjnych, które pozwalają na elegancką i bardziej zwięzłą reprezentację faktu, że gracze podejmują decyzję wielokrotnie i obserwują pośrednie skutki tych decyzji. Grę w każdej z przedstawionych reprezentacji można sprowadzić do postaci normalnej, ale taka reprezentacją będzie większa w sensie liczby potrzebnych wartości opisujących poszczególne wyniki gry, a co ważniejsze nie będzie umożliwiała zastosowania bardziej efektywnych metod obliczeniowych, wykorzystujących wielokrokową strukturę gry.

2.4.1 Gra w postaci ekstensywnej

Najbardziej rozpowszechnioną formą reprezentacji gier wielokrokowych jest postać ekstensywna. Postać ta była bez formalnej definicji używana już we wczesnych pracach z teorii gier, a współcześnie używana formalizacja pojęcia została dokonana przez Kuhna w roku 1950 [53]. Postać ekstensywna opisuje grę jako ukorzenione drzewo, w którym krawędzie etykietowane są ruchami kolejnych graczy, a liście są etykietowane wynikiem gry. Oryginalna definicja Kuhna jest nieco szersza, niż wariant używany w tej rozprawie. Praca Kuhna obejmowała gry, w których poza graczami, na przebieg rozgrywki wpływa losowy czynnik zewnętrzny, zwany naturą,

który wykonuje akcje według rozkładu prawdopodobieństwa znanego wszystkim graczom. Taki element może posłużyć na przykład do modelowania rzutów kostką w grze planszowej. Klasy gier rozważane w tej rozprawie nie obejmują gier z czynnikiem losowym, a co za tym idzie pomijamy go we wszystkich definicjach aby nie wprowadzać nadmiarowej komplikacji.

Definicja 2.14. *Grą w postaci ekstensywnej [53, 52] z niepełną informacją będziemy nazywać krotkę* $G = (\mathcal{N}, \mathcal{S}, \mathcal{Z}, \rho, \mathcal{A}, u, \mathcal{I})$*, gdzie:*

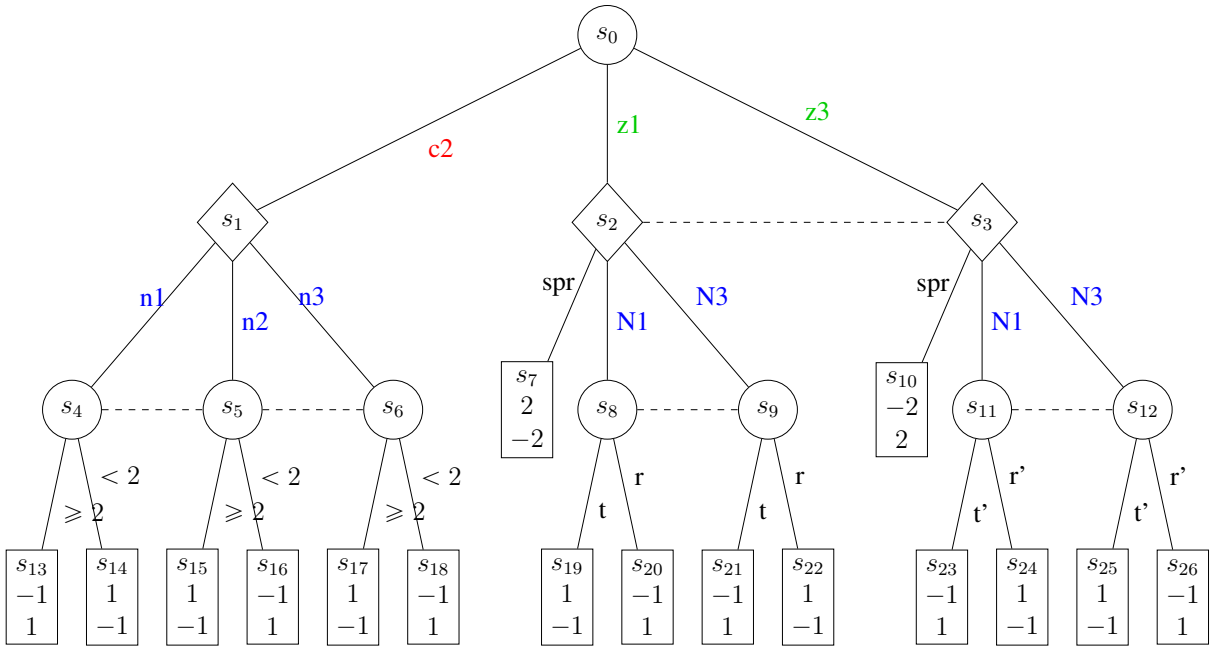
- $\mathcal{N}$ *— zbiór graczy,*
- $\mathcal{S}$ *— zbiór nieterminalnych stanów gry z wyróżnionym* s_0 *– stanem startowym, Elementy zbioru* S *są wierzchołkami skierowanego drzewa gry o korzeniu* s_0*, krawędzie tego drzewa są etykietowane akcjami ze zbioru* A *opisanego poniżej.*
- $\mathcal{Z}$ *— zbiór terminalnych stanów gry – liści drzewa gry.*
- $\rho : \mathcal{S} \to \mathcal{N}$ *— funkcja definiująca gracza, który podejmuje decyzję (wykonuje ruch) w danym stanie.*
- $\mathcal{A} = \{A_s\}_{\forall s \in \mathcal{S}}$ *— rodzina zbiorów (niekoniecznie rozłącznych)* A_s*, każdy zbiór definiuje akcje (ruchy) dostępne w stanie* s*, zbiór akcji dostępnych w stanie* s *jest izomorficzny ze zbiorem krawędzi wychodzących z tego stanu w drzewie gry – dokładnie jedna akcja odpowiada dokładnie jednej krawędzi wychodzącej z tego stanu, nie musi to być jedyny stan, z którego wychodzi dana akcja.*
- $u : \mathcal{Z} \times \mathcal{N} \to \mathbb{R}$ *— funkcja definiująca wypłatę każdego z graczy po osiągnięciu stanu terminalnego.*
- $\mathcal{I}$ *— podział zbioru* $\mathcal{S}$ *na zbiory informacyjne* I_i *(information sets), taki że:*
  - $\bigcup_k I_k = \mathcal{S}$*,*
  - $\forall I_1, I_2 \in \mathcal{I} \quad (I_1 \neq I_2 \Rightarrow I_1 \cap I_2 = \varnothing)$*,*
  - $\forall I_k \in \mathcal{I}\ \exists! n \in \mathcal{N}\ \forall s \in I_k \quad (\rho(s) = n)$ *(w obrębie każdego ze zbiorów rusza się ten sam gracz),*
  - $\forall I_k \in \mathcal{I}\ \forall s_1, s_2 \in I_k \quad (A_{s_1} = A_{s_2})$ *(dopuszczalne ruchy w obrębie zbioru są takie same).*

  Stany będące w jednym zbiorze informacyjnym są nierozróżnialne dla gracza. Bez straty ogólności będziemy również zakładać, że każda akcja dostępna jest w nie więcej niż jednym zbiorze informacyjnym: $\forall s_1, s_2 \in S \quad A_{s_1} \cap A_{s_2} \neq \varnothing \Leftrightarrow \exists I \in \mathcal{I} \quad s_1 \in I \wedge s_2 \in I$*. To dodatkowe założenie uprości notację w rozważaniach teoretycznych, pozwalając na jednoznaczną identyfikację, z którego zbioru informacyjnego pochodzi dana akcja.*

Przykład gry w postaci ekstensywnej. Rozważmy grę dla dwóch graczy. Pierwszy gracz dysponuje trzema kamieniami: dwoma zielonymi, o nominałach 1 i 3 oraz jednym czerwonym o nominale 2. W pierwszych ruchu pierwszy gracz wykłada na stół jeden z kamieni nominałem do dołu (tak, że jeśli został wybrany zielony kamień, przeciwnik nie zna jego nominału). Gracz drugi dysponuje trzema niebieskimi kamieniami o nominałach 1, 2 i 3. Jeśli pierwszy gracz zagrał czerwony kamień, to drugi gracz wykłada jeden ze swoich kamieni nominałem do dołu, a pierwszy gracz zgaduje, czy wyłożony kamień ma nominał większy równy, czy mniejszy od 2. Jeśli pierwszy gracz zagrał zielony kamień, drugi gracz może się zdecydować go odwrócić. Jeśli odwrócony kamień to 3, to drugi gracz wygrywa podwójną stawkę, w przeciwnym wypadku przegrywa podwójną stawkę. Zamiast zgadywać wartość kamienia drugi gracz może wyłożyć niebieski kamień o nominale 1 lub 3 nominałem do dołu. Wtedy pierwszy gracz zgaduje czy nominały kamieni się zgadzają, czy nie. Jeśli zgadł, wygrywa, jeśli nie, to przegrywa.

Rysunek 2.1 przedstawia powyższą grę w postaci ekstensywnej. Rysunek przedstawia drzewo o korzeniu w węźle s_0, z którego wychodzą trzy krawędzie etykietowane trzema możliwymi ruchami pierwszego gracza – włożenie czerwonego kamienia o nominale 2 (na rysunku oznaczone jako c2), wyłożenie zielonego kamienia o nominale 1 i wyłożenie zielnego kamienia o nominale 3, ruchy odpowiednio oznaczone etykietami z1 i z3. Trzy krawędzie oznaczone tymi ruchami dochodzą do trzech wierzchołków, krawędź c2 do stanu s_1, krawędź z1 do stanu s_2, krawędź z3 do staniu $s3$. Stany s_2 i s_3 są połączone przerywana linią, która oznacza, że należą do tego samego zbioru informacyjnego. Ta przynależność służy do zamodelowania faktu, że gracz drugi zna tylko kolor wyłożonego kamienia, ale nie jest w stanie powiedzieć, czy jest to nominał 1, czy 3. Następnie, z wierzchołka s_1 wychodzą trzy ruchy oznaczone n1, n2, n3, które oznaczają wyłożenie przez drugiego gracza kamienia odpowiednio o nominale 1, 2 i 3. Ruchy te prowadzą odpowiednio do stanów s_4, s_5 i s_6. Wszystkie 3 stany są w jednym zbiorze informacyjnym, ponieważ pierwszy gracz nie wie jaki kamień jest wyłożony. Z węzłów s_2 i s_3, z definicji zbioru informacyjnego, muszą być możliwe do zagrania dokładnie te same ruchy. Ruchy te oznaczone są jako spr – sprawdzenie czy pierwszy gracz zagrał kamień 3, wyłożenie jednego z niebieskich kamieni 1 lub 3. Ruchy te są oznaczone odpowiednio N1 i N3. Z punktu widzenia definicji gry ekstensywnej ruchy te mogłyby mieć etykiety n1 i n3, ale przyjęliśmy konwencję nazywania ruchów dostępnych w różnych zbiorach informacyjnych różnymi symbolami. Konstrukcja głębszych partii drzewa przebiega w analogiczny sposób. Dla zachowania czytelności, rysunek nie przedstawia etykiet zbiorów informacyjnych. Na potrzeby rozważania tego przykładu w dalszych częściach przyjmiemy etykietowanie zbiorów informacyjnych jako I_k, gdzie k będzie najmniejszym indeksem s_k węzłów wchodzących w skład tego zbioru. Na przykład zbiór $\{s_4, s_5, s_6\}$ będzie oznaczony symbolem I_4. Uwaga. O ile w przykładzie gracze wykonywali ruchy naprzemiennie, to w ogólności definicja tego nie wymusza. Jest możliwe opisanie w postaci ekstensywnej gry, w której gracz rusza się dwa razy z rzędu.

Dodatkowo zdefiniujmy zestaw pomocniczych funkcji. Funkcje te będą w wielu przypadkach oznaczane tymi samymi symbolami, to elementy definicji gry, ale kontekst użycia, przede



Rysunek 2.1: Postać ekstensywna przykładowej gry. Kształt węzłów oznacza wartość funkcji ρ wskazującej, który z graczy wykonuje ruch. Koło oznacza gracza pierwszego, romb – gracza drugiego. Węzły terminalne są oznaczone prostokątami. Węzły połączone przerywaną linią należą do tego samego zbioru informacyjnego. Liczby w liściach definiują wartości wypłat.

wszystkim dziedzina funkcji, będzie jednoznacznie wskazywał która z definicji jest użyta.

Bezpośredni poprzednik stanu s innego niż s_0 będzie oznaczany $\text{anc}(s)$, zbiór bezpośrednich następników stanu s: $\text{succ}(s)$, przyjmujemy że $\forall s \in Z \quad \text{succ}(s) = \varnothing$. Ścieżka etykiet krawędzi (akcji) od korzenia do węzła s to $\sigma(s)$. Dodatkowo zdefiniujemy rodzinę ścieżek etykiet krawędzi zawierającą tylko krawędzie będące ruchami gracza $i \in N$: $\sigma_i(s)$.

Podział rodziny zbiorów informacyjnych na rodzinę punktów decyzyjnych Lidera i naśladowcy:

$$\mathcal{I}_l = \{I \in \mathcal{I} | \forall s \in I \quad \rho(s) = l\} \tag{2.1}$$

$$\mathcal{I}_f = \{I \in \mathcal{I} | \forall s \in I \quad \rho(s) = f\} \tag{2.2}$$

Funkcja podająca w którym zbiorze informacyjnym jest dany stan: $I : \mathcal{S} \backslash \mathcal{Z} \to \mathcal{I}$. $I(s) = I$ takie, że $s \in I$. Z racji tego, że $\mathcal{I}$ jest podziałem zbioru $\mathcal{S} \backslash \mathcal{Z}$, taka definicja jest jednoznaczna.

Definicja 2.15. Strategią prostą *[52, Definicja 3.] gracza i w grze w postaci ekstensywnej będziemy nazywali funkcję $\pi_i : \mathcal{I}_i \to \sum_{A \in \mathcal{A}} A$, przypisującą każdemu ze zbiorów informacyjnych $I_k \in \mathcal{I}_i$ dokładnie jednej akcji z odpowiadającego mu zbioru dostępnych akcji $A(I_k)$. Formalnie wymagamy aby:*

$$\forall I \in \mathcal{I}_i \quad \pi_i(I) \in A_I$$

Strategie proste, tak jak w grach w postaci normalnej będziemy oznaczać $\pi_i \in \Pi_i$.

Tak zdefiniowana strategia powoduje, że gracz musi przed gra zdefiniować swoje zacho-

wanie w każdym swoim punkcie decyzyjnym, w strukturze gry (również w punktach nieosiąganych w niektórych rozgrywkach). W przypadku przykładowej gry przedstawionej wcześniej jedną z dostępnych strategii prostych gracza pierwszego może być: $I_0 \mapsto z1, I_4 \mapsto (\geqslant 2)$, $I_8 \mapsto t$,$I_{11} \mapsto t'$. Przykładem strategii prostej drugiego gracza jest przypisanie $I_1 \mapsto n2$, $I_2 \mapsto spr$.

Będziemy mówili, że strategia π_i *wybiera* ruch a, jeśli $\pi_i(I(a)) = a$.

Definicja 2.16. *Będziemy mówili, że stan $s \in \mathcal{S}$ jest* istotny *w strategii π_i, jeśli istnieją strategie π_j dla pozostałych graczy takie, że stan s będzie na ścieżce rozgrywki [52, Definicja 6.], zbiór wszystkich istotnych stanów będziemy oznaczali Rel(π_i) (od ang. relevant). Będziemy mówili, że zbiór informacyjny I jest* istotny *w strategii π_i, jeśli istnieje co najmniej jeden stan s w tym zbiorze informacyjnym, który jest istotny w π_i. W przypadku ruchów, będziemy mówili, że ruch a jest* istotny *w π_i, jeśli $I(a)$ jest istotny w π_i.*

Istotne stany będą podstawą definicji strategii zredukowanych w Rozdziale 2.4.2.

Definicja 2.17. Strategią mieszaną *[52, Definicja 7.] gracza i w grze w postaci ekstensywnej będziemy nazywali rozkład prawdopodobieństwa nad jego strategiami prostymi. Strategie mieszane, tak jak w grach w postaci normalnej będziemy oznaczać $\delta_i \in \Delta_i$.*

Gra z doskonałą pamięcią

Definicja 2.18. *Grą z* doskonalą pamięcią *(ang. perfect recall game) [52, Definicja 17.] będziemy nazywać grę, w ktorej każdy z graczy odróżnia stany do których dotarł wykonując różne sekwencje ruchów, ściśle:*

$$\forall n \in N \; \forall s_1,s_2 \in \bigcup_{I \in \mathcal{I}_n} I \; \sigma_n(s_1) \neq \sigma_n(s_2) \Rightarrow I(s_1) \neq I(s_2) \tag{2.3}$$

W grach z doskonalą pamięcią dodatkowo dla każdego gracza i, dla każdego zbioru informacyjnego $I \in \mathcal{I}_i$ będziemy używać oznaczenia $\sigma_i(I) = \sigma_i(s)$, dla dowolnego $s \in I$. Powyższa funkcja jest dobrze zdefiniowana, ponieważ wartości $\sigma_i(s)$ dla każdego $s \in I$ są takie same w myśl warunku (2.3) w definicji gry z doskonałą pamięcią.

Definicja 2.19. *Strategią behawioralną (ang. behavior strategy)*[2] *[52, Definicja 14.] gracza i w grze w postaci ekstensywnej będziemy nazywać przypisanie każdemu ze zbiorów informacyjnych $I_k \in \mathcal{I}_i$ funkcji $b_{I_k} : A_{I_k} \to [0,1]$, takiej że $\sum_{a \in A_{I_k}} b(a) = 1$. Innymi słowy będzie to funkcja definiująca rozkład prawdopodobieństwa akcji dostępnych w zbiorze I_k. Będziemy stosować oznaczenie β_i na strategię behawioralną gracza i.*

[2] w języku angielskim występuje rozróżnienie między nazwą *behavioral strategy*, która oznacza uwzględnianie w analizie zachowań w obszarach ekonomii i zarządzania czynników psychologicznych wpływających na zachowanie jednostek, a nazwą *behavior strategy*, która jest opisana w tej definicji. Autorowi rozprawy nie jest znany żaden podobny niuans stosowany w języku polskim, który pozwalałby odróżniać te dwa pojęcia. W tej rozprawie będziemy jednak korzystać jedynie ze znaczenia wywodzącego się z teorii gier.

prawd.	I_0	I_4	I_8	I_{11}
0,3	c2	$\geqslant 2$	t	r'
0,2	c2	< 2	t	t'
0,3	z1	$\geqslant 2$	t	t'
0,2	z1	< 2	r	t'

(a) Strategia mieszana. W kolejnych wierszach wymienione są kolejne strategie proste wchodzące w skład strategii mieszanej. Każda kolumna przedstawia wybór ruchu w zbiorze informacyjnym będącym etykietą kolumny. Pierwsza kolumna definiuje prawdopodobieństwa wyboru konkretnej strategii prostej.

$\mathcal{I}$		
I_0	$(0{,}5; c2)$	$(0{,}5; z1)$
I_4	$(0{,}6; \geqslant 2)$	$(0{,}4; < 2)$
I_8	$(0{,}6; t)$	$(0{,}4; r)$
I_{11}	$(1{,}0; t')$	

(b) Strategia behawioralna. W kolejnych wierszach wymienione są zbiory informacyjne, a w kolumnach pary akcja, prawdopodobieństwo.

Tabela 2.1: Dwie równoważne reprezentacje strategii w grze przedstawionej na Rysunku 2.1.

W grach z doskonałą pamięcią strategie behawioralne są równoważne strategiom mieszanym. Poprzez równoważność rozumiemy, że dla każdej strategii behawioralnej istnieje strategia mieszana, która umożliwia uzyskanie dokładnie takiego samego prawdopodobieństwa osiągnięcia każdego z liści gry i na odwrót.

Twierdzenie 2.1. *W każdej grze z doskonalą pamięcią zbiór wszystkich profili strategii behawioralnych jest równoważny zbiorowi wszystkich profili strategii mieszanych [53, Twierdzenie 2.], [52, Twierdzenie 4.].*

Strategia behawioralna może być bardziej intuicyjnym sposobem opisywania decyzji w porównaniu do strategii mieszanej. Rozważmy strategię mieszaną pierwszego gracza opisaną w Tabeli 2.1a. Gdy zamienimy tę strategię w odpowiadającą jej strategię behawioralną, zostaną uwydatnione decyzje podejmowane w konkretnych zbiorach informacyjnych. Odpowiadająca strategia behawioralna jest przedstawiona w Tabeli 2.1b. W przypadku przedstawienia strategi w postaci mieszanej nie jest od razu widoczny fakt, że gracz zawsze wybiera ruch t', a nigdy r', prawdopodobieństwa ruchów w węzłach I_4 i I_8 nie są wprost widoczne. Ponadto strategia behawioralna jest bardziej kompaktowa, szczególnie dla dużych gier.

Oprócz obiektywnych cech tego sposobu reprezentacji strategii, bardzo ważny jest też fakt, że strategia behawioralna jest podstawowym elementem budowy metody O2-UCT będącej jedną z dwóch metod przybliżania Równowagi Stackelberga będącej oryginalnym wkładem tej rozprawy opisanej w Rozdziale 4.3.

2.4.2 Postać sekwencyjna

Postać ekstensywna jest czytelnym i intuicyjnym sposobem przedstawienia gry sekwencyjnej. Jest również pomocna, w przypadku programów symulujących przebieg gry. Jednak typowym

podejściem do poszukiwania stanów równowagi, opisanym szeroko w Rozdziale 3 tej rozprawy, są programy liniowe. Program liniowy, jest wygodnym sposobem rozwiązywania gier w postaci normalnej miedzy innymi dlatego, że podstawowym sposobem reprezentacji programu liniowego jest macierz, podobnie jak reprezentacja samej gry. Reprezentacja w postaci drzewa w żaden sposób nie przystaje do macierzowej natury programowania liniowego.

Jedną z metod zamiany gry w postaci ekstensywnej na postać macierzową jest transformacja Harsányiego [35]. Efektem tej transformacji jest gra w postaci normalnej, co pozwala zastosować metody rozwiązywania gier jednokrokowych. Wadą tego rozwiązania jest rozmiar macierzy powstałej gry, wykładniczy względem liczby *węzłów* w drzewie gry[35]. Transformacja Harsányiego polega na budowie gry macierzowej, w której ruchy każdego z graczy odpowiadają strategiom prostym w grze ekstensywnej. Jeśli rozważymy grę w postaci ekstensywnej z Rysunku 2.1, to po transformacji Harsányiego gra będzie opisana macierzą przestawioną w Tabeli 2.2. Poza wykładniczym wymiarem macierzy można zaobserwować dużą liczbę powtórzeń wartości, wartość 2, która jest wypłata w dokładnie jednym liściu gry pojawia się w macierzy 24 razy.

Techniką, która pozwala częściowo poprawić tę sytuację jest stosowanie zredukowanych strategii, w miejsce strategii prostych proponowanych w Definicji 2.15. Analizując strategię $I_0 \mapsto c2, I_4 \mapsto (\geqslant 2), I_8 \mapsto t, I_{11} \mapsto t'$ w przykładowej grze, możemy zaobserwować, że jeśli gracz wybrał ruch $z1$ w I_0, to przypisania do I_8 i I_{11} nie mają znaczenia, gdyż zagranie ruchu $c2$ przenosi nas do poddrzewa, gdzie stany z I_8 i I_{11} nie występują. Czasami taka redukcja nie będzie możliwa. Jeśli rozważymy strategię drugiego gracza $I_1 \mapsto n2, I_2 \mapsto spr$, to nie jest możliwe usunięcie żadnego z przypisań, ponieważ to czy gracz będzie podejmował decyzję w I_1, czy w I_2 zależy od ruchów przeciwnika, dopóki gra się nie rozpocznie, gracz nie może zredukować tej przestrzeni. Ta obserwacja prowadzi do definicji strategii zredukowanej.

Definicja 2.20. Strategią zredukowaną *[83, Sekcja 2.2.]* π_i^* *gracza* i *w grze w postaci ekstensywnej będziemy nazywać przypisanie akcji do każdego elementu pewnego podzbioru* $\mathcal{I}_i' \subseteq \mathcal{I}_i$ *takie, że*

1. π_i^* *jest obcięciem dziedziny pewnej strategii* π_i *do zbioru* $\mathcal{I}_i'$*,*
2. *każdy istotny, w sensie Definicji 2.16, zbiór informacyjny w strategii* π_i *należy do* $\mathcal{I}_i'$*,*
3. *do* $\mathcal{I}_i'$ *należą tylko te* I*, które są istotne w* π_i

Na bazie strategii zredukowanych można zbudować wariant transformacji Harsányiego, który da w wyniku równoważną grę w postaci normalnej, mniejszej niż macierz z Tabeli 2.2. Macierz ta została przedstawiona w Tabeli 2.3. Prezentowana macierz jest znacznie mniejsza niż wynik transformacji bez zastosowania strategii zredukowanych, jednak cały czas możliwe jest skonstruowanie gier, dla których ten wzrost będzie wykładniczy [51].

W związku z problemami z rozmiarem gier powstałych stosowaniu transformacji Harsányiego, w 1996 von Stengel, Koller i Megiddo [51, 82] zaproponowali inną postać gier wielo-

I_0		c2	c2	c2	c2	c2	c2	c2	z1	z1	z1	z1	z1	z1	z3	z3	z3	z3	z3	z3	z3	z3
I_4		$\geqslant 2$	$\geqslant 2$	$\geqslant 2$	$\geqslant 2$	< 2	< 2	< 2	$\geqslant 2$	$\geqslant 2$	$\geqslant 2$	< 2	< 2	< 2	$\geqslant 2$	$\geqslant 2$	$\geqslant 2$	< 2	< 2	< 2	< 2	< 2
I_8		t	t	r	r	t	r	t	t	r	t	t	r	t	t	r	r	t	r	t	r	r
I_{11}		t'	r'	t'	r'	t'	t	r'	t'	r'	t'	r'	r'	t'	t'	t'	r'	t'	t'	r'	t'	r'
I_1	I_2																					
n1	spr	−1	−1	−1	−1	1	1	2	2	2	2	2	2	−2	−2	−2	−2	−2	−2	−2	−2	−2
n1	N1	−1	−1	−1	−1	1	1	2	2	2	−1	1	1	−1	1	−1	1	−1	−1	1	−1	1
n1	N3	−1	−1	−1	−1	1	1	−1	−1	1	−1	−1	−1	−1	1	−1	1	−1	1	−1	1	−1
n2	spr	1	1	1	1	−1	−1	2	2	2	2	2	2	−2	−2	−2	−2	−2	−2	−2	−2	−2
n2	N1	1	1	1	1	−1	−1	2	2	2	−1	1	1	−1	1	−1	1	−1	1	−1	1	−1
n2	N3	1	1	1	1	−1	−1	−1	−1	1	−1	−1	−1	−1	−1	1	−1	1	−1	1	−1	1
n3	spr	1	1	1	1	−1	−1	2	2	2	2	2	2	−2	−2	−2	−2	−2	−2	−2	−2	−2
n3	N1	1	1	1	1	−1	−1	1	1	−1	1	1	−1	−1	−1	1	−1	1	−1	1	−1	1
n3	N3	1	1	1	1	−1	−1	−1	−1	1	−1	−1	1	1	1	−1	1	−1	1	−1	1	−1

Tabela 2.2: Efekt zastosowania transformacji Harsányiego do gry w postaci ekstensywnej na Rysunku 2.1. Widoczny jest wykładniczy wzrost rozmiaru gry.

	I_0	c2	c2	z1	z1	z3	z3
	I_4	$\geqslant 2$	< 2				
	I_8			t	r		
	I_{11}					t'	r'
I_1	I_2						
n1	spr	-1	1	2	2	-2	-2
n1	N1	-1	1	1	-1	-1	1
n1	N3	-1	1	-1	1	1	-1
n2	spr	1	-1	2	2	-2	-2
n2	N1	1	-1	1	-1	-1	1
n2	N3	1	-1	-1	1	1	-1
n3	spr	1	-1	2	2	-2	-2
n3	N1	1	-1	1	-1	-1	1
n3	N3	1	-1	2	2	1	-1

Tabela 2.3: Efekt zastosowania transformacji Harsányiego wykorzystującej strategie zredukowane do gry w postaci ekstensywnej na Rysunku 2.1.

krokowych – postać sekwencyjną. Gra w tej postaci jest również reprezentowana macierzami wypłat, po jednej dla każdego gracza, jednak nie jest to postać normalna i wymiary macierzy reprezentacji są jedynie liniowe względem liczby węzłów w drzewie gry. Postać sekwencyjna bazuje na postaci ekstensywnej, ale pozwala na łatwiejsze opisanie gry w programie liniowym. Zdefiniujmy zbiór wszystkich możliwych sekwencji ruchów gracza i:

$$\Sigma_i = \{\sigma | \exists s \in \mathcal{S} \cup \mathcal{Z} \quad \sigma = \sigma_i(s)\}.$$

Dzięki założeniu, że każdy zbiór informacyjny ma unikatowe akcje, możemy w wygodny sposób identyfikować te zbiory przez dowolną z akcji.

Definicja 2.21. *Będziemy mówić, że sekwencje ruchów σ_l i σ_f graczy l i f są* zgodne*, jeśli istnieje w drzewie gry taki liść $z \in \mathcal{Z}$, dla którego $\sigma_l() = \sigma_l$ i $\sigma_f(z) = \sigma_f$.*

Innymi słowy pary sekwencji zgodnych to dokładnie te pary, których zagranie prowadzi do jakiegoś liścia w drzewie gry. Stan wskazywany przez taką parę sekwencji będziemy oznaczać przez $Z(\sigma_l, \sigma_f)$.

Będziemy stosować oznaczenie $I_i(\sigma_i) = \{I \in \mathcal{I}_i | \sigma_i(I) = \sigma_i\}$.

Definicja 2.22. *Postacią sekwencyjną gry w postaci ekstensywnej [51, 82], będziemy nazywać macierze rzadkie $M_i \in \mathbb{R}^{\Sigma_l \times \Sigma_f}$, których wiersze etykietowane są ruchami lidera, a kolumny ruchami naśladowcy, wartości w komórkach wskazywanych przez pary sekwencji zgodnych są wartościami wypłat gracza i po zagraniu takiej pary sekwencji – $u_i(Z(\sigma_l, \sigma_f))$, a pozostałe wartości macierzy są zerami.*

Uwaga. Postać sekwencyjna istnieje tylko w powiązaniu z postacią ekstensywną, która dopuszczalne definiuje sekwencje ruchów.

Definicja 2.23. *Planem realizacji strategii behawioralnej [51] będziemy nazywać funkcję* $\psi : \Sigma_i \rightarrow [0,1]$ *spełniającą następujące warunki:*

1. $\psi(\varnothing) = 1$
2. $\psi(\sigma(I)) = \sum_{a \in A(I)} \sigma(\sigma(I)a)$*, dla każdego zbioru informacyjnego* I *z* $\mathcal{I}_i$*.*

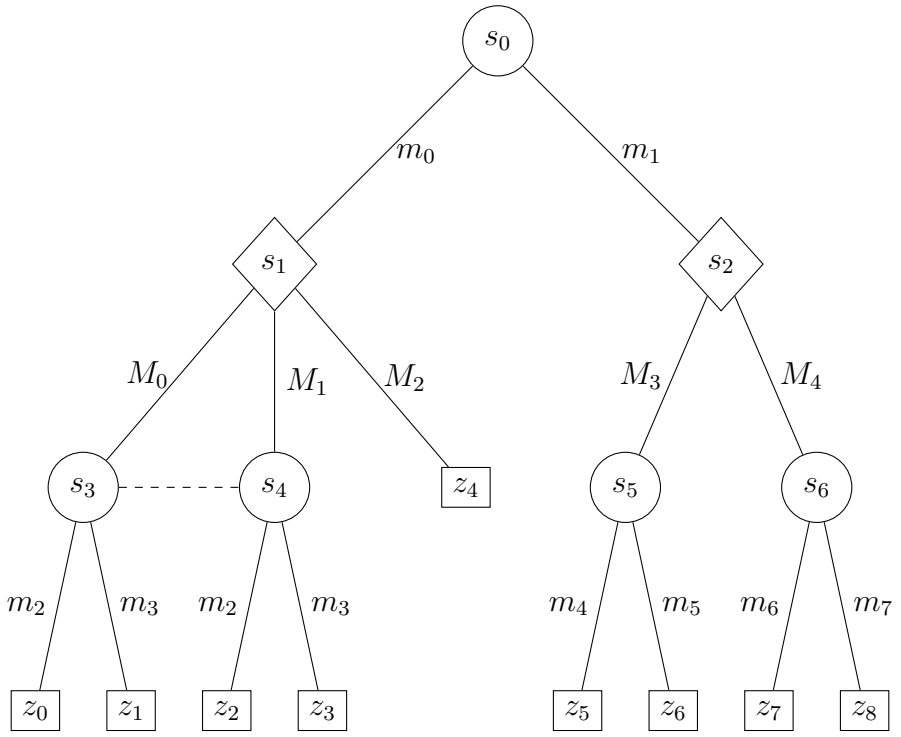
Notacja $\sigma(I)a$ *oznacza sekwencję* $\sigma(I)$ *z dopisanym na końcu ruchem* a*.*

Intuicyjnie powyższa definicja oznacza sieć przepływu prawdopodobieństw wyboru danych sekwencji, gdzie prawdopodobieństwo wyboru „rozlewa" się na prawdopodobieństwa sekwencji wydłużonych o jeden ruch. Za chwilę pokażemy że taki „przepływ" jest równoważny strategiom behawioralnym. Pokażemy też sposób w jaki można uzyskać wynik gry wykorzystując macierz z Definicji 2.22. Aby dobrze zobrazować naturę planu realizacji, rozważmy nieco bardziej skomplikowaną grę niż poprzednio. Rysunek 2.2 przedstawia grę dla dwóch graczy w postaci ekstensywnej. Gra składa się z siedmiu węzłów nieterminalnych i dziewięciu terminalnych. Stany nieterminalne oznaczamy jako s_0,$s_1, \ldots, s_6$, a stany terminalne jako $z_0, \ldots, z_8$. W korzeniu, s_0, pierwszy gracz wybiera ruch m_0 lub m_1. Ruch m_0 prowadzi do stanu s_1, w którym drugi gracz wybiera z trzech ruchów M_0, M_1, M_2. Ruchy M_0 i M_1 prowadzą odpowiednio do stanów s_3 i s_4. Oba te stany należą do jednego zbioru informacyjnego. Ruch M_2 prowadzi do stanu terminalnego z_4. W stanach s_3 i s_4, które znajdują się w jednym zbiorze informacyjnym gracz pierwszy wybiera spośród ruchów m_2 i m_3, ze stanu s_3 te ruchy prowadzą do stanów terminalnych odpowiednio z_0 i z_1, a z s_4 do z_2 i z_3. W stanie s_2 decyzję podejmuje gracz drugi. Do wyboru ma dwa ruchy: M_3 i M_4. Ruchy te prowadzą odpowiednio do stanów s_5 i s_6, w których ruch wykonuje pierwszy gracz. W każdym z tych stanów są dostępne dwa ruchy, odpowiednio m_4 i m_5 oraz m_6 i m_7, które prowadzą do stanów terminalnych z_5, z_6, z_7 i z_8. Zbiory informacyjne, podobnie jak w poprzednim przykładzie, będziemy indeksować najniższym indeksem stanu znajdującego się w tym zbiorze.

Rozważmy strategię behawioralną pierwszego gracza, która w każdym zbiorze informacyjnym oprócz I_5 wybiera każdy z ruchów z prawdopodobieństwem $0{,}5$, a w I_5 ruch m_4 jest wybierany z prawdopodobieństwem $0{,}6$, a ruch m_5 — $0{,}4$. Strategia ta jest podsumowana w Tabeli 2.4a. Możemy zdefiniować plan realizacji, który opisze dokładnie te same prawdopodobieństwa ruchów. Taki plan jest zaprezentowany w Tabeli 2.4b. Dodatkowo na Rysunku 2.4 zaprezentowane są ograniczenia na sumy prawdopodobieństw poszczególnych sekwencji ruchów wynikające z ograniczeń w definicji planu realizacji.

Obserwacja 2.1. *Każdemu planowi realizacji strategii behawioralnej odpowiada pewna strategia behawioralna i na odwrót [82, Definicja 3.3, Propozycja 3.4].*

Powyższa obserwacja oznacza, że plan realizacji strategii behawioralnej jest po prostu sposobem zapisu strategii behawioralnej, który tylko przy użyciu ograniczeń liniowych pozwala łatwo uzyskać prawdopodobieństwo osiągnięcia liścia w drzewie gry. To powoduje, że postać



Rysunek 2.2: Przykład gry w postaci ekstensywnej. Węzły w kształcie koła oznaczają stany w których rusza się pierwszy gracz, romby to stany, w których rusza się drugi gracz. Prostokąty to stany terminalne. Węzły połączone przerywaną linią należą do tego samego zbioru informacyjnego. Etykiety na krawędziach to dostępne ruchy.

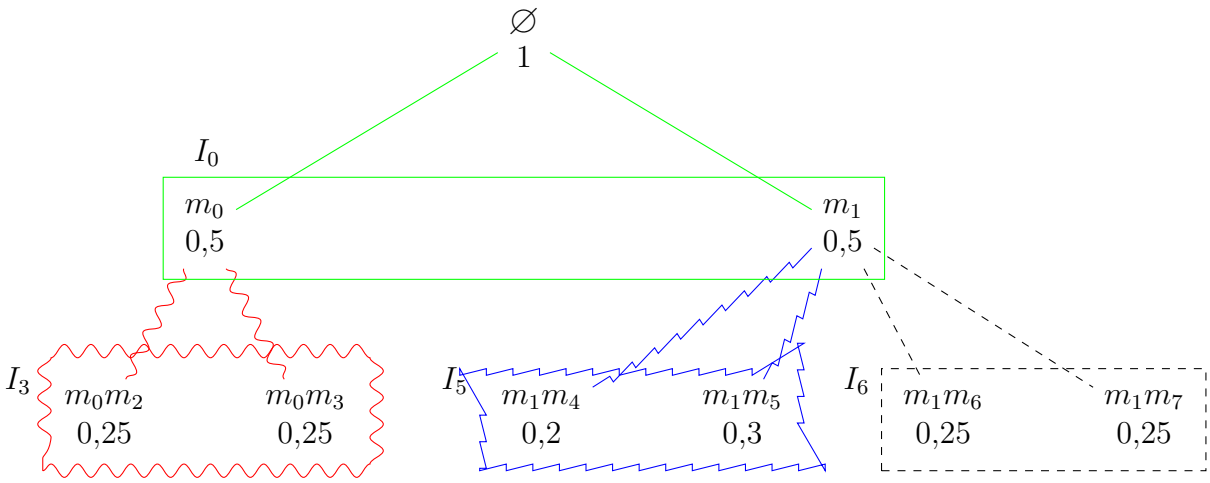
$\mathcal{I}_1$	
I_0	$m_0 \mapsto 0{,}5; m_1 \mapsto 0.5$
I_3	$m_2 \mapsto 0{,}5; m_3 \mapsto 0{,}5$
I_5	$m_4 \mapsto 0{,}6; m_5 \mapsto 0{,}4$
I_6	$m_6 \mapsto 0{,}5; m_7 \mapsto 0{,}5$

(a) Strategia behawioralna.

σ_i	$\psi(\sigma_i)$
$\varnothing$	1
m_0	0,5
m_0m_2	0,25
m_0m_3	0,25
m_1	0,5
m_1m_4	0,3
m_1m_5	0,2
m_1m_6	0,25
m_1m_7	0,25

(b) Plan realizacji

Tabela 2.4: Przykładowa strategia behawioralna pierwszego gracza i i odpowiadający jej plan realizacji w grze w Rysunku 2.2.



Rysunek 2.3: Wizualizacja sum z ograniczenia 2. z Definicji 2.23 w planie realizacji opisanym Tabelą 2.4b. Każdy kształt i kolor ramki opisuje elementy następujące po napotkaniu konkretnego zbioru informacyjnego. Strzałki w odpowiadających kształtach i kolorach wskazują na sekwencję ruchów poprzedzającą dany zbiór informacyjny. Prawdopodobieństwa w ramce sumują się do prawdopodobieństwa ruchu poprzedzającego.

sekwencyjna jest używana w metodach opartych o programowanie liniowe spotykanych w literaturze i opisywanych w Rozdziale 3 niniejszej rozprawy.

2.5 Stan Równowagi Stackelberga

Koncepcja stanu równowagi w teorii gier polega na zdefiniowaniu profilu strategii, który przy pewnych założeniach będzie optimum lokalnym funkcji wypłat każdego z graczy. Najbardziej znanym stanem równowagi rozważanym w teorii gier jest Równowaga Nasha [65]. Definicje stanów równowagi pozwalają określić co jest racjonalnym, spodziewanym zachowaniem graczy, przy odpowiednich założeniach dotyczących konstrukcji gry i dostępu do informacji. W tym podrozdziale w pierwszej kolejności zbudujemy ogólną intuicję na temat stanów równowagi na przykładzie prostszej koncepcji – Równowagi Nasha, następnie wprowadzimy podstawową definicję Równowagi Stackelberga. W dalszej części omówimy uszczegółowienia definicji stanu równowagi i przeanalizujemy uproszczenia definicji po ograniczeniu się tylko do gier skończonych.

Definicja 2.24. Równowaga Nasha *[65, Rozdział 60.]. Dana jest pewna gra. Niech* $u_i : \Pi \to \mathbb{R}$ *będzie funkcją wypłaty* i*-tego gracza. Profil strategii (prostych)* $\pi^* \in \Pi$ *będziemy nazywać stanem Równowagi Nasha w strategiach podstawowych, jeśli*

$$\forall i \in N \quad \forall \pi_i \in \Pi_i \quad u_i(\pi^*) \geqslant u_i(\pi^*_{-i}, \pi_i),$$

gdzie π^*_{-i} *oznacza krotkę strategii wszystkich oprócz* i*-tego gracza z profilu strategii* π^**. Innymi słowy dany profil strategii jest stanem równowagi, jeśli żaden z graczy nie może powiększyć*

swojej wypłaty, jeśli tylko on zmieni swoją strategię. W analogiczny sposób definiujemy stan Równowagi Nasha w strategiach mieszanych.

Przykład Równowagi Nasha. Rozważmy grę o sumie zerowej, zadaną następującą macierzą wypłat:

$$M = \begin{array}{cccc} & a & b & c \\ A & 5 & 4 & 3 \\ B & 8 & 1 & 3 \end{array}.$$

W tej grze stanem Równowagi Nasha w strategiach prostych jest profil (A, c). Wypłata gracza kolumnowego wynosi -3, a wierszowego 3. Żaden z graczy nie może poprawić swojego wyniku poprzez zmianę swojej decyzji. Zmiana decyzji gracza wierszowego co prawda nie przynosi mu straty, ale nie przynosi mu również korzyści, przez co punkt ten w myśl definicji 2.24 jest stanem równowagi.

W grze może występować więcej niż jeden stan Równowagi Nasha, co więcej każdy z tych stanów może dawać graczom inne wypłaty w grach. Rozważmy grę o sumie niezerowej, zadaną odpowiednio macierzą wypłat gracza wierszowego M_r i gracza kolumnowego M_c

$$M_r = \begin{array}{cccc} & a & b & c \\ A & 5 & 4 & 3 \\ B & 7 & 1 & 2 \end{array} \qquad M_c = \begin{array}{cccc} & a & b & c \\ A & 0 & 1 & 3 \\ B & 4 & 1 & 0 \end{array}.$$

W tej grze istnieją dwa profile strategii podstawowych będących stanami Równowagi Nasha: $\pi' = (B,a)$ i $\pi'' = (A,c)$. W obu przypadkach pojedynczy gracz zmieniając tylko swoją decyzje obniży swój wynik. Jednocześnie wypłaty graczy i suma wypłat znacząco różnią się pomiędzy każdym z rozwiązań. W stanie π' gracze otrzymują odpowiednio 7 i 4, suma 11. W stanie π'' te wyniki wynoszą 3 i 3, suma 6. Pokazuje to, że Równowaga Nasha, ze względu na niekooperacyjną naturę niekoniecznie gwarantuje optymalne rozwiązania w sensie Pareto. W niektórych przypadkach, na przykład w powszechnie znanym dylemacie więźnia, nawet jednoznacznie określony stan Równowagi Nasha może być bardzo niekorzystny dla obu graczy.

W strategiach prostych jest również możliwe, że nie będzie istniał żaden profil strategii, który byłby stanem Równowagi Nasha. Przykładem może być gra, gdzie każdy z graczy odwraca monetę, i jeśli monety są ułożone tymi samymi stronami do góry, wygrywa pierwszy z nich, a w przeciwnym wypadku drugi. Dana jest macierz tej gry o sumie zerowej:

$$M = \begin{array}{ccc} & o & r \\ O & 1 & -1 \\ R & -1 & 1 \end{array}.$$

W tej grze, w przypadku profili strategii (O, o) i (R, r), graczowi kolumnowemu opłaca się zmienić decyzję na przeciwną. W przypadku dwóch pozostałych profili decyzję chce zmienić gracz wierszowy. W przypadku, gdy zaczniemy rozważać strategie mieszane, nie jest to dłu-

żej prawdą. Każda gra skończona ma stan Równowagi Nasha w strategiach mieszanych [65]. W przypadku przytoczonej gry będzie to profil, gdzie każdy z graczy decyduje się na każdy ze swoich ruchów z prawdopodobieństwem 50%. Jeśli jeden z graczy zdecyduje się na właśnie taką strategię, to łatwo można pokazać, że niezależnie od wyboru drugiego gracza, wartość oczekiwana wypłaty wyniesie 0 dla każdego z graczy. W związku z tym przy proponowanym profilu strategii zmiana strategii jednego z graczy nie przynosi mu wzrostu wypłaty, co spełnia wymagania Równowagi Nasha.

Istotną cechą Równowagi Nasha jest symetria: żaden z graczy nie ma wyróżnionej pozycji i zmiana strategii jest rozpatrywana tak samo. Symetria jest cechą, która nie odpowiada wielu rzeczywistym problemom postawionym w poprzednim rozdziale, ponadto prowadzi ona do wielu różnych stanów równowagi bez mechanizmu pozwalającego wybrać graczom najlepszy z nich.

Przedmiotem tej rozprawy jest równowaga Stackelberga, w której istotna jest asymetria graczy. Nazwa Równowaga Stackelberga (Stackelberg Equilirbium, SE) upowszechniła się w literaturze dopiero w XXI wieku. We wcześniejszych pracach dużo częściej stosowany był termin Leadership Equilirbium, który opisywał podobne układy, (autorowi rozprawy nie jest znany żaden polski odpowiednik tej nazwy), chociaż nazwa Stackelberg też się pojawiała, na przykład w pracy [54].

Definicja 2.25. Równowaga Stackelberga *[54, 81]. Dana jest pewna gra dwuosobowa. Niech zbiór graczy $N = \{l,f\}$, gracza l będziemy nazywać liderem, a gracza f naśladowcą (ang. follower). Jeśli istnieje* jednoznacznie *określona funkcja najlepszej odpowiedzi (ang. best response) naśladowcy $BR : \Pi_l \to \Pi_f$, taka że*

$$\forall \pi_l \quad \forall \pi_f \qquad u_f(\pi_l, BR(\pi_l)) \geqslant u_f(\pi_l, \pi_f), \tag{2.4}$$

to profil strategii $\pi^* = (\pi_l^*, \pi_f^*) \in \Pi$ będziemy nazywać stanem Równowagi Stackelberga, jeśli oba następujące warunki są spełnione:

$$\pi_f^* = BR(\pi_l^*) \tag{2.5}$$

$$\forall \pi_l \in \Pi_l \quad u_l(\pi_l^*, \pi_f^*) \geqslant u_l(\pi_l, BR(\pi_l)) \tag{2.6}$$

Analogicznie można zdefiniować stan równowagi dla strategii mieszanych.

Powyższa definicja oznacza, że stanem równowagi jest para strategii, gdzie strategia naśladowcy jest najlepszą odpowiedzią (maksymalizującą wypłatę) naśladowcy, a lider nie może zaproponować innej strategii, która dałaby mu lepszy wynik, przy założeniu optymalnej odpowiedzi naśladowcy, tym razem przeciwko nowej strategii lidera. Bardzo ważny w tej definicji jest fakt, że jest ona określona tylko dla gier, gdzie jest jednoznacznie określona funkcja BR. Jednoznaczność ta oznacza, że nie może być sytuacji, gdzie naśladowca może zagrać kilka różnych strategii, które dadzą mu maksymalna możliwą wypłatę przeciwko zadanej strategii li-

dera. Rozważmy grę, gdzie lider jest graczem wierszowym, a naśladowca kolumnowym, zadaną przez następującą macierz wypłat lidera:

$$M_l = \begin{array}{cccc} & a & b & c \\ A & 1 & 3 & 5 \\ B & 3 & 1 & 1 \end{array}$$

oraz macierz wypłat naśladowcy

$$M_f = \begin{array}{cccc} & a & b & c \\ A & 3 & 1 & 3 \\ B & 4 & 1 & 2 \end{array}$$

W przypadku tej gry funkcja BR nie jest jednoznacznie określona, istnieją dwie strategie, które spełniają warunek (2.4):

$$\begin{cases} BR'(A) = a \\ BR'(B) = a \end{cases} \tag{2.7}$$

oraz

$$\begin{cases} BR''(A) = c \\ BR''(B) = a \end{cases} \tag{2.8}$$

W zależności od wybranego BR stan Równowagi Stackelberga byłby parą (B, a), gdy naśladowca podejmuje decyzje jak w definicji BR' lub (A, c), jeśli naśladowca podejmuje decyzje jak w definicji BR''.

W związku z powyższym ograniczeniem podstawowej definicji równowagi do bardzo wąskiej klasy gier, autorzy stosowali różne ujednoznacznienia stanu równowagi w przypadkach remisów, czyli sytuacji gdzie naśladowca ma do wyboru kilka strategii będących kandydatami na optymalną odpowiedź. W następujących akapitach omówimy podejścia do uogólnienia definicji stanu równowagi na gry, gdzie naśladowca ma do czynienia z remisami. Po zaprezentowaniu podejść stosowanych w literaturze zostanie wprowadzona definicja Silnej Równowagi Stackelberga, która jest wykorzystywana w rozprawie.

Już we wspomnianej wcześniej pracy Maschlera z 1966 roku [61] autor w opisie Procedury A definiującej sposób wyliczania odpowiedzi na strategię Inspektora, który jest liderem w rozważanym tam modelu, podkreśla kapitalikami, że naśladowca spośród wszystkich strategii optymalizujących jego wypłatę, wybierze tę strategię, która maksymalizuje wynik lidera. Następnie w Twierdzeniu 3.1 autor dowodzi, że w takiej sytuacji wypłaty graczy w stanie równowagi będą określone jednoznacznie. Nie oznacza to, że istnieje dokładnie jeden profil strategii spełniający taką definicję stanu równowagi. Wspomniane Twierdzenie 3.1 mówi, że zawsze istnieje *przynajmniej jeden* taki profil. W końcowej części swojej pracy Maschler rozważa także sytuację, gdy zakładamy, że naśladowca wybiera dowolną ze strategi optymalizujących jego

wynik, w nieznany liderowi sposób, wskazuje że w grze tam rozważanej, (która nie jest przypadkiem ogólnym), w takim przypadku dla dowolnego $\varepsilon > 0$ można zaproponować strategię, w której wypłata lidera będzie w przedziale $[u_l - \varepsilon, u_l]$ niezależnie od sposobu rozstrzygania remisów przez naśladowcę.

Do rozważania problemów rozstrzygania remisów przez naśladowcę wrócił w 1978 roku Leitmann, w pracy [54]. Leitmann wprowadza definicję uogólnionej strategii Stackelberga (Definicja 2.1 we wspomnianej pracy) – strategii lidera, która optymalizuje wypłatę lidera przy założeniu, że naśladowca wybierze możliwie niekorzystną dla niego opcję spośród optymalnych odpowiedzi naśladowcy. (Użycie słowa optymalizacja, zamiast maksymalizacja, jak w większej części rozprawy podyktowane jest faktem, że Leitmann dążył w swojej grze do minimalizacji kosztu, który był wynikiem gry). Istotny jest też fakt, że Leitmann oddzielnie wprowadza pojęcie uogólnionej strategii Stackelberga, a dopiero w dalszej Definicji 2.2 definiuje parę Strategii będącą stanem równowagi. Taki sposób definicji wynika z faktu, że naśladowca może w dalszym ciągu mieć niejednoznaczną optymalną odpowiedź, ale to którą z nich wybierze nie będzie miało wpływu na wynik gry. Ta obserwacja jest istotna dla tworzenia metod obliczeniowych poszukujących stanu Równowagi Stackelberga i będzie dyskutowana w dalszej części rozprawy.

Definicje używane współcześnie zostały prawdopodobnie sformalizowane po raz pierwszy przez trójkę autorów: Breton, Alj, Haurie w roku 1988 [20]. Poniżej przedstawiamy definicje zaproponowane przez autorów cytowanej wyżej pracy. Całość jest przedstawiona tak jak w oryginale, ze zmienionymi oznaczeniami, dostosowanymi do tych używanych w rozprawie. W szczególności autorzy rozważali również gry nieskończone, co oznacza, że w pewnych definicjach stosowali supremum zbiorów, podczas gdy w grach skończonych istnieje maksimum.

W dalszej części tego podrozdziału zostaną opisane zmiany i uproszczenia, które można zastosować. Zdefiniujmy funkcję f, która dowolnej strategii $\delta_l \in \Delta_l$ przyporządkowuje supremum wartości wypłat, które może uzyskać naśladowca:

$$f(\delta_l) = \sup_{\delta_f \in \Delta_f} u_f(\delta_l, \delta_f) \tag{2.9}$$

z użyciem tej funkcji możemy zdefiniować ε-optymalną odpowiedź naśladowcy. Dla dowolnego $\varepsilon \geqslant 0$ poniższa funkcja zdefiniuje zbiór wszystkich strategii naśladowcy, których wynik nie jest gorszy od supremum pomniejszonego o ε:

$$\overline{BR}(\delta_l, \varepsilon) = \{\delta_f \in \Delta_f | u_f(\delta_l, \delta_f) \geqslant f(\delta_l) - \varepsilon\} \tag{2.10}$$

Definicja 2.26. Silna (Słaba) ε-Równowaga Stackelberga *(Definicja 2.1 w [20]). Dla danego* $\varepsilon \geqslant 0$ *stanem Silnej (Słabej)* ε*-Równowagi Stackelberga będziemy nazywać parę* (δ_l^*, δ_f^*)*, taką*

że:

$$\textit{(i)} \qquad \delta_f^* \in \overline{BR}(\delta_l^*, \varepsilon) \qquad \textit{oraz} \tag{2.11}$$

$$\textit{(ii)} \qquad u_l(\delta_l^*, \delta_f^*) \geqslant \sup_{\delta_l \in \Delta_l} \sup_{\delta_f \in \overline{BR}(\delta_l, \varepsilon)} u_l(\delta_l, \delta_f) - \varepsilon \qquad \textit{(silna Równowaga)} \tag{2.12}$$

$$\textit{albo} \qquad u_l(\delta_l^*, \delta_f^*) \geqslant \sup_{\delta_l \in \Delta_l} \inf_{\delta_f \in \overline{BR}(\delta_l, \varepsilon)} u_l(\delta_l, \delta_f) - \varepsilon \qquad \textit{(słaba Równowaga)} \tag{2.13}$$

Powyższa definicja mówi, że w wariancie silnym naśladowca będzie rozstrzygał remisy na korzyść lidera, a w wariancie słabym na jego niekorzyść. Dla wartości $\varepsilon = 0$ definicja silnej równowagi sprowadza się do definicji używanej przez Maschlera [61], a definicja słabej równowagi do definicji uogólnionej równowagi wprowadzonej przez Leitmanna [54].

Definicja 2.27. *Powyższy stan równowagi dla $\varepsilon = 0$ będziemy nazywać po prostu Silną (odpowiednio Słabą) Równowagą Stackelberga.*

Obserwacja 2.2. *W grach, gdzie istnieje Stan Równowagi Stackelberga według definicji 2.25, definicje 2.25 i 2.27 (zarówno Silna jak i Słaba wersja) są równoważne.*

Przypadek Silnej (Słabej) Równowagi Stackelberga w grach skończonych. W niniejszej rozprawie ograniczamy się wyłącznie do gier skończonych, to jest takich, gdzie zbiory akcji, więc i strategi podstawowych, graczy są skończone. W takiej sytuacji podane definicje można uprościć, biorąc pod uwagę, że w każdym zbiorze skończonym istnieje maksimum (odpowiednio minimum), w związku z czym Definicję 2.26 można zatem zapisać jak poniżej.

Obserwacja 2.3. *W przypadku rozważania tylko strategii prostych w grach skończonych Równanie (2.9) można zapisać jako:*

$$f(\delta_l) = \max_{\delta_f \in \Delta_f} u_f(\delta_l, \delta_f),$$

a definicja 2.26 jest równoważna następującej:

$$\textit{(i)} \qquad \delta_f^* \in \overline{BR}(\delta_l^*, \varepsilon) \qquad \textit{oraz}$$

$$\textit{(ii)} \qquad u_l(\delta_l^*, \delta_f^*) \geqslant \max_{\delta_l \in \Delta_l} \max_{\delta_f \in \overline{BR}(\delta_l, \varepsilon)} u_l(\delta_l, \delta_f) - \varepsilon \qquad \textit{(silna równowaga)}$$

$$\textit{albo} \qquad u_l(\delta_l^*, \delta_f^*) \geqslant \max_{\delta_l \in \Delta_l} \min_{\delta_f \in \overline{BR}(\delta_l, \varepsilon)} u_l(\delta_l, \delta_f) - \varepsilon \qquad \textit{(słaba równowaga)}$$

Obserwacja 2.4. *Na podstawie powyższej obserwacji można łatwo pokazać, że Stan Rozważając profile strategii będące stanem Silnej Równowagi Stackelberga można zaobserwować jeszcze jedną własność tej równowagi w strategiach mieszanych, która umożliwiła konstrukcję efektywnych algorytmów znajdujących stan równowagi. Jak wskazano wcześniej $u_f(\delta_l, \delta_f)$ jest kombinacją wypukłą punktów będących wynikami gry dla profili strategii prostych. Ta obserwacja*

pozwala na ograniczenie zbioru potencjalnych strategii naśladowcy tylko do strategii prostych. Na podstawie powyższej obserwacji można łatwo pokazać, że Stan Silnej (Słabej) 0*-Równowaga Stackelberga w strategiach podstawowych zawsze istnieje w grach skończonych [20].*

Podobnie, zostało pokazane, że taka sama własność zachodzi w strategiach mieszanych:

Twierdzenie 2.2. *W każdej grze skończonej istnieje stan Silnej Równowagi Stackelberga w strategiach mieszanych [11, Twierdzenie 3.3].*

Przedmiotem tej rozprawy będzie heurystyczna aproksymacja strategii lidera w stanie Silnej Równowagi Stackelberga w strategiach mieszanych, która w tak zaproponowanej klasie gier zawsze istnieje. W związku z tym możemy uprościć podane wcześniej definicje:

Obserwacja 2.5. *W grach skończonych funkcję* f *zadaną jak w równaniu (2.9) możemy równoważnie zapisać jako:*

$$f(\delta_l) = \max_{\delta_f \in \Delta_f} u_f(\delta_l, \delta_f), \tag{2.14}$$

Na podstawie przedstawionego rozumowania, można stwierdzić, że dla dowolnego gracza i *oraz ustalonego* δ_l *wartość* $\hat{u_i}(\delta_f) = u_i(\delta_l, \delta_f)$ *jest kombinacją wypukłą punktów* $\{u_i(\delta_l, \pi_f) | \pi_f \in \Pi_f\}$*, a co za tym idzie przeciwobraz* $\hat{u_i}(\Delta_f)$ *ma maksimum i minimum.*

Definicja 2.28. Silna Równowaga Stackelberga *w grach skończonych. W grze skończonej stan Silnej Równowagi Stackelberga będziemy definiować jako parę strategii* (δ_l, δ_f)*, spełniającej następujące warunki:*

$$\delta_f^* \in \overline{BR}(\delta_l^*) \tag{2.15}$$

$$u_l(\delta_l^*, \delta_f^*) \geqslant \max_{(\delta_l, \delta_f) | \delta_l \in \Delta_l, \delta_f \in \overline{BR}(\delta_l, 0)} u_l(\delta_l, \delta_f) \tag{2.16}$$

gdzie $f(\delta_l) = \max_{\delta_f \in \Delta_f} u_f(\delta_l, \delta_f)$ *oraz* $\overline{BR}(\delta_l) = \{\delta_f \in \Delta_f | u_f(\delta_l, \delta_f) \geqslant f(\delta_l)$

Obserwacja 2.6. *We wszystkich stanach Silnej Równowagi Stackelberga w grach skończonych wartość wypłaty lidera jest taka sama [61, Twirdzenie 4.4].*

Twierdzenie 2.3. *W grach skończonych zawsze istnieje stan Silnej Równowagi Stackelberga w strategiach mieszanych, w którym strategia naśladowcy jest strategią prostą (strategią mieszaną, w której dokładnie jedna strategia ma prawdopodobieństwo jeden). (por. Twierdzenie 9 w [84])*

Powyższe twierdzenie jest kluczowe do konstrukcji wszystkich znanych metod algorytmicznego poszukiwania Równowagi Stackelberga. Dokładne metody bazują na siłowym przeszukaniu przestrzeni ruchów naśladowcy, co nie byłoby możliwe, jeśli przestrzeń ta byłaby nieskończona. Ten fakt jest jeszcze raz podkreślony jest podkreślone w Rozdziale 3.

W literaturze można spotkać również uogólnione warianty Równowagi Stackelberga, szczególnie popularne są:

- gra, w której gra jednocześnie $n >= 1$ naśladowców, naśladowcy znając decyzję lidera wybierają w grze rozgrywanej między sobą optymalną odpowiedź w formie stanu Równowagi Nasha, w grze z ustaloną strategią lidera. Takie podejście jest rozważane, na przykład w pracach [54, 3, 10], w części prac ten stan równowagi jest określony stanem Równowagi Stackelberga-Nasha (Stackelberg-Nash Equilibrium),

- gra w której możliwych jest wielu naśladowców, ale gra tylko jeden z nich. Wybór naśladowcy następuje przed rozpoczęciem gry ze z góry ustalonego rozkładu. Lider zna tylko rozkład, z którego wybierani są naśladowcy, jednak nie wie który z nich został wybrany. Takie gry noszą nazwę Bayesowskich Gier Stackelberga (Bayesian Stackelberg Games) i są szczególnie popularne w obszarze Security Games, gdzie poszukiwane są optymalnych zachowania w sytuacji, gdy spodziewamy się jednego z kilku rodzajów ataku, na przykład próby przemytu narkotyków lub broni, znane są z doświadczenia częstotliwość poszczególnych typów przemytu, ale nie wiadomo, który przemytnik pojawi się w danym momencie na granicy [73, 49, 69, 2]. Ogólne definicje gier Bayesowskich, niezwiązane bezpośrednio z Równowagą Stackelberga, można znaleźć w pracy Harsányiego[35].

W związku z tym, że metody obliczeniowe proponowane z tej rozprawie są dedykowane wariantowi SSE z jednym naśladowcą, nie podajemy ścisłych definicji tych gier i Równowagi Stackelberga w tych wariantach.

2.5.1 Relacja do dwupoziomowych problemów optymalizacyjnych

Problem wyliczania Równowagi Stackelberga jest szczególnym przypadkiem dwupoziomowego problemu optymalizacyjnego (ang. bi-level optimization problem) [27]. Wielopoziomowe, w szczególności dwupoziomowe, problemy optymalizacyjne są często rozważane w literaturze z obszaru badań operacyjnych [80]. Wielopoziomowym problemem optymalizacyjnym nazywamy problem optymalizacyjny, w którym wartość optymalizowanej funkcji zależy od zmiennej decyzyjnej i od wartości będącej rozwiązaniem problemu optymalizacyjnego, którego parametrem jest zmienna decyzyjna problemu zewnętrznego. Najprostszymi przypadkami są problemy dwupoziomowe, w ogólnej postaci:

$$\min_x f(x, \min_y(g(x,y))).$$

Można wyróżnić cztery rodzaje takich problemów, ze względu na kierunek optymalizacji: min-max, max-min, max-max, min-min, przy czym nie są to klasy istotnie różne. Problem z jednej klasy można zawsze przekształcić na problem z dowolnej innej klasy poprzez dopisanie minusa przed odpowiednią optymalizowaną funkcją.

W Definicji 2.28 definiujemy Równowagę Stackelberga jako:

$$u_l(\delta_l^*, \delta_f^*) \geqslant \max_{(\delta_l,\delta_f)|\delta_l\in\Delta_l,\delta_f\in\overline{BR}(\delta_l,0)} u_l(\delta_l, \delta_f),$$

gdzie $f(\delta_l) = \max_{\delta_f\in\Delta_f} u_f(\delta_l, \delta_f)$ oraz $\overline{BR}(\delta_l) = \{\delta_f \in \Delta_f | u_f(\delta_l, \delta_f) \geqslant f(\delta_l)$. Optymalną wartość wypłaty lidera możemy na tej podstawie wyznaczyć w sposób następujący:

$$u_l^* = \max_{\delta_l} u_l \left(\delta_l, \arg\max_{\delta_f} u_f(\delta_l, \delta_f) \right),$$

to znaczy w postaci dwupoziomowego problemu optymalizacyjnego właśnie. Pokazuje to, że możemy patrzeć na wyznaczanie Równowagi jako na taki problem optymalizacyjny.

Prace z obszaru badań operacyjnych nie zawsze stosują terminologię znaną z teorii gier, jednak autorzy często zauważają, że postawiony przez nich problem dwupoziomowy można interpretować jako poszukiwanie pewnej Równowagi Stackelberga [14, 22, 80]. Prace w tym obszarze stosują techniki MILP to rozwiązywania postawionych problemów. Istotnym wkładem prac z obszaru rozwiązywania problemów dwupoziomowych jest wskazanie szczególnych klas problemów, które mogą zostać sprowadzone do problemów jednopoziomowych [80]. Szczegółowy opis tych technik jest jednak poza zakresem tej rozprawy i nie będzie tu omówiony. Autor chce jednak zaznaczyć, że przykładem takiego przekształcenia jest rozwiązywanie problemu SMOS o sumie zerowej za pomocą programu liniowego, opisane w Rozdziale 3.5.

2.6 Cechy czyniące poszukiwanie Równowagi Stackelberga trudnym

Problem poszukiwania (Silnej) Równowagi Stackelberga możemy potraktować jako problem optymalizacyjny. W tak postawionym problemie zbiorem rozwiązań dopuszczalnych będzie zbiór wszystkich strategii lidera, natomiast wartością takiego rozwiązania będzie wypłata lidera, gdy ta strategia zostanie zagrana przeciwko optymalnej odpowiedzi naśladowcy. Optymalne rozwiązanie tak postawionego programu będzie strategią lidera w profilu strategii będącym stanem Równowagi Stackelberga.

Prace dotyczące efektywnego obliczania równowagi w dużych grach wielokrokowych z niepełną informacją zaczęły się pojawiać dopiero w ostatnich piętnastu latach. Wynika to przede wszystkim z faktu, że do czasu zaproponowania postaci sekwencyjnej, główną alternatywą była transformacja Harsányiego, która sama w sobie powodowała wykładniczy wzrost złożoności problemu. Mimo stosunkowo krótkiego okresu, w którym problem był badany, znane są obszerne wyniki dotyczące złożoności problemu poszukiwania Równowagi Stackelberga w tych grach.

Złożoność problemu poszukiwania stanu Równowagi Stackelberga została zbadana przez J.

Letchforda, który przedstawił wyniki swoich badań w rozprawie doktorskiej w 2013 roku [55]. Wspomniana rozprawa przedstawia analizę poszukiwania Równowagi Stackelberga w różnych klasach gier i prezentuje wielomianowe algorytmy lub dowody NP-trudności danego przypadku. Ogólnym wnioskiem z pracy Letchforda jest fakt że poza bardzo prostymi klasami gier poszukiwanie Równowagi Stackelberga jest problemem NP-trudnym. W szczególności autor w Twierdzeniu 12 dowodzi poprzez redukcję wielomianową problemu 3-SAT do poszukiwania Równowagi Stackelberga w wielokrokowej grze dla dwóch graczy o sumie niezerowej z niepełną informacją, że problem poszukiwania Równowagi Stackelberga w tej klasie gier jest problemem NP-trudnym.

Twierdzenie 2.4. *Poszukiwanie Silnej Równowagi Stackelberga jest problemem NP-trudnym [55, Twierdzenie 12.].*

Ponadto nie są znane żadne algorytmy aproksymacyjne (z gwarancją uzyskania zadanego przybliżenia w czasie wielomianowym dla ustalonego przybliżenia) w ogólnej klasie gier wielokrokowych z niepełną informacją. Dla tej klasy, w chwili pisania tej rozprawy, nie są również znane wyniki mówiące że takich algorytmów nie ma. Poniżej prezentujemy opublikowane wyniki dotyczące algorytmów aproksymujących Równowagę Stackelberga.

Definicja 2.29. *Algorytmem $f(n)$-aproksymacyjnym [6, rozdział 8.] będziemy nazywać algorytm wielomianowy rozwiązujący pewien problem optymalizacyjny O, taki że dla dowolnego zadania $o \in O$ rozwiązanie $\tilde{s}$ znajdowane przez ten algorytm jest dopuszczalnym rozwiązaniem o oraz s jest co najwyżej $O(f(n))$ razy gorsze on rozwiązania optymalnego s^*, gdzie n jest rozmiarem zadania o.*

Definicja 2.30. *Klasą problemów $f(n)$-APX [6, Definicja 8.2] będziemy nazywać zbiór takich problemów dla których istnieje algorytm $f(n)$-aproksymacyjny, w szczególności będziemy wyróżniać klasę Poly-APX, w której $f(n)$ jest dowolnym wielomianem.*

W środowisku zajmującym się grami Stackelberga i Security Games przeważa opinia, że poszukiwanie Równowagi Stackelberga w grach wielokrokowych z niepełną informacją nie jest w klasie Poly-APX, niektórzy przypuszczają również że nie istnieją algorytmy $f(n)$-aproksymacyjne dla żadnych $f(n)$. Według wiedzy autora, w chwili obecnej nie są znane żadne wyniki dotyczące przynależności do klasy Poly-APX poszukiwania Równowagi Stackelberga w wielokrokowych grach z niepełną informacją z jednym naśladowcą.

W ostatnim czasie pojawiły się wyniki dla kilku klas gier z wieloma naśladowcami.

- W jednokokowych Grach Stackelberga z wieloma naśladowcami, gdzie liczba naśladowców nie jest znana z góry, o sumie niezerowej nie jest w klasie Poly-APX. Taka sama własność została wykazana też dla Bayesowskich Gier Stackelberga (c.f. uogólnienia Równowagi Stackelberga na stronie 47) z wieloma naśladowcami. Oba wyniki są przedstawione w pracy [66]. Drugi wynik jest o tyle istotny, że jego konsekwencją jest fakt,

że gry wielokrokowe z niepełna informacją i losowym czynnikiem zewnętrznym (ang. chance moves lub nature moves) nie należą do klasy Poly-APX.

- Podobne wyniki uzyskany dla Gier Zatorowych Stackelberga (Stackelberg Congestion Games), w których mamy wielu naśladowców chcących korzystać z puli wspólnych zasobów. Im więcej graczy decyduje się skorzystać z danego zasobu, tym niższe będą wyniki graczy z niego korzystających. W przypadku, gdy naśladowcy mogą zająć dokładnie jeden zasób w grze, wykazano, że dla ogólnego przypadku, gdy nie ma dodatkowych ograniczeń na funkcję wypłat problem nie należy do klasy Poly-APX [59]. W roku 2019 wyniki te zostały rozszerzone o dowód, że jeśli ograniczymy się do monotonicznych funkcji wypłat, a dopuścimy wybór przez naśladowców więcej niż jednego zasobu to problem również nie należy do Poly-APX [58].

Fakt, że nie spodziewamy się istnienia efektywnych algorytmów aproksymacyjnych z gwarancją jakości rozwiązania, powoduje, że interesującym staje się rozwój metod miękkich opartych o metaheurystyki stosowane w sztucznej inteligencji takie jak przeszukiwanie Monte Carlo, algorytmy rojowe, czy ewolucyjne. Metody takie są w stanie dostarczyć aproksymacje rozwiązań bez gwarancji jakości, ale w zamian oferują dużo krótsze czasy obliczeń niż algorytmy dokładne. Jest to główną motywacją stojącą za powstaniem metod będących głównym przedmiotem tej rozprawy.

2.7 Odejście od pełnej racjonalności przeciwnika

Wszystkie rozważane dotychczas warianty Równowagi Stackelberga zakładały naśladowcę, który jest w pełni racjonalny i doskonale zna strategię lidera. To założenie może nie być poprawne, szczególnie jeśli mamy do czynienia z naśladowcami, którzy nie działają w zorganizowanych strukturach. W przypadkach działań przeciwko terrorystom i zorganizowanym grupom przestępczym założenie, że przeciwnicy skrupulatnie planują swoje działania jest prawdziwe i należy się spodziewać, że ich zagrania będą optymalne [79]. Nie wszystkie zastosowania obejmują jednak właśnie takie sytuacje. Jednym z przykładów są Zielone Gry Obronne (ang. Green Security Games, GSG) [32], które mają za zadanie pomóc w różnych działaniach mających na celu ochronę przyrody. W takich przypadkach przeciwnikami często są jednostki, takie jak kłusownicy, które nie prowadzą pełnego wywiadu, więc mogą nie znać w pełni strategii lidera. Ich decyzje mogą być nie w pełni racjonalne. Przykładem takiej nieracjonalności jest sytuacja, gdy kłusownicy działają w najczęściej patrolowanych rejonach, gdzie występują chronione gatunki, ze względu na szansę dużego zysku ze sprzedaży takiego zwierzęcia. Mimo, że prawdopodobieństwo złapania jest tak duże, że wartość oczekiwana takiej akcji jest wartością ujemną, kłusownik zdecyduje się na taką próbę, przyciągnięty wysokim zyskiem w przypadku powodzenia akcji, podobnie jak dzieje się to w przypadku udziału w loteriach i grach liczbowych.

Możliwość uwzględnienia w metodach poszukiwania Równowagi Stackelberga ne w pełni racjonalnych naśladowców jest dodatkowym atutem metod obliczeniowych. Głównym przedmiotem tej rozprawy są metody poszukiwania Równowagi Stackelberga z racjonalnym naśladowcą, jednak w Rozdziale 5.1 opisującym potencjalne dalsze kierunki badań wskazujemy, że wprowadzenie ograniczonej racjonalności do proponowanych metod jest możliwe i warte zbadania w przyszłości.

Rozdział 3

Istniejące podejścia do stanu Równowagi Stackelberga

Ten rozdział prezentuje metody stosowane do rozwiązywania Gier Stackelberga występujące w literaturze. Wszystkie przytoczone metody, oprócz jednej, bazują na programowaniu liniowym. Ten fakt powoduje, że w wielu przypadkach skalowalność pamięciowa metody jest niewystarczająca do stosowania jej w grach wielokrokowych. Wraz z prezentacją kolejnych podejść są prezentowane ich ograniczenia i trudności przy dostosowaniu do ogólniejszej postaci. Przy opisach technik, które stały się inspiracją do budowy metod wprowadzonych w Rozdziale 4 pojawiają się odpowiednie uwagi podkreślające ten fakt.

W celu zastosowania technik optymalizacyjnych, takich jak przytoczone już programowanie liniowe, należy sformułować poszukiwanie Równowagi Stackelberga jako problem optymalizacyjny. Natura Równowagi Stackelberga, gdzie naśladowca podejmuje decyzję znając już strategię lidera, powoduje że powstały problem optymalizacyjny jest dwupoziomowy. Co więcej, w ogólnym sformułowaniu lider maksymalizuje funkcję swojej wypłaty, a naśladowca swojej.

Poszukiwanie równowagi Stackelberga możemy wyrazić jako poszukiwanie profilu strategii mieszanych (δ_l, δ_f) będących optymalnym rozwiązaniem następującego dwupoziomowego problemu optymalizacyjnego:

$$\max_{\delta_l \in \Delta_l} u_l(\delta_l, \arg\max_{\delta_f \in \Delta_f} u_f(\delta_l, \delta_f)). \tag{3.1}$$

Na mocy Twierdzenia 2.3, możemy w wewnętrznym problemie optymalizacyjnym uwzględnić tylko strategie proste naśladowcy:

$$\max_{\delta_l \in \Delta_l} u_l(\delta_l, \arg\max_{\pi_f \in \Pi_f} u_f(\delta_l, \pi_f)). \tag{3.2}$$

Uwaga. Dla uproszczenia zapisu to sformułowanie nie zawiera rozstrzygania remisów z definicji Silnej Równowagi Stackelberga (Definicja 2.28). Będziemy jednak zakładać, że w przypadku remisów w wewnętrznym problemie optymalizacyjnym naśladowca optymalizuje wy-

płatę lidera.

3.1 Metoda rozwiązująca wiele programów liniowych

W roku 2006 pojawiła się pierwsza praca, która skupiała się właśnie na poszukiwaniu Równowagi Stackelberga metodami algorytmicznymi [24]. Przytoczona praca zawiera zarówno algorytmy dedykowane jednokrokowym grom z jednym naśladowcą, jak i grom z wieloma naśladowcami i grom bayesowskim. W tym miejscu przytoczymy tylko rozwiązanie dedykowane grom w postaci normalnej z jednym naśladowcą.

Definicja 3.1. Obszar najlepszej odpowiedzi *(ang. best response region) [88]. Dla każdej strategii prostej naśladowcy $\pi_f \in \Pi_f$ możemy zdefiniować zbiór strategii lidera $X^{\pi_f} \subseteq \Delta_l$ taki, że dla dowolnej strategii lidera z tego zbioru, π_f zagrane przeciwko tej strategii lidera daje wyższą wypłatę niż dowolna inna strategia naśladowcy, innymi słowyπ_f jest optymalną odpowiedzią na strategie lidera ze zbioru X^{π_f}. Formalnie ten warunek można wyrazić jako:*

$$X^{\pi_f} = \{\delta_l \in \Delta_l | \forall \pi'_f \in \Pi_f u_f(\delta_l, \pi'_f) \leqslant u_f(\delta_l, \pi'_f)\} \tag{3.3}$$

Twierdzenie 3.1. *Problem optymalizacyjny z równania (3.2) można równoważnie zapisać jako*

$$\max_{\pi_f \in \Pi_f} \max_{\sigma_l \in X^{\pi_f}} u_l(\sigma_l, \pi_f), \tag{3.4}$$

przy założeniu że w wewnętrznym problemie optymalizacyjnym w (3.2) naśladowca rozstrzyga remisy na korzyść lidera [88, Twierdzenie 3].

Powyższy fakt jest bazą techniki stosowanej w ogromnej większości metod rozwiązywania Gier Stackelberga. W omawianej metodzie dla każdej strategii naśladowcy π_f rozwiązywany jest program liniowy, który znajduje strategię $\delta_l \in X^{\pi_f}$, która daje najwyższą wypłatę lidera, gdy jest grana przeciwko π_f. Część programów może być sprzeczna, ale przynajmniej jeden musi mieć rozwiązanie, bo wiadomo, że istnieje Równowaga Stackelberga w tej grze (Twierdzenie 2.2). Takie podejście w literaturze bywa określane jako wiele programów liniowych (ang. Multiple-LP). Metoda ta prezentowana jest w Algorytmie 3.1.

Program liniowy wykorzystywany przez ten algorytm dla wybranego π_f jest następującej

Algorytm 3.1: Poszukiwanie Równowagi Stackelberga poprzez rozwiązywanie wielu programów liniowych [24].

```
1  u_l^* ← −∞;
2  δ_l^* ← nil;
3  π_f^* ← nil;
4  for π_f ∈ Π_f do
5      SolveLP(π_f)// Rozwiąż program liniowy (3.5)-(3.8)
6      if LPIsFeasible() then
7          (δ_f, u_l) ← GetLPSolution// Pobierz wartość funkcji celu i
               strategię mieszaną z rozwiązania programu
               liniowego
8          if δ_f > δ_f^* then
9              u_l^* ← u_l;
10             δ_l^* ← δ_l;
11             π_f^* ← π_f;
12         end
13     end
14 end
```

postaci:

$$\max_{p} \sum_{\pi_l \in \Pi_l} p_{\pi_l} u_l(\pi_l, \pi_f) \tag{3.5}$$

z zachowaniem warunków

$$\sum_{\pi_l \in \Pi_l} p_{\pi_l} u_f(\pi_l, \pi_f) \geqslant \sum_{\pi_l \in \Pi_l} p_{\pi_l} u_f(\pi_l, \pi'_f) \qquad \forall \pi'_f \in \Pi'_f \tag{3.6}$$

$$\sum_{\pi_l \in \Pi_l} p_{\pi_l} = 1 \tag{3.7}$$

$$p_{\pi_l} \geqslant 0 \qquad \forall \pi_l \in \Pi_l \tag{3.8}$$

W powyższym programie liniowym zmienne p_{π_i} definiują rozkład prawdopodobieństwa nad strategiami lidera, czyli strategie mieszaną lidera. Program maksymalizuje wypłatę lidera. Ograniczenia (3.7) i (3.8) wymuszają, żeby zmienne p definiowały rozkład prawdopodobieństwa, ograniczenie (3.6) wymusza przynależność strategii mieszanej definiowanej przez p do zbioru X^{π_f}. Uwaga. Mogą istnieć strategie naśladowcy, dla których program (3.5)—(3.8) jest sprzeczny, to oznacza tyle, że odpowiedni zbiór X dla tej strategii jest pusty.

Jest to najprostsze podejście algorytmiczne do Równowagi Stackelberga, jednak jego skalowalność jest silnie ograniczona. Główna pętla wykonuje się $|\Pi_f|$ razy, a program liniowy rozwiązywany w każdej iteracji ma $|\Pi_l|$ zmiennych i $|\Pi_f| + 2$ ograniczeń. W przypadku gier wielokrokowych, gdzie liczności zbiorów strategii są wykładnicze względem liczby rund w grze, praktycznie nie jest możliwe stosowanie tego rozwiązania.

Ważną obserwacją jest też fakt, że w takim sformułowaniu problemu, zewnętrznym pozio-

mem przeszukiwania są strategie naśladowcy, a wewnętrznym — strategie lidera. Jest to układ odwrotny do sformułowania Równowagi Stackelberga w Definicji 2.28.

3.2 Metoda DOBSS – wykorzystanie metody podziału i ograniczeń wbudowanej w solvery MILP

Kolejnym kamieniem milowym w rozwiązywaniu Gier Stackelberga jest metoda DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) [69, 68] opublikowana w 2008 roku. O ile opis metody w przytoczonych pracach zanurzony jest w zagadnienie z obszaru bezpieczeństwa publicznego, o tyle sama metoda działa dla dowolnej gry w postaci normalnej o sumie niezerowej. W przeciwieństwie do poprzedniej omówionej metody, która rozwiązywała wiele programów liniowych, ta rozwiązuje tylko jeden mieszany program liniowy (ang. Mixed Integer Linear Program, MILP). W porównaniu z poprzednim podejściem program jest trudniejszy zarówno ze względu na wielkość – zawiera więcej zmiennych i ograniczeń, jak i dopuszczalne wartości – część zmiennych w programie jest binarna, co czyni go dużo bardziej kosztownym do rozwiązania.

Inspiracją do budowy metody DOBSS była chęć eliminacji pełnego przeglądu strategii naśladowcy. Cel ten osiągnięto poprzez przeniesienie faktu uwzględnienia każdej strategii naśladowcy jako potencjalnie optymalnej w programie liniowym i dodanie rodziny zmiennych binarnych wybierających dokładnie jedną z nich jako wybraną strategię. Formalnie DOBBS służy do rozwiązywania gier Bayesowskich, które nie są przedmiotem tej rozprawy. W związku z tym prezentowane tu programy są uproszczone i nie zawierają elementów związanych z grami Bayesowskimi.

W dalszej części podrozdziału zaprezentujemy dwa programy matematyczne z pracy [69]. W pierwszej kolejności program z kwadratową funkcją celu, który nie jest parametryzowany żadną strategią naśladowcy i pozwala w jednym przebiegu znaleźć Równowagę Stackelberga, a następnie transformację tego programu, po której funkcja celu staje się liniowa.

Do programu z rozwiązania Multiple-LP, prezentowanego wyżej w równaniach (3.5)–(3.8), w pierwszej kolejności dodano rodzinę zmiennych $q_{\pi_f} \in \{0,1\}$ dla każdej strategii naśladowcy $\pi_f \in \Pi_f$. W ten sposób razem z rodziną zmiennych p_{π_l}, można zakodować profil strategii (δ_l, π_f). Wtedy funkcja celu przyjmie postać:

$$\max_{p,q} \sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} u_l(\pi_l, \pi_f) p_{\pi_l} q_{\pi_f}, \tag{3.9}$$

w każdym składniku sumy występuje iloczyn dwóch zmiennych i stałej. Oprócz zmiany funkcji celu, konieczne jest także dodanie ograniczenia pozwalającego na wybór dokładnie jednej

strategii naśladowcy

$$\sum_{\pi_f \in \Pi_f} q_{\pi_f} = 1 \tag{3.10}$$

oraz najbardziej skomplikowany element modyfikacji, czyli zapewnienie, że strategia δ_l definiowana przez zmienne p należy do zbioru X^{π_f} dla strategii naśladowcy, dla której $q_{\pi_f} = 1$. W tym celu dodano do programu dodatkową zmienną typu slack, oznaczmy ją jako $a \in \mathbb{R}$. Następnie rodzinę ograniczeń (3.6) w oryginalnym programie zastąpiono przez następującą rodzinę ograniczeń:

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} u_f(\pi_l, \pi_f) \leqslant (1 - q_{\pi_f})M \qquad \forall \pi_f \in \Pi_f, \tag{3.11}$$

gdzie M to odpowiednio duża stała. Żeby wyjaśnić sens tego ograniczenia, można je rozpisać w dwóch wariantach: gdy $q_{\pi_f} = 1$ i gdy $q_{\pi_f} = 0$. W pierwszym przypadku równanie (3.11) upraszcza się do:

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} u_f(\pi_l, \pi_f) \leqslant 0,$$

czyli efektywnie do równości

$$a = \sum_{\pi_l \in \Pi_l} u_f(\pi_l, \pi_f).$$

W przypadku, gdy wartość zmiennej wynosi 0 równanie upraszcza się do

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} u_f(\pi_l, \pi_f) \leqslant M,$$

prawą nierówność możemy po prostu usunąć, bo znajduje się tam tylko M,

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} p_{\pi_l} u_f(\pi_l, \pi_f),$$

po przeniesieniu sumy na lewą stronę otrzymujemy

$$\sum_{\pi_l \in \Pi_l} u_f(\pi_l, \pi_f) \leqslant a.$$

Ta rodzina ograniczeń oznacza zatem, że dla strategii naśladowcy, która jest wybrana do profilu strategii, zmienna a jest ustawiona na wartość wypłaty naśladowcy, gdy gramy strategię lidera przeciw tej strategii. Dla pozostałych strategii naśladowcy ta wypłata musi być nie większa niż a, co jest dokładnie definicją obszaru najlepszej odpowiedzi. Po uwzględnieniu tych modyfika-

cji program wynikowy jest w następującej formie:

$$\max_{p,q,a} \sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} u_l(\pi_l, \pi_f) p_{\pi_l} q_{\pi_f}$$

z zachowaniem warunków

$$\begin{aligned}
0 \leqslant & a - \sum_{\pi_l \in \Pi_l} p_{\pi_l} u_f(\pi_l, \pi_f) \leqslant (1 - q_{\pi_f}) M && \forall \pi_f \in \Pi_f \\
\sum_{\pi_l \in \Pi_l} p_{\pi_l} = & 1 \\
p_{\pi_l} \geqslant & 0 && \forall \pi_l \in \Pi_l \\
\sum_{\pi_f \in \Pi_f} q_{\pi_f} = & 1 \\
& p_{\pi_l} \in \mathbb{R} \qquad a \in \mathbb{R} \qquad q_{\pi_f} \in \{0,1\}
\end{aligned}$$

Kolejnym etapem transformacji jest usunięcie nieliniowości z funkcji celu. W tym celu zamieniono rodzinę zmiennych p_{π_l} na rodzinę zmiennych p_{π_l,π_f}, która będzie równa iloczynowi prawdopodobieństw wyboru ruchu π_l i π_f w rozważanym profilu strategii. To upraszcza funkcję celu do następującej postaci:

$$\max_{p,q,a} \sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} u_l(\pi_l, \pi_f) p_{\pi_l,\pi_f}, \tag{3.12}$$

następnie dodano ograniczenie wymuszające poprawność nowej rodziny zmiennych p, korzystając z faktu, że rodzina q to zmienne binarne:

$$\sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} p_{\pi_l,\pi_f} = 1 \tag{3.13}$$

$$q_{\pi_f} \leqslant \sum_{\pi_l \in \Pi_l} p_{\pi_l,\pi_f} \leqslant 1 \qquad \forall \pi_f \in \Pi_f \tag{3.14}$$

Nierówność (3.14) oznacza, że dla dokładnie jednego π_f zmienne p_{π_l,π_f} są niezerowe. Wraz z ograniczeniem (3.13) oznacza to, że dla pozostałych π_f sumy wynoszą 0. Na koniec poprawiono ograniczenie (3.11), tak aby wykorzystywało nowe zmienne p. Jako że $p_{\pi_l} = \sum_{\pi_f \in \Pi_f} p_{\pi_l,\pi_f}$, wynikowa postać jest następująca (uwaga na różne strategie π_f i π'_f):

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} \sum_{\pi'_f \in \Pi_f} p_{\pi_l,\pi'_f} u_f(\pi_l, \pi_f) \leqslant (1 - q_{\pi_f}) M \qquad \forall \pi_f \in \Pi_f. \tag{3.15}$$

Poniżej prezentujemy mieszany program liniowy autorów metody DOBSS w ostatecznej

	Multiple-LP	DOBSS
Liczba rozwiązywanych programów	$\|\Pi_f\|$	1
Liczba zmiennych ciągłych	$\|\Pi_l\|$	$\|\Pi_l\| \cdot \|\Pi_f\| + 1$
Liczba zmiennych binarnych	-	$\|\Pi_f\|$
Liczba ograniczeń	$\|\Pi_f\| + 2$	$4\|\Pi_f\| + 2$

Tabela 3.1: Porównanie złożoności metod Multiple-LP i DOBSS

formie:

$$\max_{p,q,a} \sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} u_l(\pi_l, \pi_f) p_{\pi_l,\pi_f}$$

z zachowaniem warunków

$$0 \leqslant a - \sum_{\pi_l \in \Pi_l} \sum_{\pi'_f \in \Pi_f} p_{\pi_l,\pi'_f} u_f(\pi_l, \pi_f) \leqslant (1 - q_{\pi_f}) M \qquad \forall \pi_f \in \Pi_f$$

$$\sum_{\pi_l \in \Pi_l} \sum_{\pi_f \in \Pi_f} p_{\pi_l,\pi_f} = 1$$

$$q_{\pi_f} \leqslant \sum_{\pi_l \in \Pi_l} p_{\pi_l,\pi_f} \leqslant 1 \qquad \forall \pi_f \in \Pi_f$$

$$\sum_{\pi_f \in \Pi_f} q_{\pi_f} = 1$$

$$p_{\pi_l,\pi_f} \in [0,1] \qquad a \in \mathbb{R} \qquad q_{\pi_f} \in \{0,1\}$$

W metodzie DOBSS konieczne jest rozwiązanie tylko jednego mieszanego programu liniowego, ale program ten jest większy, ma $|\Pi_l| \cdot |\Pi_f| + 1$ zmiennych całkowitych, $|\Pi_f|$ zmiennych binarnych oraz $4|\Pi_f| + 2$ ograniczeń. Tabela 3.1 podsumowuje złożoności podejść DOBSS i Multiple-LP. Mimo, że program z podejścia Multiple-LP jest istotnie mniejszy, wyniki raportowane w pracach [69, 68] jednoznacznie wskazują, że dla losowych gier macierzowych średni czas działania DOBSS jest mniejszy. Poprawa ta wynika z faktu, że rozwiązując MILP solvery wykorzystują metodę podziału i ograniczeń lub inne mechanizmy, które ograniczają przestrzeń przeszukiwania wartości zmiennych dyskretnych. W programie DOBSS zmienne dyskretne modelują przestrzeń strategii naśladowcy. Prowadzi to do następującego wniosku: *Uniknięcie pełnego przeglądu wszystkich możliwych strategii naśladowcy istotnie wpływa na czas obliczania Równowagi Stackelberga.* Wniosek ten nie jest zaskoczeniem, jeśli zwrócimy uwagę na fakt, że w sformułowaniu obliczania Równowagi Stackelberga jako dwupoziomowego problemu optymalizacyjnego (3.4) maksimum po strategiach naśladowcy jest zewnętrznym problemem optymalizacyjnym.

3.3 Metoda ASPEN — wykorzystanie specyfiki konkretnego modelu i bardziej wydajna metoda podziału i ograniczeń

Kolejną metodą, gdzie zastosowano interesującą technikę optymalizacji jest ASPEN (Accelerated SPars Engine) [36], która jest przeznaczona do klasy gier o nazwie SPARS (Security Problems with ARbitrary Schedules). Metoda ta zawiera interesujący mechanizm, polegający na iteracyjnym dodawaniu do programu liniowego zmiennych opisujących najbardziej obiecujące strategie lidera. Takie podejście pozwala, w klasie gier SPARS, na uzyskanie znacznie krótszego czasu obliczeń niż w przypadku metody DOBSS.

W dalszej części tego podrozdziału w pierwszej kolejności opisany jest model SPARS, a następnie opisana jest metoda ASPEN ze szczególnym naciskiem na mechanizm dodawania zmiennych do programu liniowego.

Gra SPARS jest inspirowana zagadnieniami bezpieczeństwa, szczególnie problemem alokacji szeryfów powietrznych w Stanach Zjednoczonych (US Federal Air Marshals Service, FAMS) i składa się z trzech zbiorów: zbioru potencjalnych celów ataku (samolotów w przypadku domeny FAMS) $T = \{t_1, \ldots, t_n\}$ oraz zbioru zasobów do ochrony celów (szeryfów) $R = \{r_1, \ldots, r_m\}$. Trzecim zbiorem jest zbiór dopuszczalnych planów alokacji zasobu $S \subseteq T$, który definiuje zbiory celów, które mogą być chronione jednym zasobem. W przypadku FAMS zbiory $s \in S$ definiowane są przez ograniczenia czasu i przestrzeni — żeby chronić dwa samoloty, szeryf musi mieć możliwość przesiąść się z jednego do drugiego, czyli lotnisko startowe drugiego samolotu musi się pokrywać z lotniskiem docelowym pierwszego, a czas przylotu musi być odpowiednio wcześniejszy od czasu odlotu. Uwaga. Na poziomie modelu mówimy tylko o zbiorach celów, bez uwzględniania ich kolejności. Gra rozgrywana jest między obrońcami, którzy pełnią rolę lidera w Grze Stackelberga oraz atakującego, który jest naśladowcą. Strategiami prostymi obrońcy jest zbiór wszystkich przypisań $\Pi_d = \{f : R \to S\}$, czyli wybór celów do ochrony dla każdego z zasobów, dla uproszczenia można rozważać tylko funkcje różnowartościowe — nigdy nie będzie przypisania dwóch zasobów to tego samego planu alokacji. Strategie proste naśladowcy to wskazanie jednego celu spośród T do zaatakowania. Dla każdego celu zdefiniowane są dwie pary wypłat: $u_d^c(t)$ — nagroda dla obrońcy, jeśli atakujący zaatakował cel, który jest chroniony (ang. covered), $u_a^c(t)$ – kara dla atakującego w sytuacji, gdy zaatakował chroniony cel. Analogicznie dla przypadku ataku na niechroniony (ang. uncovered) cel jest nagroda dla atakującego $u_a^u(t)$ i kara dla obrońcy $u_d^u(t)$. Model gry nakłada ograniczenie, aby $u_d^c(t) > u_d^u(t)$ i analogicznie $u_a^c(t) < u_a^u(t)$, które jest zgodne z intuicją za nazwami „kara” i „nagroda”.

Każda strategia podstawowa obrońcy $\pi_d \in \Pi_d$ definiuje pewne pokrycie celów będące sumą teoriomnogościową pokryć celów przez wszystkie jednostki. Na przykład dla $R = r_1, r_2$, strategia $\pi_d(r_1) = \{t_1, t_5\}$, $\pi_d(r_2) = \{t_2, t_4\}$ pokrywa zbiór celów $\{t_1, t_2, t_4, t_5\}$. Zbiór wszystkich pokryć generowanych przez strategie obrońcy oznaczono jako $J = \{J_1, \ldots, J_k\}$ (oznaczenie J pochodzi od angielskiego joint schedule). Niech $C_{i,l}$ będzie 1 w przypadku, gdy cel t_i jest chro-

niony przez J_l, a 0 w przeciwnym przypadku. Podstawą metody ASPEN jest program liniowy prezentowany poniżej:

$$\max_{x,a,k,d} d \tag{3.16}$$

z zachowaniem ograniczeń

$$d - \sum_{J_l \in J} C_{l,i} x_l \cdot u_d^c(t_i) - \left(1 - \sum_{J_l \in J} C_{l,i} x_l\right) u_d^u(t_i) \leqslant (1 - a_i) M \qquad \forall t_i \in T \tag{3.17}$$

$$k - \sum_{J_l \in J} C_{l,i} x_l \cdot u_a^c(t_i) - \left(1 - \sum_{J_l \in J} C_{l,i} x_l\right) u_a^u(t_i) \leqslant (1 - a_i) M \qquad \forall t_i \in T \tag{3.18}$$

$$\sum_{J_l \in J} C_{l,i} x_l \cdot u_a^c(t_i) + \left(1 - \sum_{J_l \in J} C_{l,i} x_l\right) u_a^u(t_i) \leqslant k \qquad \forall t_i \in T \tag{3.19}$$

$$\sum_{J_l \in J} x_l = 1 \tag{3.20}$$

$$x_l \geqslant 0 \qquad a_i \geqslant 0 \tag{3.21}$$

Zmienna d ma wartość równą wypłacie obrońcy (lidera), zmienna k — wypłacie atakującego (naśladowcy). Zmienne x_l definiują rozkład prawdopodobieństwa pokryć celów (strategię mieszaną obrońcy). Zmienna a_i definiują rozkład prawdopodobieństwa ruchów atakującego. Wartość oczekiwana wypłaty w grze, to wypłata gdy cel pokryty razy prawdopodobieństwo jego pokrycia plus wypłata, gdy cel niepokryty razy prawdopodobieństwo zdarzenia przeciwnego. Przy założeniu, że zmienne a są binarne (dyskusja tego faktu dalej), działanie ograniczeń (3.18) i (3.19) jest analogiczne jak ograniczenia (3.15) w DOBSS i wymusza, żeby zmienna k była maksymalną wypłatą atakującego, a aktywna była tylko zmienna a_i odpowiadająca za atak na cel maksymalizujący tę wypłatę. Podobnie ograniczenie (3.17) wymusza, żeby d było wypłatą lidera przy ataku na ten właśnie cel. Ograniczenie (3.20) gwarantuje, że zmienne x_l będą definiować poprawny rozkład prawdopodobieństwa.

W początkowej wersji tego programu zmienne a_i nie są binarne, a co za tym idzie, jego rozwiązanie nie musi być Równowagą Stackelberga (prawe strony ograniczeń (3.17) i (3.18) mają sens, tylko, gdy a_i jest binarne). Wartość d wyliczona przez ten program jest jednak ograniczeniem górnym na wartość wypłaty lidera w stanie Równowagi Stackelberga, ponieważ mamy do czynienia z problemem maksymalizacji, w którym jedyną różnicą jest szersza dziedzina. Istotnym problemem, który stoi na przeszkodzie do praktycznego stosowania tego programu jest fakt, że liczba pokryć celów J może w praktyce być wykładnicza względem liczby celów. W związku z tym autorzy metody ASPEN użyli mechanizmu generowania kolumn (column generation), który polega na iteracyjnym rozwiązywaniu programu najpierw z bardzo ograniczonym podzbiorem zbiorem zmiennych x_l i iteracyjnym dodawaniu kolejnych pokryć do programu, aż do osiągnięcia stanu, gdy dodanie kolejnego x_l już nie poprawi wyniku.

Drugim aspektem decydującym o szybkości rozwiązania jest implementacja metody podziału i ograniczeń, która nie wykorzystuje cech solvera MILP. W pierwszej kolejności omówimy mechanizm generowania kolumn, a następnie pokażemy jak łączy się on z implementacją metody podziału i ograniczeń, która zagwarantuje, że zmienne a_i po ostatniej iteracji metody będą zmiennymi binarnymi.

Generowanie kolumn. Mechanizm generowania kolumn [28] jest popularną techniką stosowaną w programowaniu liniowym do przyspieszenia rozwiązywania dużych programów liniowych, wynikającą z obserwacji, że w wielu dziedzinach duża część zmiennych w rozwiązaniu optymalnym ma wartość 0. Podejście to polega na rozpoczęciu rozwiązywania programu liniowego od jego ograniczonej wersji, gdzie część zmiennych jest usunięta, a następnie dodawanie do programu zmiennych, które pozwolą poprawić wartość funkcji celu. Nazwa generowanie kolumn nawiązuje do reprezentacji programu liniowego jako macierzy, gdzie wiersze to kolejne ograniczenia, a kolumny to kolejne zmienne. Program w takiej ograniczonej wersji możemy traktować równoważnie z programem, gdzie usunięte zmienne zostały ustawione na zero, jednak program z usuniętymi zmiennymi wymaga dużo mniej pamięci i czasu, żeby wyliczyć rozwiązanie. Wyzwaniem przy stosowaniu tej metody jest zbudowanie wydajnego algorytmu stwierdzania czy dodanie danej zmiennej może poprawić rozwiązanie i w jakim stopniu. Algorytm ten w praktyce jest zawsze wyspecjalizowany dla szczególnej postaci programu liniowego.

Celem tego fragmentu rozprawy jest jedynie zademonstrowanie, że mechanizm generowania kolumn nie jest uniwersalny i wersji zbudowanej dla jednej klasy gier nie da się w żaden prosty sposób uogólnić na inne klasy.

W przypadku SPARS mechanizm generowania kolumn będzie zastosowany do zmiennych reprezentujących strategie lidera. Strategie te mają bardzo szczególną postać: są układem n z m dopuszczalnych planów, plany z kolei pokrywają cele. Taka struktura umożliwiła zbudowanie efektywnego mechanizmu wyboru strategii (zmiennych), które można dodać do programu. Początkowo program rozwiązywany jest z niedużym, arbitralnie wybranym zbiorem strategii lidera (może to być po prostu jedna, pierwsza strategia z wektora dostępnych strategii). Po rozwiązaniu takiego programu należy rozważyć, czy któraś ze zmiennych nieobecnych w tej wersji programu, po jej dodaniu i ustawieniu jej wartości na zero nie będzie miała ujemnej wartości zredukowanego kosztu — cechy która mówi, czy warto zwiększyć wartość danej zmiennej (potencjalnie zmieniając też inne zmienne, żeby ograniczenia były dalej spełnione). Podejście to podsumowane jest w Algorytmie 3.2. Algorytmy stosowane do rozwiązywania programów liniowych i definicje z tym związane (w tym zredukowany koszt) nie są przedmiotem tej rozprawy. Informacje o programowaniu liniowym można znaleźć w odpowiednich podręcznikach, na przykład w [13]. Najbardziej uniwersalnym sposobem wyliczania wartości zredukowanego kosztu jest transformacja programu liniowego do programu dualnego (w którym ograniczenia stają się zmiennymi, a zmienne ograniczeniami, a rozwiązanie optymalne jednego z programów, pozwala wskazać rozwiązanie drugiego), a następnie próba dodania każdej z potencjal-

Algorytm 3.2: Generowanie kolumn w metodzie ASPEN.

input: Dopuszczalne wartości a_i

$P \leftarrow$ program (3.17)–(3.20) z małym podzbiorem zmiennych x_l i ustawieniami zmiennych a_i z wejścia;

repeat

  rozwiąż program P;

  $D \leftarrow$ program dualny do P;

  wylicz zmienną x_l, która może poprawić wartość funkcji celu w P;

  dodaj x_l do P;

until *nie istnieje kandydat na x_l, które poprawi rozwiązanie*;

return *rozwiązanie programu P*;

nych zmiennych (w programie dualnym ograniczenia) i wyliczenie wartości zredukowanego kosztu podręcznikową techniką. W programie dualnym do (3.17)– (3.20) mamy rodziny zmiennych odpowiadające poszczególnym ograniczeniom, oznaczmy jako $\{w_i\}$ rodzinę zmiennych odpowiadających ograniczeniom (3.17), jako $\{y_i\}$ — ograniczeniom (3.18), jako $\{z_i\}$ — ograniczeniom (3.19), a jako q zmienną odpowiadającą ograniczeniu (3.20). Aktualne wartości tych zmiennych możemy wyznaczyć na podstawie wartości zmiennych w aktualnym rozwiązaniu programu prymalnego. Na podstawie ograniczenia odpowiadającego zmiennej x_l w programie dualnym możemy wyliczyć zredukowany koszt, który wynosi:

$$v_l = q + \sum_{t_i \in T} C_{i,l} \left(w_i \left(u_d^c(t_i) - u_d^u(t_i) \right) + y_i \left(u_a^c(t_i) - u_a^u(t_i) \right) - z_i \left(u_a^c(t_i) - u_a^u(t_i) \right) \right) \quad (3.22)$$

Takie rozwiązanie jest jednak niewydajne. Wielokrotne wyznaczenie zredukowanego kosztu w ten sposób spowodowałoby w efekcie wydłużenie czasu obliczeń w stosunku do pierwotnego programu ze wszystkimi zmiennymi.

Autorzy metody ASPEN zaproponowali budowę pewnego grafu i znalezienie w nim maksymalnego przepływu o minimalnym koszcie, a następnie udowodnili, że strategia opisana przez ten przepływ będzie miała największy możliwy zredukowany koszt w rozwiązywanym programie liniowym. Przytoczymy teraz samą konstrukcję grafu, bez dowodu, tylko aby pokazać, jak bardzo specyficzna dla dziedziny SPARS jest ta metoda.

Zbudowano graf skierowany złożony ze ścieżek pomiędzy wierzchołkiem źródłowym f i ujściem g. Dla każdego dopuszczalnego planu alokacji zasobu $s_h = \{t_{k_1}, \ldots t_{k_r}\}$ dodano do grafu ścieżkę, złożoną z $2|s_h|$ wierzchołków — każdemu z celów będzie odpowiadać para wierzchołków $a_{s_h,t_{k_i}}, b_{s_h,t_{k_i}}$, a dodana ścieżka jest następująca: f,$a_{s_h,t_{k_1}}$,$b_{s_h,t_{k_1}}$,$a_{s_h,t_{k_2}}$,$b_{s_h,t_{k_2}}, \ldots a_{s_h,t_{k_r}}$, $a_{s_h,t_{k_r}} g$. Wszystkie krawędzie w tych ścieżkach mają przepustowość 1, wartości kosztów zdefiniujemy za chwilę. Dodatkowo w grafie jest bezpośrednia krawędź między źródłem a ujściem o nieskończonej przepustowości — posłuży ona do kierowania tam przepływu, gdy nie ma już planów alokacji, które mogłyby poprawić wynik. Koszty w grafie są tak skonstruowane, aby sumaryczny koszt maksymalnego przepływu niezawierającego krawędzi fg odpowiadał pew-

nej wartości v_l. Dla każdego s_h, dla każdego t_k, dla krawędzi a_{s_h,t_i}, a_{s_h,t_i} koszt jednostkowy przepływu wynosi $w_i(u_d^c(t_i) - u_d^u(t_i)) + y_i(u_a^c(t_i) - u_a^u(t_i)) - z_i(u_a^c(t_i) - u_a^u(t_i))$. Koszty pozostałych krawędzi wynoszą 0. Z użyciem algorytmu poszukiwania przepływu o minimalnym koszcie znaleziono w tak zbudowanym grafie przepływ między f i g o wielkości n (liczba zasobów obrońcy). Jako że wartości $C_{i,l}$ wiążą ze sobą pokrycia poszczególnych celów z pokryciami przez strategie obrońcy, łatwo można zauważyć, że koszt przepływu będzie wartością zredukowanego kosztu odpowiadającego mu pokrycia J_l powiększoną o h. Poprzez dodanie sztucznego źródła i połączenia go z bieżącym źródłem krawędzią o koszcie $-h/n$ można wyrównać ten błąd. Jeśli w przepływie została użyta krawędź łącząca źródło z ujściem, to nie ma już kandydatów na poprawienie wyniku.

Tak skonstruowany graf silnie zależy od istnienia w grze struktury celów i pokryć, stąd nie da się tego podejścia uogólnić na inne gry.

Metoda podziału i ograniczeń. W związku z zastosowaniem mechanizmu generowania kolumn, wykorzystanie zmiennych binarnych, tak jak w DOBSS byłoby niewydajne — rozwiązywanie programu mieszanego jest dużo bardziej kosztowne, a po każdym kroku dodania kolumny rozwiązywany jest kolejny program. W związku z tym autorzy metody ASPEN zastosowali własną metodę podziału i ograniczeń.

W programie DOBSS zmienne binarne miały bardzo szczególny układ. Na mocy ograniczenia (3.10) dokładnie jedna zmienna q_i jest jedynką, a pozostałe są zerami, co odpowiada faktowi, że poszukujemy jednej strategii prostej, która jest optymalną odpowiedzią naśladowcy. Analogicznie jest w programie wykorzystywanym przez ASPEN, gdzie mamy ograniczenie (3.20). W związku z takimi własnościami, podział przestrzeni w metodzie ASPEN działa w następujący sposób: weź pierwszą strategię naśladowcy t_i i oblicz wynik algorytmu generowania kolumn dla $a_i = 1$ i pozostałych $a_j = 0$. Jest to pewien wynik, będący rozwiązaniem dopuszczalnym w sensie problemu optymalizacyjnego, a więc ograniczenie dolne na wartość wypłaty lidera. Następnie uruchom algorytm generowania kolumn dla $a_i = 0$ i pozostałych a_j będących ciągłymi zmiennymi programu. Z punktu widzenia programu liniowego mamy przestrzeń przeszukiwania, która jest nadzbiorem przestrzeni zmiennych w programie, gdzie zmienne a są binarne. W ten sposób otrzymuje się oszacowanie górne wyniku, gdy $a_i = 0$. Jeśli oszacowanie górne jest mniejsze lub równe dolnemu, zwracamy wynik dla $a_i = 1$. W przeciwnym przypadku powtarzamy proces dla kolejnych strategii naśladowcy, z tą różnicą, że wcześniej sprawdzone strategie naśladowcy mają zmienne a_j ustawione na 0. Postępowanie to podsumowane jest w Algorytmie 3.3. Można zobaczyć, że jeśli pierwsza część warunku pętli w linii 6 nie będzie spełniona, to program nie sprawdzi wszystkich możliwych strategii naśladowcy, a co za tym idzie będzie działał szybciej. W tym miejscu warto zauważyć, że strategie atakującego są przeglądane jedna po drugiej i kolejność ich przeglądania może mieć wpływ na wcześniejsze lub późniejsze zakończenie pętli. Autorzy metody ASPEN zaproponowali także heurystykę wyznaczającą kolejność przeglądu strategii atakującego. Heurystyka, podobnie jak cała metoda

Algorytm 3.3: Metoda podziału i ograniczeń w ASPEN.

input: wektor strategii naśladowcy $(t_i)_{\forall t_i \in T}$

$\pi_d^* \leftarrow$nil // Najlepsza znana dotąd strategia obrońcy (lidera).
$LB \leftarrow -\infty$// Ograniczenie dolne na wypłatę obrońcy (wypłata dla strategii π_d^*)
$UB \leftarrow +\infty$// Ograniczenie górne na wypłatę obrońcy
$ZA \leftarrow \varnothing$// Zbiór zmiennych a_i, które będą ustawione na zero w rozwiązywanym programie liniowym
$Q \leftarrow (t_i)_{\forall t_i \in T}$// Kolejka strategii atakującego do rozważenia
while $LB < UB \wedge Q \neq \varnothing$ **do**
    $t_k \leftarrow$ pobierz pierwszą wartość z kolejki Q;
    $(U,\pi_d) \leftarrow$ uruchom Algorytm 3.2 z $a_k = 1$ i pozostałymi $a_i = 0$;
    if $U > LB$ **then** $(LB,\pi_d^*) \leftarrow (U, \pi_d)$;
    $ZA \leftarrow ZA \cup \{a_k\}$;
    $UB \leftarrow$ uruchom Algorytm 3.2 ze zmiennymi ze zbioru ZA ustawionymi na 0, a pozostałym zmiennym pozwól na dowolne wartości ze zbioru $[0,1]$.
end
return (LB,π_d^*)

jest ściśle związana ze strukturą gry SARS i polega na znalezieniu optymistycznego pokrycia (potencjalnie większego niż wynika z ograniczeń gry) celów przez obrońcę, które zminimalizuje maksymalną wypłatę atakującego. Dla tego pokrycia są wyliczane wartości oczekiwane wypłaty obrońcy przeciwko każdej ze strategii atakującego. Na koniec strategie atakującego są posortowane malejąco według obliczonych wartości, co oznacza, że najpierw sprawdzane są najbardziej obiecujące cele. Pokrycie jest wyliczane przy użyciu poniższego programu liniowego, który był stosowany do aproksymacji wyniku we wcześniejszym rozwiązaniu dedykowanemu tej klasie gier — ERASER-C [48]. W poniższym programie L oznacza maksymalny rozmiar (liczbę celów) dopuszczalnego planu dla jednostki obrońcy, a $\hat{C}_{t_i,s}$ mówi, czy t_i należy do planu s, a $T(s)$ jest arbitralnie wybranym elementem zbioru s:

$$\min_{a,c,k} k \tag{3.23}$$

z zachowaniem ograniczeń

$$0 \geqslant k - c_i u_a^c(t_i) - (1 - c_i) u_a^u(t_i) \geqslant (1 - a_i)M \qquad \forall t_i \in T \tag{3.24}$$

$$\sum_{s \in S} \hat{C}_{T(s),s} \leqslant n \tag{3.25}$$

$$c_i \leqslant \sum_{s \in S} \hat{C}_{t_i,s} \qquad \forall t_i \in T \tag{3.26}$$

$$\sum_{t_i \in T} c_i \leqslant n \cdot L \tag{3.27}$$

$$c_i \leqslant q_i \qquad \forall t_i \in T \tag{3.28}$$

$$q_i \in \{0,1\}, \qquad c_i \in [0,1] \tag{3.29}$$

Zmienne c_i opisują prawdopodobieństwo pokrycia celu przez obrońcę. Ograniczenie (3.24) jest analogiczne do (3.18) i gwarantuje, że zmienna k będzie miała wartość będącą optymalną wypłatą atakującego przy zadanym pokryciu celów. Ograniczenia (3.25)–(3.27) definiują pokrycie, które częściowo uwzględnia zasady gry. Potencjalnie może być większe niż możliwe do zrealizowania, ale na pewno nie mniejsze. Należy też zwrócić uwagę na brak ograniczenia na sumę zmiennych q — rozważane są wszystkie warianty ataku z maksymalną wypłatą. Ostatnie ograniczenie — (3.28) wymusza, żeby pokryte były tylko cele dla których atakujący ma najwyższą wypłatę, bo tylko wtedy ograniczenie (3.24) będzie lepiej przybliżało wartość wypłaty atakującego jako k (chociaż w przypadku, gdy więcej niż jedno q jest niezerowe, ta wartość będzie przeszacowana).

Wyniki w przytoczonej pracy pokazują, że dla klasy gier SPARS metoda ASPEN jest nieporównywalnie szybsza od DOBSS i skaluje się wielomianowo wraz ze wzrostem liczby celów, podczas gdy DOBSS skaluje się wykładniczo.

Uwaga. W oryginalnej pracy autorzy proponują nieco ogólniejszą wersję metody, gdzie obrońca ma do dyspozycji kilka klas jednostek i każda klasa ma inny zbiór dopuszczalnych planów S. Z punktu widzenia koncepcji rozwiązania, nie wnosi ona jednak nic ponadto co zostało zaprezentowane powyżej, a komplikuje prezentowane programy linowe. W związku z tym prezentujemy tylko uproszczony wariant.

Obserwacja 3.1. *Konstrukcja metody ASPEN pokazuje, że można osiągnąć znaczące przyspieszenie obliczeń, jeśli uniknie się przeglądania wszystkich możliwych strategii lidera (mechanizm generowania kolumn) i naśladowcy (metoda podziału i ograniczeń). Jest to szczególnie istotne kiedy przestrzeń strategii jest wykładnicza względem parametrów gry. Niestety metody te wymagają bardzo szczegółowej analizy struktury gry i wykorzystania jej cech. Nie jest możliwe uogólnienie takich podejść na wszystkie klasy gier.*

3.4 Metody wykorzystujące strategie brzegowe

Program liniowy (3.24) – (3.28) jest przykładem wykorzystania strategii brzegowych — w programie liniowym mamy zmienne definiujące oczekiwane pokrycie każdego z celów a nie wejściową strategię mieszaną lidera. W grach, gdzie definiujemy pokrycia obiektów, zawsze możemy zdefiniować strategie brzegowe jako wektory pokrycia celów, a każdej strategii mieszanej możemy przyporządkować pewną strategię brzegową. To przyporządkowanie nie musi być funkcja różnowartościową ani „na”. Na przykład dla trzech celów — t_1, t_2, t_3, t_4 i strategii prostych $s_1 = \{t_1, t_3\}$, $s_2 = \{t_2,t_3\}$, $s_3 = \{t_1, t_4\}, s_4 = \{t_2, t_4\}$, dysponując jedną jednostką obrony:

- strategię brzegową, w której każdy z celów pokryty jest z prawdopodobieństwem $0{,}5$ możemy uzyskać na dwa sposoby: $P(s_1) = 0{,}5$ i $P(s_4) = 0{,}5$ oraz $P(s_2) = 0{,}5$ i $P(s_3) = 0{,}5$,

- w żaden sposób nie jesteśmy w stanie uzyskać strategii pokrywającej zarówno t_3 i t_4 z prawdopodobieństwem $0{,}6$.

To z jaką dokładnością można przybliżać strategie mieszane strategiami brzegowymi zależy od dodatkowych ograniczeń nałożonych na strukturę zbiorów s_i. W ogólnym przypadku (omówionym w poprzednim podrozdziale) nie da się zaproponować dobrego przybliżenia strategii taką metodą. Jednak w momencie, gdy nałożymy dodatkowe ograniczenia na strukturę s_i, może się okazać, że metoda strategii brzegowych daje wyniki bliskie dokładnym. Tak jest na przykład w przypadku modelu, który przydziela różnym klasom pasażerów różne wyposażenie bramek kontrolnych na lotnisku [73].

Innym, ciekawym, z punktu widzenia tej rozprawy, podejściem jest zastosowanie metody strategii brzegowych do następującego modelu gry wielokrokowej:

Gra SMOS (Stackelberg Model of the Oil-Siphoning problem) jest kolejną grą z nurtu Gier Obronnych i została zaproponowana w pracy [90]. Modeluje sytuację, gdzie jednostki obrony patrolują morze po którym pływają tankowce z ropą. Po morzu poruszają się też piraci, którzy mogą tankowiec porwać i przepłynąć nim do swojej bazy, gdzie spuszczą z niego ropę. Formalnie gra jest zdefiniowana poprzez:

- Prostokątną przestrzeń punktów $[0{,}w] \times [0{,}h]$, podzieloną na równomierną siatką na kwadratowe strefy. Oznaczmy zbiór tych stref jako Z.
- Dyskretny czas złożony z τ równoodległych punktów czasowych $\tau = t_1, \ldots, t_n$.
- m łodzi patrolowych obrońcy, które poruszają się z prędkością pozwalającą przepłynąć nie dalej niż do sąsiedniej strefy w jednym kroku czasowym.
- Jednej łodzi piratów, z takimi samymi ograniczeniami ruchu jak obrońca.
- Zbiorów stref z których jednostka obrońcy może stratować $S \subseteq Z$ i $T \subseteq Z$ — w praktyce będą to porty lub przystanie, gdzie cumują łodzie obrońcy.
- Zbioru stref $O \subseteq Z$ w którym piraci mają swoje bazy i mogą spuścić ropę z tankowca.
- Funkcji wypłat piratów $u : \tau \times Z \rightarrow \mathbb{R}^+ \cup \{0\}$ definiującą wypłatę w każdej ze stref w każdym kroku czasowym.

Strategia prosta piratów polega na zaplanowaniu trasy statku, którym piraci przeprowadzają atak. Trasa polega na wyborze punktu czasowego t_h i strefy $z_1 \in Z$, w której zostanie zaatakowany statek, a następnie doprowadzenie go do wybranej strefy $o \in O$ po dowolnej trajektorii spełniającej ograniczenia na ruch, pod warunkiem osiągnięcia tego punktu najpóźniej w czasie t_τ. Oznaczmy tę trasę jako $F = (t_h, z_1), (t_{h+1}, z_2), \ldots (t_{h+l} m z_l)$. Poza zakresem czasu $[t_h, t_{h+1}]$ piraci są niewidoczni dla obrońców.

Strategia obrońcy polega na zaplanowaniu pozycji każdej z jednostek obrony tak, żeby w pierwszym kroku czasowym pozycja była w dowolnej ze stref $s \in S$, w ostatnim w dowolnej ze stref $t \in T$, a kroki pośrednie są zgodne z ograniczeniami na ruch.

Gra ma następujący przebieg:

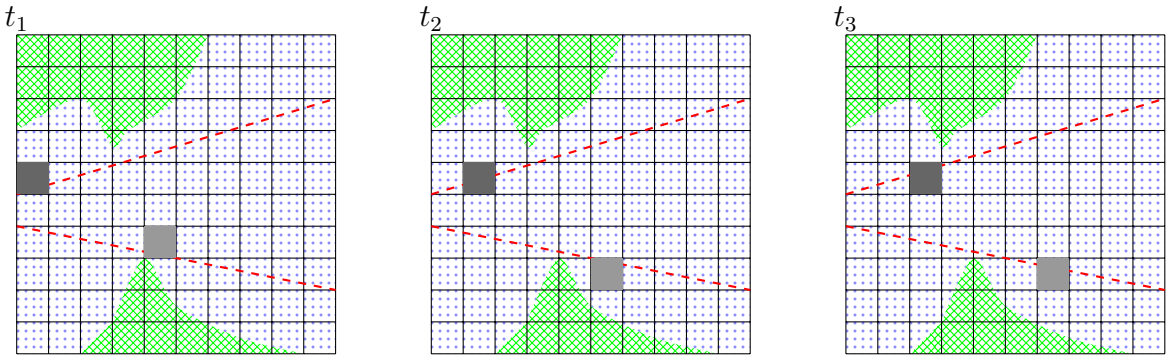
1. Jeśli istnieje para krok czasowy, pozycja na ścieżce piratów, która pokrywa się z pozycją jednostki obrony, to z prawdopodobieństwem $d \leqslant 1$ piraci zostają złapani. Jeśli kilka jednostek obrony jest w tym samym punkcie, to prawdopodobieństwo złapania jest odpowiednio zwielokrotnione (ale nie więcej niż do jedynki w sumie).

2. Podobnie, jeśli istnieje para krok czasowy, pozycja na ścieżce piratów, która jest w odległości 1 od odpowiedniej pozycji jednostki obrony, to piraci zostają wykryci i złapani z prawdopodobieństwem $e < d$. Podobnie — wiele jednostek obrony powoduje zwielokrotnienie prawdopodobieństwa. Co więcej te prawdopodobieństwa sumują się z prawdopodobieństwami z poprzedniego punktu w każdym kroku czasowym.

3. Ponadto w momencie rozpoczęcia ataku t_h załoga przejmowanego tankowca próbuje wysłać sygnał alarmowy, z prawdopodobieństwem sukcesu α. Jeśli sygnał został wysłany i jest jednostka obrony, która jest w stanie dogonić piratów (znaleźć się w odpowiednim czasie na dowolnym punkcie ścieżki piratów do wierzchołka końcowego, poruszając się zgodnie z ograniczeniami ruchu), to piraci zostają złapani.

Prawdopodobieństwa mniejsze od jedynki modelują fakt, że powierzchnia jednej strefy w proponowanym zastosowaniu może wynosić kilkadziesiąt kilometrów kwadratowych i obrońca nie jest w stanie zapewnić doskonałej obserwacji całego jej obszaru. W przypadku złapania piratów obie strony otrzymują wypłatę 0. W przypadku, gdy piraci nie zostaną złapani otrzymują wypłatę $u(t_h, z_1)$, gdzie t_h i z_1 to odpowiednio czas i strefa, w której został przejęty tankowiec. Obrońca otrzymuje wtedy wypłatę przeciwną: $-u(t_h, z_1)$. Gra jest grą o sumie zerowej, co zostało wykorzystane w konstrukcji metody jej rozwiązywania. Gra nie definiuje wprost pozycji tankowców na morzu, zamiast tego użyte są zróżnicowane wartości wypłat — w punktach gdzie w danej chwili nie ma żadnego tankowca, wypłaty są zerami. W przypadku, gdy w danej strefie w danym momencie przebywa tankowiec, wypłata jest zdefiniowana według wartości ładunku tankowca. Przykładowa siatka stref z układem wypłat jest prezentowana na Rysunku. 3.1.

Metoda znajdowania Równowagi Stackelberga w grze SMOS zaproponowana w przytoczonej pracy opiera się o strategie brzegowe — zmienne definiujące pokrycie każdej ze stref w każdym kroku czasowym. Oprócz strategii brzegowych metoda wykorzystuje mechanizm generowania ograniczeń, który pozwala na włączenie do programu tylko wybranych strategii naśladowcy (podobnie jak mechanizm generowania kolumn w przypadku strategii lidera) oraz metodę abstrahowania gry — budowy uproszczonej gry na bazie oryginalnej w celu dalszej poprawy wydajności. Poniżej zaprezentujemy najpierw program liniowy i wskażemy jakie uproszczenia zostały w nim dokonane, następnie omówimy zasadzę działania mechanizmu generowania ograniczeń, a na koniec technikę abstrahowania gry.



Rysunek 3.1: Przykładowa siatka definiująca grę SMOS. Kolory poszczególnych stref odpowiadają wartości funkcji $u(t,z)$ w trzech kolejnych krokach czasowych. Linią przerywaną zaznaczono trasy tankowców, większego u góry i mniejszego u dołu. Trasy te są przybliżone w modelu z użyciem wartości $u(t,z)$. Stopień zacieniowania komórki odpowiada wartości funkcji u. W miejscu, gdzie komórka jest niewypełniona, wartość wynosi 0.

W programie liniowym rozwiązującym SMOS wystąpi rodzina zmiennych $c_{t,z} \geqslant 0$ definiująca brzegowe prawdopodobieństwo że obrońca będzie w strefie z w czasie t. Technicznie będzie to prosta suma prawdopodobieństw znalezienia się tu każdej z jednostek, w związku z czym wartość ta może być większa od jedynki. Symbolem F będziemy oznaczać strategię piratów, a symbolem $\mathcal{F}$ zbiór wszystkich strategii prostych piratów. Następnie, przez $N(z)$ oznaczymy zbiór wszystkich stref sąsiadujących z z w pionie lub poziomie. Dodatkowo stosując nadużycie notacji, $N(F)$, gdzie F to strategia piratów, będzie zawierać wszystkie pary (t,z), gdzie w czasie t strefa z sąsiaduje z pozycją piratów w danej strategii, formalnie $N(F) = \{(t,z)|\exists z' z \in N(z') \wedge (t,z') \in F\}$ oraz $\mathcal{R}(F)$ jako zbiór par (t_h, z), gdzie t_h to czas rozpoczęcia ataku w strategii F, a z to dowolna strefa z której jednostka obrony zdąży przeciąć trajektorię atakującego i go zatrzymać po rozpoczęciu ataku. Te funkcje pomocnicze pozwolą nam zdefiniować prawdopodobieństwo niepowodzenia ataku

$$dpp(\mathbf{c}, F) = \min \left\{ 1, d \sum_{(t,z)\in F} c_{t,z} + e \sum_{(t,z)\in N(F)} c_{t,z} + \alpha \sum_{(t,z)\in \mathcal{R}(F)} c_{t,z} \right\}, \tag{3.30}$$

gdzie $\mathbf{c}$ oznacza wektor zmiennych $c_{t,z}$. Wzór (3.30) stosuje uproszczenie — zawiera sumę prostą prawdopodobieństw złapania piratów w kolejnych krokach czasowych. W szczególności oznacza to, że jeśli zarówno strategia atakującego jaki i pewnej jednostki obrońcy zawierają dwa kolejne wspólne punkty (t_x, z_x) i (t_{x+1}, z_{x+1}), to prawdopodobieństwo złapania piratów w tych punktach zostaną zsumowane, mimo że w modelu piraci byliby złapani w drugim punkcie tylko pod warunkiem, że nie byli złapani w poprzednim kroku czasowym. Jeśli współczynniki d, e i α są duże lub gdy obrońca dysponuje dużą liczbą jednostek, to uproszczenie może powodować drastyczne przeszacowanie stopnia ochrony stref, gdy przytoczone wartości są małe, przeszacowanie wartości będzie również małe. Takie uproszczenie jest jednak konieczne do budowy programu liniowego, ponieważ tak zdefiniowana funkcja dpp jest liniowa (z dokładnością do

minimum) względem zmiennych $c_{t,z}$. Na podstawie powyższego możemy zdefiniować wartość oczekiwaną wypłaty piratów: $U_a(\mathbf{c}, F) = (1 - dpp(\mathbf{c}, F))u(t_h, z_1)$, gdzie t_h i z_1 są odpowiedni czasem i miejscem rozpoczęcia ataku w F. W poniższym programie będzie użyta notacja $u(F)$ do określenia tej wypłaty.

Korzystając z definicji pomocniczych możemy teraz w sposób zwięzły przytoczyć program liniowy rozwiązujący SMOS:

$$\max_{c,U,\phi} \quad U \tag{3.31}$$

z zachowaniem ograniczeń

$$U \leqslant -(1 - p_F)u(F) \qquad \forall F \in \mathcal{F} \tag{3.32}$$

$$p_F \leqslant 1 \qquad \forall F \in \mathcal{F} \tag{3.33}$$

$$p_F \leqslant d \sum_{(t,z)\in F} c_{t,z} + e \sum_{(t,z)\in N(F)} c_{t,z} + \alpha \sum_{(t,z)\in \mathcal{R}(F)} c_{t,z} \tag{3.34}$$

$$c_{t_i,z_j} = \sum_{z_k \in N(z_j)\cup\{z_j\}} \phi_{(z_j,t_i),(z_k,t_{i+1})} \qquad \forall z_j \in Z \quad \forall k \in \{1, \dots n-1\} \tag{3.35}$$

$$c_{t_i,z_j} = \sum_{z_k \in N(z_j)\cup\{z_j\}} \phi_{(z_k,t_{i-1}),(z_j,t_i)} \qquad \forall z_j \in Z \quad \forall k \in \{2, \dots n\} \tag{3.36}$$

$$m = \sum_{z_k \in S} c_{z_k,t_1} \tag{3.37}$$

$$m = \sum_{z_k \in T} c_{z_k,t_n}, \tag{3.38}$$

$$\tag{3.39}$$

gdzie zmienna U ma wartość optymalnej wypłacie obrońcy (lidera). Zmienne p_F przechowują wartość $dpp(\mathbf{c},F)$. Ograniczenia (3.33) i (3.34) służą wyliczeniu minimum z definicji dpp. Jest to typowa sztuczka w programach liniowych, możliwa do zastosowania, gdy wartość minimum wchodzi z dodatnim współczynnikiem do funkcji celu. Ograniczenie (3.32) minimalizuje wypłatę obrońcy po wszystkich strategiach piratów. Z racji tego, że gra ma zerową sumę to ograniczenie wystarczy do wymuszenia optymalnej odpowiedzi piratów, którzy są naśladowcą w tej grze. Ograniczenia (3.35) i (3.36) definiują „przepływ" ochrony, zmienne ϕ mówią ile prawdopodobieństwa obrony danej strefy przepływa do sąsiedniej strefy w kolejnym kroku czasowym. Ograniczenia (3.37) i (3.38) gwarantują start jednostek obrony ze stref z S i zakończenie ich patrolu w strefach z T.

Podsumowując, proponowany program ma dwa uproszczenia, które należy jeszcze raz podkreślić:

- gra o sumie zerowej pozwala uniknąć stosowania zmiennych binarnych,
- stosowanie sumy prostej w funkcji dpp prowadzi do przeszacowania wyniku obrońcy, ale pozwala zastosować solvery programowani liniowego.

3.4.1 Generowanie ograniczeń

Tak sformułowany program nie jest jeszcze możliwy do zastosowania w praktyce, ponieważ liczba ograniczeń (3.32) – (3.34) jest proporcjonalna do liczby strategii piratów. Liczba tych strategii rośnie z kolei wykładniczo wraz z liczbą kroków czasowych. Warto jednak zauważyć, że ogromna liczba tych strategii to ataki na zupełnie nieistotne strefy, które nie zawierają żadnych statków lub ataki z niepotrzebnie długo trajektorią ucieczki. W związku z tym rozwiązanie w prezentowanej pracy używa dodatkowo mechanizmu generowania ograniczeń, które odpowiadają za poszczególne strategie naśladowcy.

Mechanizm generowania ograniczeń jest pomysłem podobnym do generowania kolumn, jednak prostszym w implementacji. Podejście to polega na rozwiązaniu programu z niewielkim podzbiorem ograniczeń, a następnie szukamy pośród pozostałych ograniczeń takiego, które nie jest spełnione i dodajemy je do programu. Jeśli wszystkie ograniczenia są spełnione, to rozwiązanie zredukowanego programu jest też rozwiązaniem programu z wszystkimi ograniczeniami. Podobnie ja w generowaniu kolumn, tu również w najprostszym wariancie można przejrzeć wszystkie ograniczenia sprawdzając, czy któreś z nich nie jest naruszone, jednak nie jest to rozwiązanie efektywne.

Generowanie ograniczeń w przytoczonym rozwiązaniu jest wykorzystane do trójek ograniczeń (3.32) – (3.34) odpowiadających strategiom piratów. Zwróćmy uwagę, że w takiej trójce najważniejsze jest ograniczenie (3.34), bo pozostałe dwa realizują funkcję minimum. Co więcej, ze względu na maksymalizację zmiennej U w programie i konstrukcję ograniczenia (3.32), najlepszym kandydatem na naruszone ograniczenie (3.34) będzie takie, w którym suma po prawej stronie jest najmniejsza. Intuicyjna interpretacja tej zależności mówi, że rozpoczynamy rozwiązywanie od ograniczonej gry, gdzie piraci mają do dyspozycji pewien podzbiór strategii, a następnie poszukujemy takiej strategii piratów do dodania, żeby mogli uzyskać możliwie dużą wypłatę przy znalezionej strategii obrońcy. Przypomnijmy tu, że gra jest o sumie zerowej, więc podniesienie wypłaty piratów automatycznie obniży wypłatę obrońcy.

W celu efektywnego poszukiwania takich strategii możemy zbudować graf, który dla ustalonej strategii obrońcy będzie następującej postaci: wierzchołkami grafu będą pary (t_i, z). Krawędzie w grafie będą pomiędzy takimi parami $((t_i, z_1), (t_{i+1}, z_2))$, w których $z_1 = z_2$ lub z_1 jest sąsiadem z_2. Innymi słowy graf reprezentuje trójwymiarową przestrzeni gry (2 wymiary siatki stref i 1 wymiar czasu), a krawędzie są między tymi wierzchołkami, miedzy którymi zgodnie z zasadami gry łódź może się przemieścić. Dodatkowo dodajemy do tego grafu wierzchołek X i krawędzie z każdego wierzchołka, gdzie z_j jest ze zbioru S stref, gdzie piraci spuszczają ropę z tankowców. Wagi wszystkich krawędzi prowadzących do wierzchołka (t_i, z_j) są takie same, równie prawdopodobieństwu wykrycia piratów, gdy znajdą się w tym wierzchołku, będącego sumą dc_{t_i,z_j} i ec_{t_i,z_k} dla wszystkich sąsiadów z_k wierzchołka z_j. Przy tak dobranych wagach dowolna ścieżka realizująca strategię atakującego będzie miała wagę odpowiadającą sumie prawdopodobieństw z ograniczenia (3.34) z pominięciem składnika mnożonego przez

α. Krawędzie do wierzchołka X mają wagę 0. Generowanie ograniczeń w przytoczonej metodzie opiera się o znalezienie dla każdego możliwego startu ataku, czyli wierzchołka (t_i, z_j) najkrótszej ścieżki do wierzchołka X. Do znalezienia tej ścieżki stosowana jest jednak modyfikacja algorytmu Dijkstry, która dodatkowo uwzględnia przecięcie trasy z punktami osiągalnymi przez obrońcę w przypadku wystąpienia alarmu (innymi dla każdej sprawdzanej pary (t_i, z_j)). Szczegóły tej implementacji nie są istotne z punktu widzenia niniejszej rozprawy, więc ich nie przytaczamy.

Podsumowując, podobnie jak generowanie kolumn, efektywne generowanie ograniczeń jest silnie związane z konkretną grą. W tym przypadku są eksploatowane dwie cechy: struktura dająca się opisać grafem oraz suma zerowa. Uogólnienie takich metod, zachowując ich efektywność nie jest możliwe.

3.4.2 Abstrahowanie gry

Ostatnią techniką zastosowaną w metodzie rozwiązywania SMOS jest abstrahowanie mniejszej gry z gry oryginalnej w celu dalszego zwiększenia szybkości działania. Zastosowana technika abstrakcji polega na prostej agregacji czasu i przestrzeni. Parametrem metody jest współczynnik agregacji $s \in \mathbb{N}$. Metoda polega na zastąpieniu oryginalnej siatki siatką mającą s razy mniej wierszy i kolumn — każda strefa w nowej grze odpowiada s^2 strefom w grze oryginalnej i rozważeniu tylko co s-tego punktu czasowego, a wypłaty są odpowiednio uśrednione. Optymalna strategia znajdowana jest w tak uproszczonej grze. Następnie stosowana jest procedura odzyskiwania strategii w grze oryginalnej, która polega zastąpieniu każdej strategii prostej obrońcy poprzez rozkład jednostajny nad s^2 strategiami prostymi, w których każda jednostka obrońcy porusza się odpowiadających strefach wchodzących w skład agregowanej strefy.

Abstrahowanie jest kolejnym mechanizmem silnie związanym ze strukturą danej gry. W tym przypadku wykorzystuje fakt, ze gra zdefiniowana jest na siatce, którą można agregować. Każdorazowe stosowanie tego typu podejścia wymaga analizy struktury danej gry.

3.5 Uogólnienie SMOS do sumy niezerowej

We wszystkich, za wyjątkiem DOBSS, opisanych powyżej metodach przyspieszenia obliczania Równowagi Stackelberga istotne były specyficzne cechy modelu i uogólnienie na szerszą klasę gier nie było możliwe bez znaczącego nakładu pracy. W tym podrozdziale pokażemy jak rezygnacja z sumy zerowej w grze SMOS, nawet przy uproszczeniu innych jej elementów, komplikuje program liniowy poszukujący Równowagi Stackelberga. Treść zawarta w tym podrozdziale jest jedynym elementem tego rozdziału, której współautorem jest autor rozprawy. Prezentowane tu podejście jest fragmentem pracy opublikowanej w styczniu 2019 na konferencji AAAI [47]. Autor rozprawy wraz z Adamem Żychowskim współpracował nad uogólnieniem samego modelu gry do sumy niezerowej i jest autorem modyfikacji programu liniowego rozwiązującego

zmodyfikowany SMOS o niezerowej sumie. Autorstwo obejmuje koncepcje modyfikacji i zapis programu liniowego w formie matematycznej. Autorem programu komputerowego implementującego to rozwiązanie jest Filip Grajek, kolejny współautor pracy [47].

Zmodyfikowana gra SMOS. Zasadniczą różnicą modyfikacji w stosunku do oryginalnego sformułowania jest dopuszczenie sumy niezerowej — w przypadku sukcesu piratów obrońca otrzymuje pewną ujemną karę, a piraci dodatnia nagrodę. Wartości te nie muszą sumować się do 0. Poza tym gra została nieco uproszczona. Uproszczenie polega na zdefiniowaniu wprost tras statków, które napadają piraci, jako odcinków w obszarze gry oraz uproszczeniu modelu ataku do sytuacji, gdzie piraci wybierają punkt czasowy i statek, a następnie poruszają się ze statkiem przez ustalone k rund, żeby przeprowadzić atak. Takie podejście znacznie zmniejsza przestrzeń decyzyjną piratów, co było koniecznie, ze względu na brak możliwości zastosowania wspomnianej metody generowania kolumn w grze o niezerowej sumie.

Formalnie zmodyfikowana gra opisana jest przez:

- prostokąt o wymiarach $w \times h$. Na prostokąt nałożona jest siatka punktów Z (w przeciwieństwie do oryginalnego modelu tu rozważamy punkty, a nie całe strefy). Sąsiednie punkty są od siebie odległe o d zarówno w pionie, jak i w poziomie,
- n kolejnych, równoodległych punktów czasowych $\tau = t_1, \ldots, t_n$,
- n_f statków oznaczonych $F_1, \ldots F_{n_f}$, które należy bronić. Każdy ze statków pływa pomiędzy dwoma węzłami siatki Z, z prędkością v_f, zgodnie z założonym rozkładem. Każdy statek jest opisany krotką $F_j = \left(u_j^{a+}, u_j^{a-}, u_j^{d+}, u_j^{d-}, S_j\right)$, gdzie
  - $u_j^{a+} \in \mathbb{R}^+$ to wypłata piratów za skuteczny atak na statek,
  - $u_j^{a-} \in \mathbb{R}^-$ to kara piratów za złapanie w trakcie ataku na statek,
  - $u_j^{d-} \in \mathbb{R}^-$ to kara obrońcy za skuteczny atak na statek,
  - $u_j^{d+} \in \mathbb{R}^+$ to nagroda obrońcy za złapanie piratów w trakcie ataku na statek,
  - $S_j = (a_j, b_j, D_j)$, rozkład jazdy statków, gdzie $a_j, b_j \in Z$ — pozycja portu startowego i końcowego, $D_j \in [t_1, t_n]^{w_j}$ — czasy odjazdów statku naprzemiennie z portu a_j i b_j, liczba odjazdów — w_j może być inna dla każdego statku. Czasy muszą być poprawne — zapewnić, że z prędkością v_f można dopłynąć do portu przez kolejnym odjazdem. Ruch statków zdefiniowany jest w ciągłej przestrzeni czasu, chociaż z punktu widzenia zasad gry będziemy rozpatrywać tylko ich pozycje w punktach t_j.
- n_d — liczbę statków obrońcy. Każdy ze statków porusza się z prędkością nie przekraczającą v_d. Każdy ze statków ma przypisany punkt startowy $s_k \in Z$.

- zasięg obrońcy r, który mówi, że jeśli statek obrońcy znajduje się w pewnym punkcie $z \in Z$ to obrońca jest w stanie wykryć atak w kole o promieniu r od tego punku. Zastępuje wykrywanie w strefie i sąsiednich strefach w oryginalnej grze,

- liczby kroków czasowych, które zajmuje atak λ.

Przebieg gry. W każdym kroku czasowym statki, które należy chronić zajmują pozycję wynikającą z rozkładu jazdy. Piraci w trakcie gry wykonują dokładnie jeden atak wybierając parę $(\hat{F}, t_a)$ złożoną ze statku, który atakują i czasu rozpoczęcia ataku. Strategia prosta obrońcy to przypisanie pozycji każdego statku obrońcy j w każdym kroku czasowym t_i pozycji $l(t_i,j) \in Z$, tak żeby: (1) pozycje tego samego statku w kolejnych krokach czasowych były oddalone o nie więcej niż $v_d\mathbf{t}$, gdzie $\mathbf{t}$ to odległość między kolejnymi punktami czasowymi, (2) pierwsza pozycja była w odległości nie większej niż $v_d\mathbf{t}$ od punktu startowego danego statku. Jeśli w przynajmniej jednym punkcie czasowym z przedziału $[t_a, t_{a+\lambda}]$ statek $\hat{F}$ znajduje się w odległości nie większej niż r od któregoś ze statków obrońcy, to atak zostaje wykryty i gra kończy się stosownymi wypłatami. Jeśli czas rozpoczęcia ataku t_a jest zbyt późny, żeby zakończyć atak ($a + \lambda > n$) i atak nie został wykryty, gra kończy się zerową wypłatą dla obu stron. Jeśli atak nie został wykryty i się zakończył, gra kończy się odpowiednimi wypłatami dla skutecznego ataku na ten statek.

Zmodyfikowany program liniowy. Zmodyfikowany program łączy w sobie zmienne reprezentujące pokrycie celów wykorzystane w oryginalnym programie oraz dekompozycję iloczynu prawdopodobieństw strategii naśladowcy (piratów) i lidera (obrońcy) z metody DOBSS, opisanej w Rozdziale 3.2. W programie liniowym, z uwagi na wspomnianą dekompozycję, zamiast stosować jedną rodzinę zmiennych opisującą, tym rzem nie prawdopodobieństwo przebywania w strefach, ale w węzłach siatki, będziemy mieli oddzielne rodziny opisujące pokrycie dla każdej strategii prostej piratów. Konieczne jest, na wzór programu DOBSS wprowadzenie zmiennych binarnych decydujących o wyborze dokładnie jednej strategii naśladowcy. W opisywanym programie liniowym będą występować zmienne $q_{\pi_f} \in \{0, 1\}$, dla każdej strategii naśladowcy $\pi_f \in \Pi_f$. Aby zapewnić, że dokładnie jedna strategia naśladowcy zostanie wybrana, formułujemy ograniczenie analogiczne do ograniczenia (3.10) w DOBSS:

$$\sum_{\pi_f \in \Pi_f} q_{\pi_f} = 1. \tag{3.40}$$

Zmienne definiujące pokrycie poszczególnych punktów siatki będziemy indeksować trzema wartościami: c_{z,t_i,π_f}, gdzie z to węzeł siatki, t_i to krok czasowy, a dodatkowy indeks π_f to strategia prosta naśladowcy, odpowiednik indeksu π_f przy zmiennych p w programie DOBSS.

Zmienne c_{z,t_i,π_f} będą związane ograniczeniami przepływu analogicznymi do (3.35)–(3.38) w pierwotnym programie SMOS, z zastrzeżeniem faktu, że w uproszczonym sformułowaniu

programu nie mamy zbioru T punktów, gdzie obrońca musi skończyć patrol. Zdefiniujemy oddzielną rodzinę ograniczeń dla każdej strategii π_f:

$$c_{t_i,z_j,\pi_f} = \sum_{z_k \in N(z_j) \cup \{z_j\}} \phi_{(z_j,t_i),(z_k,t_{i+1}),\pi_f} \quad \forall z_j \in Z \quad \forall k \in \{1,\dots n-1\} \quad \forall \pi_f \in \Pi_f \quad (3.41)$$

$$c_{t_i,z_j,\pi_f} = \sum_{z_k \in N(z_j) \cup \{z_j\}} \phi_{(z_k,t_{i-1}),(z_j,t_i),\pi_f} \quad \forall z_j \in Z \quad \forall k \in \{2,\dots n\} \quad \forall \pi_f \in \Pi_f \quad (3.42)$$

$$q_{\pi_f} \cdot m = \sum_{z_k \in S} c_{z_k,t_1,\pi_f} \quad \forall \pi_f \in \Pi_f. \quad (3.43)$$

Podobnie jak w programie DOBSS, w obszarze dopuszczalnym tylko jedna rodzina zmiennych c będzie miała niezerowe wartości, co jest wymuszone przez przemnożenie przez q_{π_f} lewej strony równania (3.43).

Zdefiniujemy pomocniczą funkcję $dpp : \Pi_f \times \Pi_f \times \mathbf{c} \to \mathbb{R}$, która będzie opisywać prawdopodobieństwo wykrycia ataku będącego drugim argumentem funkcji, przy założeniu odwiedzania punktów z, według wartości pokrycia ze zmiennych $\mathbf{c}$ z ostatnim indeksem będącym strategią naśladowcy podaną jako pierwszy argument. Jako symbol pomocniczy zdefiniujmy zbiór $R(\pi_f)$, który będzie zawierał takie pary (t_i, z_j), gdzie t_i — punkt czasowy, z_j punkt w siatce, które zgodnie z rozkładem jazdy statku atakowanego w strategii π_f są w odległości co najwyżej r od tego pozycji tego statku w punkcie czasowym t_i oraz punkt t_i jest w zakresie trwania ataku, czyli pomiędzy startem ataku t_a, a jego końcem $t_{a+\lambda}$. Innymi słowy dokładnie tych punktów, które spowodują wykrycia ataku π_f. Funkcja dpp, podobnie jak w oryginalnym sformułowaniu, i zastrzeżeniem tej samej niedokładności związanej z warunkowym prawdopodobieństwem w kolejnych krokach czasu, jest wyrażona jako suma prawdopodobieństw przebywania w węzłach.

$$dpp(\pi_g, \pi_f, \mathbf{c}) = \min \left\{ \sum_{(t_i,z_j) \in R(\pi_f)} c_{t_i,z_j,\pi_g}, 1 \right\} \quad (3.44)$$

Pozostałe składniki sumy, które były obecne w pierwotnym sformułowaniu dpp są pominięte, ponieważ uproszczony model gry zakłada wykrycie z prawdopodobieństwem 1 w promieniu r i nie zakłada istnienia alarmu. Program można łatwo rozszerzyć o te cechy, jednak nie są one kluczowe w demonstracji przyrostu złożoności po wprowadzeniu niezerowej sumy. Parametryzowanie funkcji dpp dwiema strategiami naśladowcy, a nie jedną, jest konieczne do zbudowania programu liniowego z ograniczeniem odpowiadającym ograniczeniu (3.15) w DOBSS, w którym tez pojawiają się dwie strategie naśladowcy — jedna definiująca strategię wynikającą z wartości zmiennych p i jedna definiująca potencjalną odpowiedź naśladowcy, co do której upewniamy się, że nie jest najlepsza odpowiedzią.

W programie DOBSS funkcja celu (3.12) była wyrażona sumą, której dokładnie jeden składnik był niezerowy. W tym programie potrzeba aby uzyskać taką samą sumę funkcję po-

mocniczą, która przyjmie wartość wypłaty lidera lub 0 w zależności od tego, która zmienna q jest niezerowa. Dla odróżnienia od faktycznej wypłaty gracza oznaczymy tę funkcję wielka literą U. Dodatkową komplikacją funkcji U jest fakt, że w zmodyfikowanym sformułowaniu wprowadziliśmy kary dla piratów za bycie złapanym (w pierwotnej wersji w takiej sytuacji wypłata wynosiła 0). Zdefiniujmy przeciążenie notacji $u_i^{\pm}(\pi_f)$ jako wypłatę i-tego gracza za odpowiednio skuteczny lub nieskuteczny atak na cel wynikający ze strategii π_f, wtedy:

$$U_d(\mathbf{c}, \pi_g, \pi_f) = q_{\pi_g} \cdot (dpp(\pi_g, \pi_f, \mathbf{c})u_d^+(\pi_f) + (1 - dpp(\pi_g, \pi_f, \mathbf{c}))u_d^-(\pi_f)) \tag{3.45}$$

$$U_a(\mathbf{c}, \pi_g, \pi_f) = q_{\pi_g} \cdot (dpp(\pi_g, \pi_f, \mathbf{c})u_a^-(\pi_f) + (1 - dpp(\pi_g, \pi_f, \mathbf{c}))u_a^+(\pi_f)). \tag{3.46}$$

Tak zdefiniowane funkcje U nie są liniowe względem zmiennych w programie — zawierają iloczyn q_{π_f} i zmiennych c. Kolejną komplikacją związaną z funkcją dpp i jej implementacją w programie liniowym jest fakt, że po rezygnacji z założenia o sumie zerowej nie jest możliwe zastosowanie prostej techniki implementującej minimum tylko dwoma ograniczeniami nierównościowymi. Zwróćmy jednak uwagę na fakt, że ograniczenie (3.43) wymusza, że $dpp(\pi_g, \pi_f, \mathbf{c})$ będzie zerem, jeśli q_{π_g} jest zerem, więc w tej części sumy mnożenie przez q_{π_g} jest zbędne. Wprowadzimy teraz funkcję $dpp_{neg}(\pi_g, \pi_f, \mathbf{c})$, która przyjmie wartość $(1 - dpp(\pi_g, \pi_f, \mathbf{c}))$, gdy q_{π_g} jest jedynką i 0 w przeciwnym przypadku.

$$dpp_{neg}(\pi_g,\pi_f, \mathbf{c}) = \min\left\{q_{\pi_g}, 1 - dpp(\pi_g,\pi_f,)\right\} \tag{3.47}$$

Sformułowanie z użyciem takiego minimum jest wystarczające, ponieważ $1 - dpp(\pi_g,\pi_f,)$ nigdy nie przekracza 1.

Dysponując funkcją $dpp_{neg}(\pi_g,\pi_f,)$ możemy wyrazić $U_i^{\pm}$ w następujący sposób:

$$U_d(\mathbf{c}, \pi_g, \pi_f) = dpp(\pi_g, \pi_f,)u_d^+(\pi_f) + dpp_{neg}(\pi_g, \pi_f,)u_d^-(\pi_f) \tag{3.48}$$

$$U_a(\mathbf{c}, \pi_g, \pi_f) = dpp(\pi_g, \pi_f,)u_a^-(\pi_f) + dpp_{neg}(\pi_g, \pi_f,)u_a^+(\pi_f), \tag{3.49}$$

a te sformułowania są liniowe względem zmiennych, z dokładnością do użycia minimum. Jak wspomnieliśmy gra o niezerowej sumie powoduje, że wartość funkcji celu nie jest wprost proporcjonalna do wartości minimum. To powoduje konieczność wprowadzenia kolejnych zmiennych binarnych, żeby to minimum wyrazić. Podręcznikowy schemat, pozwalający wyrazić dowolne minimum $x = \min\{f_1(a), f_2(a)\}$ w programie liniowy m z wykorzystaniem zmiennych

binarnych jest następujący:

$$f_1(a) - m_1 M \leqslant x \leqslant f_1(a)$$
$$f_2(a) - m_2 M \leqslant x \leqslant f_2(a)$$
$$m_1 + m_2 = 1.$$

m_1 i m_2 są zmiennymi binarnymi, a M jest odpowiednio dużą stałą. Powyższy układ ograniczeń wymaga, żeby dla m_i, które ma wartość 1 zachodziła równość z odpowiednią funkcją $f_i(a)$, a dla drugiej zachodziła relacja mniejsze równe.

Do budowanego programu dodajemy rodziny zmiennych $dpp(\pi_g, \pi_f)$ i $dpp_{neg}(\pi_g, \pi_f)$ reprezentujące odpowiednie wartości funkcji o odpowiadających nazwach. W związku z tym będziemy mieli dwie rodziny minimów (po jednym dla każdej pary strategii (π_g, π_f) piratów dla definicji dpp i tyle samo dla definicji dpp_{neg}), w związku z tym będziemy mieli cztery rodziny zmiennych binarnych m. Zmienne dotyczące definicji dpp oznaczymy symbolami $m_1(\pi_g, \pi_f)$ oraz $m_2(\pi_g, \pi_f)$, a zmienne dotyczące definicji dpp_{neg} odpowiednio $m_3(\pi_g, \pi_f)$ oraz $m_4(\pi_g, \pi_f)$. W efekcie do programu liniowego dodamy rodziny ograniczeń realizujących równania (3.48) i (3.49):

$$1 - m_1(\pi_g,\pi_f)M \leqslant dpp(\pi_g, \pi_f) \leqslant 1 \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.50)$$

$$\sum_{(t_i,z_j)\in R(\pi_f)} c_{t_i,z_j,\pi_g} - m_2(\pi_g,\pi_f)M \leqslant dpp(\pi_g,\pi_f) \leqslant \sum_{(t_i,z_j)\in R(\pi_f)} c_{t_i,z_j,\pi_g} \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.51)$$

$$m_1(\pi_g,\pi_f) + m_2(\pi_g,\pi_f) = 1 \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.52)$$

$$q_{\pi_g} - m_3(\pi_g,\pi_f)M \leqslant dpp_{neg}(\pi_g, \pi_f) \leqslant q_{\pi_g} \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.53)$$

$$1 - dpp(\pi_g,\pi_f) - m_4(\pi_g,\pi_f)M \leqslant dpp_{neg}(\pi_g, \pi_f) \leqslant 1 - dpp(\pi_q,\pi_f) \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.54)$$

$$m_3(\pi_g,\pi_f) + m_4(\pi_g,\pi_f) = 1 \qquad \forall \pi_g, \pi_f \in \Pi_f \quad (3.55)$$

Dla czytelności końcowego programu zdefiniujemy jeszcze dwa symbole pomocnicze realizujące funkcje opisane równaniami (3.48) i (3.49).

$$U_a(\pi_g, \pi_f) = dpp(\pi_g, \pi_f)u_a^-(\pi_f) + dpp_{neg}(\pi_g, \pi_f)u_a^+(\pi_f)$$
$$U_d(\pi_g, \pi_f) = dpp(\pi_g, \pi_f)u_d^+(\pi_f) + dpp_{neg}(\pi_g, \pi_f)u_d^-(\pi_f)$$

Po zdefiniowaniu wszystkich obiektów pomocniczych, możemy zdefiniować ograniczenie wymuszające optymalną odpowiedź naśladowcy (piratów), odpowiednik ograniczenia (3.15) w DOBSS:

$$0 \leqslant a - \sum_{\pi_g\in\Pi_f} U_a(\pi_g, \pi_f) \leqslant (1 - q_{\pi_f}) \cdot M \qquad \forall \pi_f \in \Pi_f \quad (3.56)$$

gdzie a jest zmienną rzeczywistą. To ograniczenie zapewnia, że strategia π_f, dla której $q_{\pi_f} = 1$ jest optymalną odpowiedzą naśladowcy. Ostatnim opisywanym elementem, poniekąd najważ-

niejszym, jest funkcja celu, również analogiczna jak w DOBSS:

$$\max \sum_{\pi_f \in \Pi_f} U_d(\pi_f, \pi_f), \tag{3.57}$$

dla czytelności równania zdecydowaliśmy się pominąć zmiennych decyzyjnych przy symbolu maksimum. Jest to oczywiście zbiór wszystkich zmiennych wprowadzonych w tym programie liniowym. Poniżej przedstawiamy jeszcze raz funkcję celu i zebrane ograniczenia (3.40)–(3.43), (3.50)–(3.55), (3.56), definiujące pełny program liniowy:

$$\begin{aligned}
&\max \sum_{\pi_f \in \Pi_f} U_d(\pi_f, \pi_f) \\
0 \leqslant a - \sum_{\pi_g \in \Pi_f} U_a(\pi_g, \pi_f) &\leqslant (1 - q_{\pi_f}) \cdot M \quad \forall \pi_f \in \Pi_f \\
1 - m_1(\pi_g,\pi_f)M \leqslant dpp(\pi_g, \pi_f) &\leqslant 1 && \forall \pi_g, \pi_f \in \Pi_f \\
\sum_{(t_i,z_j) \in R(\pi_f)} c_{t_i,z_j,\pi_g} - m_2(\pi_g,\pi_f)M \leqslant dpp(\pi_g,\pi_f) &\leqslant \sum_{(t_i,z_j) \in R(\pi_f)} c_{t_i,z_j,\pi_g} && \forall \pi_g, \pi_f \in \Pi_f \\
m_1(\pi_g,\pi_f) + m_2(\pi_g,\pi_f) &= 1 && \forall \pi_g, \pi_f \in \Pi_f \\
q_{\pi_g} - m_3(\pi_g,\pi_f)M \leqslant dpp_{neg}(\pi_g, \pi_f) &\leqslant q_{\pi_g} && \forall \pi_g, \pi_f \in \Pi_f \\
1 - dpp(\pi_g,\pi_f) - m_4(\pi_g,\pi_f)M \leqslant dpp_{neg}(\pi_g, \pi_f) &\leqslant 1 - dpp(\pi_q,\pi_f) && \forall \pi_g, \pi_f \in \Pi_f \\
m_3(\pi_g,\pi_f) + m_4(\pi_g,\pi_f) &= 1 && \forall \pi_g, \pi_f \in \Pi_f \\
\sum_{\pi_f \in \Pi_f} q_{\pi_f} &= 1
\end{aligned}$$

$$\begin{aligned}
c_{t_i,z_j,\pi_f} &= \sum_{z_k \in N(z_j) \cup \{z_j\}} \phi_{(z_j,t_i),(z_k,t_{i+1}),\pi_f} && \forall z_j \in Z \quad \forall k \in \{1, \dots n-1\} \quad \forall \pi_f \in \Pi_f \\
c_{t_i,z_j,\pi_f} &= \sum_{z_k \in N(z_j) \cup \{z_j\}} \phi_{(z_k,t_{i-1}),(z_j,t_i),\pi_f} && \forall z_j \in Z \quad \forall k \in \{2, \dots n\} \quad \forall \pi_f \in \Pi_f \\
q_{\pi_f} \cdot m &= \sum_{z_k \in S} c_{z_k,t_1,\pi_f} && \forall \pi_f \in \Pi_f
\end{aligned}$$

$$\begin{aligned}
a \in \mathbb{R} \qquad & \forall \pi_f \in \Pi_f \quad q_{\pi_f} \in \{0,1\} \\
\forall \pi_f, \pi_g \in \Pi_f \; dpp(\pi_g, \pi_f), dpp_{neg}(\pi_g, \pi_f) \in \mathbb{R} \qquad & \forall i \in \{1,2,3,4\} \forall \pi_f, \pi_g \in \Pi_f m_i(\pi_g, \pi_f) \in \{0,1\} \\
\forall t_i \in \tau, z_j \in Z, \pi_f \in \Pi_f c_{t_i,z_j,\pi_f} \in \mathbb{R} \qquad & \forall t_i \in \tau, z_j, z_k \in Z, \pi_f \in \Pi_f \phi_{(t_i,z_j),(t_{i+1},z_k),\pi_f} \in \mathbb{R}
\end{aligned}$$

Wnioski z modyfikacji. Odejście od tylko jednego założenia, zerowej sumy gry, wymagało modyfikacji programu linowego wymagającej dogłębne zrozumienie jego konstrukcji. Co więcej zmodyfikowany program liniowy posiada $|\Pi_f|$ razy więcej zmiennych rzeczywistych, a po-

nadto $O(|\Pi_f|^2)$ zmiennych binarnych, których nie było w pierwotnej wersji. To oznacza, że rozwiązanie takiego programu wymaga dużo więcej zasobów obliczeniowych, zmniejszając konkurencyjność czasową tego rozwiązania względem metod ogólnych. Ta modyfikacja jest przykładem na to, że uogólnienie metod wykorzystujących własności konkretnych gier nie jest proste i często niesie za sobą duże straty w kontekście wydajności zmodyfikowanej metody.

3.6 Metody wykorzystujące podwójną wyrocznię

Zupełnie inną grupą metod, stosowaną głównie do gier o sumie zerowej, są metody stosujące podejście podwójnej (lub naprzemiennej) wyroczni (ang. double oracle) [62]. Podejście to polega na budowie dwóch procedur (wyroczni), jednej która przy ustalonej strategii drugiego gracza, znajduje optymalną (w pewnym, założonym, sensie) strategię gracza pierwszego i drugą wyrocznię, która znajduje strategię gracza drugiego przy ustalonej strategii gracza pierwszego. Następnie znajdowany jest stan równowagi w którym uwzględniamy tylko strategie będące optymalnymi odpowiedziami we wszystkich poprzednich iteracjach. Iteracyjne znajdowanie optymalnych odpowiedzi pozwala na zawężenie przeglądanych strategii w grze tylko do tych, które są istotne — są optymalnymi odpowiedziami przeciwko pewnym strategiom innych graczy. Idea podwójnej wyroczni jest zbliżona do metod generowania kolumn i ograniczeń w programach liniowych, choć sama realizacja jest zupełnie inna. Tu zaprezentujemy metodę służącą poszukiwaniu Równowagi Nasha w grach o sumie zerowej opublikowaną w roku 2014 [16]. Zanim to jednak nastąpi, należy odnotować, że zbliżonym podejściem, które jest stosowane w literaturze jest metoda fikcyjnej rozgrywki (ang. fictitious play) [12, 21], która polega na rozpoczęciu poszukiwania stanu równowagi od pewnego profilu strategii, a następnie uaktualnianie strategii każdego z graczy z osobna, przy założeniu, że strategie pozostałych są niezmienne. Takie podejście, przy odpowiednio dobranym sposobie aktualizacji pozwala uzyskać zbieżność w pewnych klasach gier o sumie zerowej lub gier z pełną informacją [60, 94]. Oba podejścia mają naturę iteracyjną, jednak metoda z podwójną wyrocznią kładzie nacisk na ograniczenie liczby rozpatrywanych strategii graczy, co jest szczególnie istotne w grach wielokrokowych.

Wspomniana metoda dedykowana poszukiwaniu Równowagi Nasha w grach w postaci ekstensywnej o sumie zerowej [16] wykorzystuje postać sekwencyjną wprowadzoną w Rozdziale 2.4.2, żeby umożliwić rozważanie strategii pojedynczego gracza w zwartej formie. Z racji tego, że w tym podrozdziale będziemy rozważać Równowagę Nasha, a nie Równowagę Stackelberga, przyjmiemy że zbiór graczy to $\mathcal{N} = \{1,2\}$, jako że żaden z graczy nie ma wyróżnionej pozycji. Będziemy tez stosować notację gracz $-i$, co będzie oznaczało przeciwnik gracza i, czyli odpowiednio $-1 := 2$ i $-2 := 1$.

Ogólny zarys metody podwójnej wyroczni (niekoniecznie w grach w postaci ekstensywnej lub sekwencyjnej) jest następujący:

1. weź podgrę oryginalnej gry, która zawiera niewielki podzbiór strategii każdego z graczy,

2. znajdź stan równowagi w podgrze.

3. Zapytaj każdą z wyroczni o najlepszą możliwą odpowiedź przy strategii przeciwnika ze znalezionego stanu równowagi. Jeśli któraś z tych strategii nie jest w podgrze, rozszerz podgrę o te strategie i przejdź do punktu 2. W przeciwnym przypadku zakończ algorytm.

W przypadku omawianej metody, gdzie stosujemy postać sekwencyjną, początkowa podgra może nie zawierać wszystkich możliwych zbiorów informacyjnych, a co za tym idzie, w pełnej grze będą istniały sekwencje, których nie da się zagrać w podgrze. Głównym wkładem przytoczonej pracy jest zastosowanie technik, które pozwolą zastosować metodę podwójnej wyroczni i zagwarantują jej zbieżność mimo tego problemu.

Pierwszym krokiem jest zdefiniowanie pojęcia strategii domyślnej — w każdym zbiorze informacyjnym wyznaczamy jeden ruch jako ruch domyślny, który będzie używany do wyliczania wypłat, gdy dany zbiór informacyjny nie jest częścią podgry. Nie chodzi tu o zastosowanie heurystyki wybierającej ruch w jakikolwiek sposób obiecujący lub optymalny, a deterministyczny wybór ruchu. Z racji tego, że mówimy o podejściu algorytmicznym, może to być po prostu ruch w pierwszej komórce tablicy dostępnych ruchów. Zastosujemy oznaczenie $D : \mathcal{I} \to \bigcup_{A \in \mathcal{A}} A$, który każdemu zbiorowi informacyjnemu $I \in \mathcal{I}$ przypisze domyślną akcję ze zbioru $A(I)$.

Korzystając z domyślnych strategii można dla zadanej gry $\Gamma = (\mathcal{N}, \mathcal{S}, \mathcal{Z}, \rho, \mathcal{A}, u, \mathcal{I})$ zdefiniować podgrę indukowaną przez pewne podzbiory sekwencji pierwszego (Σ_1') oraz drugiego (Σ_2') z graczy $\Gamma'(\Sigma_1', \Sigma_2') = (\mathcal{N}, \mathcal{S}', \mathcal{Z}', \rho', \mathcal{A}', u', \mathcal{I}')$. Poza zbiorem graczy, który pozostaje niezmieniony, elementy podgry są definiowane przez wybrane zbiory sekwencji w sposób opisany poniżej. Dla uproszczenia można założyć, że zbiory Σ_1' i Σ_2' są domknięte ze względu na operację prefiksu, to znaczy że jeśli pewna sekwencja σ należy do zbioru, to należą do niego wszystkie prefiksy tej sekwencji, włączając w to prefiks pusty.

- Zbiory $\mathcal{S}'$ i $\mathcal{Z}'$. Niech H' — zbiór takich węzłów drzewa gry, (zarówno nieterminalnych, jak i terminalnych), dla których sekwencje ruchów graczy, które do nich prowadzą należą do odpowiednich zbiorów Σ_i', formalnie $H' = \{h \in \mathcal{S} \cup \mathcal{Z} | \sigma_1(h) \in \Sigma_1' \wedge \sigma_2(h) \in \Sigma_2'\}$ Zbiory Σ_i' są domknięte ze względu na prefiksy, w związku z czym w zbiorze H' mamy do czynienia nadal z drzewem i korzeń oryginalnego drzewa jest korzeniem nowego drzewa. Nie jest zachowany natomiast stan bycia węzłem wewnętrznym lub liściem. Część węzłów wewnętrznych mogła się stać liśćmi, ponieważ do zboru H' nie trafiły ich następniki. W związku z tym zbiór $\mathcal{S}'$ będzie zawierał tylko te elementy H', które mają następniki w H' a zbiór $\mathcal{Z}'$ pozostałe elementy, formalnie $\mathcal{S}' = \{s \in H' | \exists a \in A_s \exists s' \in H' \sigma_{\rho(s)}(s') = \sigma_{\rho(s)}(s)a\}$, natomiast $\mathcal{Z}' = \mathcal{Z}_z' \cup \mathcal{Z}_n'$, gdzie $\mathcal{Z}_z' = H' \cap \mathcal{Z}$, a $\mathcal{Z}_n' = H' \cap \mathcal{S} \backslash \mathcal{S}'$. Zbiory $\mathcal{Z}_z'$ i $\mathcal{Z}_n'$ to odpowiednio liście które były liśćmi również w oryginalnej grze i liście „nowe". Podział ten będzie istotny przy definiowaniu funkcji wypłat.

- Funkcja definiująca gracza wykonującego ruch będzie oryginalną funkcją ρ z dziedziną obciętą do zbioru $\mathcal{S}'$, $\rho' = \rho|_{\mathcal{S}'}$.

- Podobnie rodzina zbiorów akcji jest obcięta po pierwsze tylko do akcji wychodzących z węzłów $\mathcal{S}'$, a po drugie każdy ze zbiorów akcji jest obcięty tylko do tych akcji, które prowadzą do węzłów w zbiorze H', czyli nowym drzewie gry. Dla każdego zbioru akcji A_s zdefiniujmy zbiór akcji w grze ograniczonej $A'_s = \{a \in A_s | \exists s' \in \mathcal{S}' \quad seq_{\rho(s)}(s') = seq_{\rho(s)}(s)a$. Zbiór $\mathcal{A}'$ będzie zawierał wszystkie niepuste ograniczone zbiory akcji $\mathcal{A}' = \{A'_s | A'_s \neq \varnothing\}$.

- Struktura zbiorów informacyjnych zostaje zachowana, ale zawiera tylko węzły ze zbioru $\mathcal{S}'$. $\mathcal{I}' = \{I \cap \mathcal{S}' | I \in \mathcal{I} \wedge I \cap \mathcal{S}' \neq \varnothing\}$.

- Jedynym nietrywialnym elementem są funkcje wypłat u'_i. Dla liści w podgrze, które były również liśćmi w oryginalnej grze, wypłaty pozostają niezmienione. Konieczne jest jednak zdefiniowanie wypłat dla nowych liści. Dla każdego z graczy i funkcja wypłaty $u'_i(z')$, będzie wynosiła wypłatę z pierwotnej gry $u_i(z')$, jeśli z' był również liściem w oryginalnej grze. W przeciwnym wypadku wypłata gracza i jest ustawiana na wynik rozgrywki, gdzie gracz i gra od punktu z' strategię domyślną D_i, a gracz $-i$ gra najlepszą odpowiedź (efektywnie jest to gra dla jednego gracza — $-i$, więc wyliczenie najlepszej odpowiedzi jest dużo mniej kosztowne niż w grze dla dwóch graczy). Tak dobrane wypłaty maja dwie cechy: po pierwsze jest to dolne oszacowanie wypłaty gracza i. Po drugie powoduje, że gra przestaje spełniać warunek zerowej sumy. Taki dobór wypłat jest jednak konieczny do gwarantowanej zbieżności całego procesu, co autorzy dowodzą w Twierdzeniu 5.5 w pracy [16].

Algorytm 3.4 przedstawia działanie opisywanej metody. Rozpoczynając od podgry indukowanej przez małą liczbę sekwencji każdego z graczy, następnie podgra rozwiązywana jest programem liniowym zaprezentowanym w kolejnym akapicie, który znajduje Równowagę Nasha korzystając z postaci sekwencyjnej gry o sumie zerowej. W następnym kroku dla każdego z graczy stosowana jest wyrocznia, która znajduje sekwencję ruchów gracza będącą optymalną odpowiedzią na strategię przeciwnika znalezioną przez program liniowy. Jeśli któraś z wyroczni znajdzie sekwencję nie należącą jeszcze do podgry, podgra jest rozszerzana i procedura jest powtarzana.

Poszukiwanie Równowagi Nasha realizowane jest poprzez rozwiązanie poniższego pro-

Algorytm 3.4: Metoda poszukiwania Równowagi Nasha w grach w postaci ekstensywnej o sumie zerowej, wykorzystująca mechanizm podwójnej wyroczni [16].

$\Sigma'_1 \leftarrow$ nieduży podzbiór sekwencji gracza 1;
$\Sigma'_2 \leftarrow$ nieduży podzbiór sekwencji gracza 2;
$\Sigma_1 \leftarrow \varnothing$;
$\Sigma_2 \leftarrow \varnothing$;
while $\Sigma'_1 \neq \Sigma_1 \vee \Sigma'_2 \neq \Sigma_2$ **do**
    $\Sigma_1 \leftarrow \Sigma'_1$;
    $\Sigma_2 \leftarrow \Sigma'_2$;
    Rozwiąż grę $\Gamma'(\Sigma_1, \Sigma_2)$ programem liniowym (3.58)–(3.62);
    $\Sigma^*_1 \leftarrow$ najlepsza odpowiedź gracza 1 według wyroczni; $\Sigma^*_2 \leftarrow$ najlepsza odpowiedź gracza 2 według wyroczni; $\Sigma'_1 \leftarrow \Sigma_1 \cup \Sigma^*_1$;
    $\Sigma'_2 \leftarrow \Sigma_2 \cup \Sigma^*_2$;
end

gramu liniowego dla każdego z graczy $i = 1{,}2$ [75, Rozdział 5]:

$$\max_{\psi, w} \quad v_{I_{-i}(\varnothing)} \tag{3.58}$$

z zachowaniem ograniczeń

$$v_{I_{-i}(\sigma_{-i})} - \sum_{I' \in \mathcal{I}_{-i} | \sigma_{-i}(I') = \sigma_{-i}} v_{I'} \leqslant \sum_{\sigma_i \in \Sigma_i} u_i(\sigma_i, \sigma_{-i}) \psi_i(\sigma_i) \qquad \forall \sigma_{-i} \in \Sigma_{-i} \tag{3.59}$$

$$\psi_i(\varnothing) = 1 \tag{3.60}$$

$$\sum a \in A(I_i) \psi(\sigma_i a) = \psi(\sigma_i) \qquad \forall \sigma_i = \sigma(I_i) | I_i \in \mathcal{I}_i \tag{3.61}$$

$$\psi_i(\sigma_i) \leqslant 0 \qquad \forall \sigma_i \in \Sigma_i. \tag{3.62}$$

Program ten daje prawidłowy wynik tylko w przypadku gier o sumie zerowej. Ograniczenia (3.60)—(3.62) gwarantują, że zmienne ψ definiują poprawny plan realizacji strategii w postaci sekwencyjnej i odpowiadają wprost warunkom z Definicji 2.23. Ograniczenie (3.59) wymusza założenie że przeciwnik wybierze taką strategię, która będzie jego najlepsza odpowiedzią. Z racji tego, że rozwiązujemy ten sam program dla obu graczy, gwarantuje to, że znaleziona para strategii będzie stanem Równowagi Nasha.

Algorytm wyroczni. Algorytm wyroczni opiera się o przeszukiwanie drzewa oryginalnej gry z perspektywy gracza i. Przed przeszukiwaniem strategia gracza $-i$ wyliczona dla podgry jest uzupełniana do strategii w pełnej grze poprzez wybór ruchów według strategii domyślnej we wszystkich zbiorach informacyjnych, które nie należą do podgry. Dla tak ustalonej strategii przeciwnika, przeszukiwane jest drzewo gracza i. Przeszukiwanie realizowane jest metodą wgłąb, z zapamiętywaniem najlepszego dotąd znalezionego rezultatu, co pozwala obciąć część gałęzi przeszukiwania, które nie mają szansy poprawić najlepszego znanego rozwiązania.

Tak zbudowana metoda iteracyjna zbiega do Równowagi Nasha dla każdej gry o sumie ze-

rowej, co autorzy pokazują w Twierdzeniu 5.5 [16]. Analizując konstrukcję zaproponowanej tu metody można zauważyć, że, podobnie jak w metodach generowania kolumn i ograniczeń, położono tu nacisk na ograniczenie przestrzeni dostępnych ruchów graczy (tu konkretnie sekwencji). To wzmacnia obserwację, której dokonaliśmy przy poprzednich metodach, że kluczowe w grach wielokrokowych jest uniknięcie przeszukiwania wszystkich możliwych profili strategii graczy. Metoda podwójnej wyroczni jest wyjątkowo interesująca z punktu widzenia „miękkich" metod sztucznej inteligencji, co zauważają autorzy przytaczanej pracy, ponieważ zastępując wyrocznię heurystyczną metodą przeszukiwania (potencjalnie bez gwarancji wyboru najlepszej możliwej sekwencji) na przykład próbkowaniem Monte Carlo można osiągnąć jeszcze krótsze czasy obliczeń.

3.7 Metody dedykowane całej klasie gier wielokrokowych o sumie niezerowej

Wszystkie zaprezentowane dotąd podejścia były dedykowane specyficznym podklasom gier. Techniki stosowane w tych podejściach stanowią inspirację dla metod proponowanych w Rozdziale 4 rozprawy, jednak nie jest możliwe uczciwe porównanie skuteczności i wydajności tych metod z metodami specyficznymi dla dziedziny. W tym podrozdziale zaprezentujemy trzy metody, które można stosować do klasy gier wielokrokowych z niepełną informacją o sumie niezerowej.

Te trzy metody zostały porównane w eksperymentach prezentowanych w Rozdziale 4 z metodami proponowanymi w rozprawie.

Wszystkie wspomniane metody wykorzystują postać sekwencyjną do efektywnego rozwiązywania gier z użyciem programu liniowego, w związku z tym w tym podrozdziale będziemy wykorzystywać notację wprowadzona wraz z definicją postaci ekstensywnej i postaci sekwencyjnej w rozdziałach 2.4.1 i 2.4.2. Będziemy stosować rozszerzoną notację wypłaty $u_i(\sigma_l, \sigma_f)$. W przypadkach, gdy para sekwencji podana w nawiasach wskazuje na liść gry, będzie to wypłata w tym liściu. W pozostałych przypadkach, to znaczy zarówno, gdy para sekwencji prowadzi do węzła niebędącego liściem, jak i gdy para sekwencji nie jest ze sobą zgodna w myśl Definicji 2.21, będziemy przyjmować odpowiednie $u_i(\sigma_l, \sigma_f) = 0$. Takie podejście uprości zapis, a w praktyce spowoduje że składniki sum liczących wypłaty wyzerują się dla par sekwencji, które nie prowadzą do liści.

3.7.1 Program liniowy wykorzystujący postać sekwencyjną do poszukiwania Równowagi Stackelberga

W roku 2015 została opublikowana praca [17], która proponuje program liniowy znajdujący Równowagę Stackelberga z użyciem sekwencyjnej reprezentacji gry. Opisywany program łą-

czy w sobie cechy programu DOBSS, który działa na grach w postaci normalnej i reprezentacji strategii używanej na przykład w programie poszukującym Równowagi Nasha (3.58)–(3.62). W rozdziałach pracy prezentujących eksperymentalną ewaluacje metod, będziemy tę metodę nazywać *BC2015* od inicjałów autorów i roku publikacji. W zaproponowanym programie liniowym występują następujące rodziny zmiennych:

- $\psi_l(\sigma_l) \qquad \forall \sigma_l \in \Sigma_l \in [0,1]$ – zmienne opisujące plan realizacji lidera, zgodny z Definicją 2.23,
- $\psi_f(\sigma_f) \qquad \forall \sigma_f \in \Sigma_f \in \{0,1\}$ – zmienne binarne opisujące plan realizacji naśladowcy,
- $p(z) \qquad \forall z \in \mathcal{Z}$ – zmienne reprezentujące prawdopodobieństwa znalezienia się w zadanym liściu drzewa gry, gdy gracze wybiorą plan realizacji opisany zmiennymi ψ.
- $v_I \qquad \forall I \in \mathcal{I}$ – zmienne pomocnicze przechowujące wartość oczekiwaną wypłaty naśladowcy po osiągnięciu zadanego zbioru informacyjnego, gdy zagrany zostanie profil strategii definiowany przez zmienne ψ. Te zminne posłużą do zapewnienia, że odpowiedź naśladowcy jest optymalna.
- $s_{\sigma_f} \qquad \forall \sigma_f \in \Sigma_f$ – zmienne *slack*, które zapewnią odpowiednio równy bądź mniejszy równy w poszczególnych warunkach na optymalność odpowiedzi.

Program optymalizuje wartość oczekiwaną wypłaty lidera:

$$\max_{p,\psi,v.s} \quad \sum_{z \in \mathcal{Z}} p(z) u_l(z) \tag{3.63}$$

Plany realizacji obu graczy są ograniczone zgodnie z definicją:

$$\psi_i(\varnothing) = 1 \qquad \forall i \in \mathcal{N} \tag{3.64}$$

$$\sum a \in A(I_i)\psi(\sigma_i a) = \psi(\sigma_i) \qquad \forall i \in \mathcal{N} \qquad \forall \sigma_i = \sigma(I_i) | I_i \in \mathcal{I}_i. \tag{3.65}$$

Następnie zdefiniowano ograniczenia na zmienne p, które mają wskazywać na prawdopodobieństwo odpowiednich liści, czyli mieć wartość iloczynu prawdopodobieństw zagrania odpowiedniej sekwencji ruchów przez każdego z graczy. Korzystając z faktu, że zmienne ψ_f są binarne, można uniknąć zapisywania iloczynu wprost i rozbić to na dwa ograniczenia:

$$0 \leqslant p(z) \leqslant \psi_l(\sigma_l(z)) \qquad \forall z \in \mathcal{Z} \tag{3.66}$$

$$0 \leqslant p(z) \leqslant \psi_f(\sigma_f(z)) \qquad \forall z \in \mathcal{Z}, \tag{3.67}$$

$$\tag{3.68}$$

co gwarantuje, że jeśli odpowiednie $\psi_f(\sigma_f(z))$ jest 0, to $p(z)$ też wyniesie 0. Nie ma jednak gwarancji, że jeśli $\psi_f(\sigma_f(z))$ jest 1, to wartość $p(x)$ osiągnie wartość równą $\psi_l(\sigma_l(z))$. Aby

zagwarantować, że ta wartość będzie nie mniejsza, wymuszono, żeby prawdopodobieństwa liści sumowały się do 1:

$$1 = \sum_{z \in \mathcal{Z}} p(z). \tag{3.69}$$

Ostatnim elementem programu jest wymuszenie, aby sekwencje naśladowcy, dla których $\psi_f(\sigma_f)$ jest 1 definiowały optymalną odpowiedź naśladowcy. Zdefiniowano po jednym ograniczeniu na każdy punkt decyzyjny naśladowcy. Każde z tych ograniczeń będzie gwarantować, ze w danym punkcie decyzyjnym zmiana decyzji nie poprawi wyniku. Technicznie nie będzie to pojedyncze ograniczenie, ale para ograniczeń związana z każdą sekwencją naśladowcy.

$$0 \leqslant s_{\sigma_f} \leqslant (1 - \psi_f(\sigma_f))M \qquad \forall \sigma_f \in \Sigma_f \tag{3.70}$$

$$v_{I_f(\sigma_f)} = s_{\sigma_f} + \sum_{I' \in \mathcal{I}_f | \sigma_f(I') = \sigma_f} v_{I'} + \sum_{\sigma_l \in \Sigma_l} \psi_l(\sigma_l) u_f(\sigma_l, \sigma_f) \qquad \forall \sigma_f \in \Sigma_f \tag{3.71}$$

Pomocnicze ograniczenie (3.70) służy do wymuszenia wartości 0 odpowiedniej zmiennej slack, gdy $\psi_f(\sigma_f)$ jest jedynką, a w pozostałych przypadkach zezwolenie aby odpowiednia zmienna przyjęła dowolną wartość nieujemną. Głównym ograniczeniem jest (3.71). Dodanie zmiennej slack do prawej strony tego ograniczenia należy interpretować jako ograniczenie równościowe, gdy $\psi_f(\sigma_f)$ jest 0 i mniejsze lub równe w przeciwnym przypadku. Jeśli pominiemy zmienną s_{σ_f}, prawa strona tego ograniczenia jest sumą dwóch sum. Zmienne v_I przechowują iloczyn prawdopodobieństwa trafienia do zbioru informacyjnego I i wartości oczekiwanej wypłaty pod warunkiem trafienia do tego zbioru. Pierwsza z sum to wartości oczekiwane z węzłów poniżej zbioru informacyjnego w którym wykonywany był ostatni ruch z sekwencji σ_f. Druga suma, iloczyn prawdopodobieństwa trafienia do danego liścia przy strategii lidera definiowanej przez zmienne $\psi_l(\sigma_l)$ przy założeniu zagrania przez naśladowcę sekwencji σ_f. Ograniczenie mówi, że dla wszystkich sekwencji nie wybranych przez zmienne $\psi_f(\sigma_f)$ ta suma powinna być mniejsza lub równa od odpowiedniego v_I, a dla σ_f, które są wybrane, powinna zachodzić równość. To gwarantuje, że zmienne $\psi_f(\sigma_f)$ wskazują optymalną odpowiedź naśladowcy.

Eksperymenty przeprowadzone przez autorów pracy wskazują jednoznacznie, że ta metoda jest znacznie szybsza od metody DOBSS zastosowanej do gier w postaci ekstensywnej w postaci normalnej po transformacji Harsányiego. Skalowalność tego podejścia jest jednak nadal ograniczona, ze względu na konieczność opisania całej struktury gry jednym programem liniowym. Powoduje to, że dla dużych gier zarówno czas obliczeń, jak i zajętość pamięci staje się problemem.

3.7.2 Metoda oparta o skorelowane stany równowagi

Poza koniecznością uwzględnienia w metodzie BC2015 wszystkich możliwych sekwencji graczy, istotnym problemem jest też duża liczba zmiennych binarnych. Każda sekwencja ruchów naśladowcy odpowiada jednej zmiennej binarnej w programie. Opublikowana niewiele później,

bo w roku 2016 [87] metoda oparta o Skorelowany Stan Równowagi Stackelberga w Grach w Postaci Ekstensywnej (ang. Stakclebrg Extensive Form Correlated Equilibrium, SEFCE) [18] i jego stopniowe poprawianie w celu usyskania Równowagi Stackelberga charakteryzuje się mniejszą liczbą zmiennych całkowitych w rozwiązywanych programach liniowych. W dalszej części pracy będziemy odnosić się do tej metody poprzez *C2016* – inicjał pierwszego autora i rok publikacji. Zanim przystąpimy do wyjaśniania modus operandi tej metody, konieczne jest jednak objaśnienie czym są Skorelowane Stany Równowagi.

Skorelowana Równowaga w grach w postaci normalnej

Najpierw wprowadzimy pojęcie Skorelowanej Równowagi (ang. Correlated Equilibrium) [5] w grach w postaci normalnej. Skorelowana Równowaga polega na udziale w procesie wyboru strategii zewnętrznego bytu niezwiązanego z żadnym z graczy. Byt ten losuje profile strategii prostych w grze (krotki zawierające strategię prostą każdego z graczy), a następnie podaje do wiadomości każdemu z graczy jaka strategia została wylosowana dla niego. Taką wylosowana i przekazaną do wiadomości strategię będziemy nazywać rekomendacją. Każdy z graczy zna tylko wynik losowania dotyczący jego strategi, ale nie wie co wylosowali pozostali gracze. Rozkład prawdopodobieństwa z którego losuje profile byt zewnętrzny jest jednak publicznie znany. Oznacza to, że każdy z graczy jest w stanie wyznaczyć warunkowe prawdopodobieństwa wystąpienia konkretnych strategii przeciwników, gdy on otrzymał pewna strategię. Skorelowanym stanem równowagi nazywamy sytuację, gdy podmiot losujący wybiera profile z takiego rozkładu, że żaden z graczy nie poprawi wartości oczekiwanej swojego wyniku gry poprzez zagranie strategii innej niż wylosowana przez losującego.

Definicja 3.2. Skorelowana Równowaga. *[5] Dana jest gra w postaci normalnej dla dwóch graczy: kolumnowego i wierszowego* $\mathcal{N} = \{r,c\}$. *Niech* $\Pi = \Pi_r \times \Pi_c$ *będzie zbiorem wszystkich profili strategii prostych w tej grze. Oznaczmy przez* $p : \Pi \to [0,1]$ *punkcję opisującą rozkład prawdopodobieństwa nad przestrzenią* Π*, czyli taka, że* $\sum_{\pi\in\Pi} p(\pi) = 1$. *Zdefiniujmy funkcje pomocnicze opisujące strategie brzegowe każdego z graczy* $i \in \mathcal{N}$ *jako* $p_i : \pi_i \mapsto \sum_{\pi_{-i}\in\Pi_{-i}} p(\pi_i, \pi_{-i})$. *Ponadto niech oznaczmy zbiór wszystkich funkcji definiujących rozkład prawdopodobieństwa strategii prostych (strategie mieszaną) gracza* i *przez* $Q_i = \{q_i : \Pi_i \to [0,1] | \sum_{\pi_i\in\Pi_i} q_i(\pi_i) = 1\}$

Funkcję p *będziemy nazywać Skorelowanym Stanem Równowagi, jeśli dla każdego z graczy* $i \in \mathcal{N}$ *spełniona jest następująca własność:*

$$\forall q_i \in Q_i \quad \sum_{\pi_i\in\Pi_i} \sum_{\pi_{-i}\in\Pi_{-i}} p_i(\pi_i)p_{-i}(\pi_{-i})u_i(\pi_i,\pi_{-i}) \geqslant \sum_{\pi_i\in\Pi_i} \sum_{\pi_{-i}\in\Pi_{-i}} q_i(\pi_i)p_{-i}(\pi_{-i})u_i(\pi_i,\pi_{-i}) \tag{3.72}$$

Podstawowa definicja skorelowanej równowagi nie jest związana z grami Stackelberga, w szczególności jest w pełni symetryczna. SEFCE jest modyfikacją tego pojęcia dla gry w postaci ekstensywnej, gdzie w każdym zbiorze informacyjnym gracz może otrzymać nową rekomendację, a następnie dostosowanie koncepcji do gier Stackelberga poprzez ustalenie, że to

lider jest stroną decydującą o rozkładzie prawdopodobieństwa ruchów. W przeciwieństwie to Skorelowanej Równowagi, losowanie nie jest wykonywane jednostopniowo, a kolejne ruchy są losowane przez lidera wraz z osiąganiem kolejnych stanów. To oznacza, że naśladowca może otrzymać rekomendację będącą rozkładem prawdopodobieństwa, ponieważ lider nie wylosował jeszcze konkretnego ruchu w stanie poniżej bieżącego węzła gry.

Definicja 3.3. SEFCE. *[18] Rozkład prawdopodobieństwa ϕ nad profilami strategii prostych* Π *będziemy nazywać Skorelowanym Stanem Równowagi Stackelberga w Grach w Postaci Ekstensywnej (SEFCE), jeśli spełnione są oba następujące warunki:*

- *W każdym stanie s będącym punktem decyzyjnym naśladowcy, naśladowca na podstawie bieżącego zbioru informacyjnego I_f, w którym się znajduje oraz rekomendacji od lidera, która jest warunkowym prawdopodobieństwem zagrania poszczególnych ruchów ze stanu I_f pod warunkiem, że zostały zagrane ruchy **lidera** prowadzące do stanu s nie jest w stanie uzyskać wyższej wypłaty niż grając zgodnie z otrzymaną rekomendacją.*

- *wartość oczekiwana wypłaty naśladowcy gdy grane są ruchy z rozkładu ϕ jest maksymalna spośród wszystkich rozkładów spełniających pierwszy warunek. Remisy rozstrzygane są na korzyść lidera.*

Podobnie jak w przypadku Równowagi Nasha i Skorelowanej Równowagi, również tu wypłata lidera w stanie Równowagi Stackelberga jest nie większa niż wypłata lidera w SEFCE [87, Twierdzenie 2.].

Ponadto autorzy wspomnianej pracy pokazują, że Równowagę Stackelberga można zdefiniować poprzez rozszerzenie punktu pierwszego definicji SEFCE o wymaganie, że rekomendacje dla naśladowcy muszą być strategiami prostymi i muszą być niezależne or ruchów zagranych przez lidera [87, Twierdzenie 1.]. Jeśli rekomendacje dla naśladowcy mają takie własności będziemy je określać jako spójne. W przeciwnym wypadku będziemy mówili o rekomendacjach niespójnych. Właśnie to twierdzenie jest podstawą zasady działania metody *C2016*. W pierwszym kroku dla danej gry znajdowane jest SEFCE. Następnie rozwiązanie przeglądane jest pod kątem rekomendacji dla naśladowcy, które nie są strategiami prostymi lub nie są niezależne od wyborów lidera. Następnie dodawane są ograniczenia do programu linowego w miejscach, gdzie były podane niespójne rekomendacje i program rozwiązywany jest ponownie. Krok sprawdzenia i dodania ograniczeń powtarzany jest tak długo, aż wszystkie rekomendacje dla naśladowcy spełniają warunki Równowagi Stackelberga. Autorzy metody proponują trzy techniki wprowadzania nowych ograniczeń, które omówimy dalej. Główna pętla metody jest podsumowana w Algorytmie 3.5. Zmienna M przechowuje kolejkę par zestaw ograniczeń do dodania do programu liczącego SEFCE i ograniczenie górne na wypłatę lidera w stanie Równowagi Stackelberga (o ile zastosowanie tych ograniczeń prowadzi do uzyskania w kolejnych krokach Równowagi Stackelberga, bo może prowadzić do programu sprzecznego). Zmienna LB to wypłata lidera dla najlepszego znanego dotychczas kandydata na Równowagę Stackelberga — takiego

Algorytm 3.5: Główna pętla metody *C2016* [87, Algorytm 1.]

```
1  M ← {(∞, ∅)};
2  LB ← −∞;
3  s ← nil;
4  while M ≠ ∅ ∧ MaxFirst(M) > LB do
5  |  (UB, r) ← MaxByFirst(M);
6  |  M ← M\(UB, r);
7  |  P ← program (3.73)–(3.79) z dodanymi ograniczeniami r;
8  |  Solve(P);
9  |  if IsFesible(P) then
10 |  |  (U, s′) ← Solution(P);
11 |  |  I′ ← InconsistientRecommendations(s′);
12 |  |  if I′ ≠ ∅ then
13 |  |  |  if U > LB then
14 |  |  |  |  LB ← U;
15 |  |  |  |  s ← s′
16 |  |  |  end
17 |  |  else
18 |  |  |  M ← M ∪ RestrictionsToTry(I′)// Jeden z wariantów SI-LP,
   |  |  |     SI-MILP, AI-MILP
19 |  |  end
20 |  end
21 end
22 return s;
```

rozkładu prawdopodobieństwa w którym rekomendacje dla naśladowcy będą spełniały warunki Równowagi Stackelberga (natomiast strategia lidera nie musi być strategią optymalną). Początkowo nie ma żadnego kandydata, więc przyjmujemy $-\infty$. Główna pętla wykonywana tak długo, jak w kolejce M jest zestaw ograniczeń, który może poprawić LB. W każdym przebiegu wybierany jest jeden z potencjalnie poprawiających LB zestawów ograniczeń, stosujemy heurystykę wybierającą wariant o najwyższym ograniczeniu górnym. Dla wybranego zestawu ograniczeń rozwiązywany jest program liniowy wyliczający SEFCE z dodanymi ograniczeniami wyjętymi z kolejki. Jeśli we wszystkich punktach decyzyjnych naśladowcy rekomendacja jest spójna, to znalezione rozwiązanie jest kandydatem na rozwiązane — sprawdzane jest czy poprawia wypłatę naśladowcy i jeśli tak, jest to nowy kandydat na najlepsze rozwiązanie. Jeśli istnieją rekomendacje niespójne, to stosując metody opisane dalej do kolejki M dodawane są nowe zestawy ograniczeń. Dodawane ograniczenia są rozszerzeniem ograniczeń wyjętych w tej iteracji z kolejki o ograniczenia, które mają zapewnić, że w zbiorach informacyjnych, w których bieżące rozwiązanie daje niespójne rekomendacje naśladowcy, rekomendacje będą spójne (może to jednak spowodować powstanie niespójnych rekomendacji w kolejnych zbiorach informacyjnych, co zostanie rozwiązane poprzez dodanie kolejnych ograniczeń w dalszych iteracjach). Może się również zdarzyć, że niektóre z proponowanych ograniczeń doprowadzą do tego, że program będzie sprzeczny. Wtedy algorytm przechodzi do kolejnego zestawu ograniczeń.

Program liniowy wyliczający SEFCE. W metodzie BC2015 program liniowy wykorzystywał plany realizacji (prawdopodobieństwa sekwencji strategii) zamiast definiowania prawdopodobieństwa dla każdej ze strategii prostych gracza. W programie wyliczającym SEFCE zaproponowanym w pracy [18] zastosowana jest podobna technika, ale do rozkładu prawdopodobieństw profili strategii. Zastosowane podejście jest nazywane planem korelacji (ang. correlation plan). Zamiast definiować prawdopodobieństwa każdej parze strategi prostych lidera i naśladowcy, definiujemy prawdopodobieństwo że zostaną zagrane pary sekwencji ruchów. W grach z doskonałą pamięcią oba sposoby definiowania rozkładu profili strategii są równoważne, co jest wykazane w pracy [18]. Ograniczenia na wartości prawdopodobieństw poszczególnych par sekwencji są uogólnieniem warunków dla planu realizacji dla strategii behawioralnej. W programie liniowym będziemy mieli rodzinę zmiennych $p(\sigma_l, \sigma_f) \in [0, 1] \quad \forall \sigma_l \in \Sigma_l, \sigma_f \in \Sigma_f$, które będą opisywać prawdopodobieństwa w planie korelacji, a maksymalizowana będzie wartość oczekiwana wypłaty lidera:

$$\max \sum_{\sigma_l \in \Sigma_l} \sum_{\sigma_f \in \Sigma_f} p(\sigma_l, \sigma_f) u_l(\sigma_l, \sigma_f). \tag{3.73}$$

Zmienne p będą ograniczone tak, aby prawdopodobieństwa sekwencji wydłużonych o jeden ruch sumowały się do prawdopodobieństwa sekwencji niewydłużonej, oddzielnie w wymiarze σ_l i oddzielnie w wymiarze σ_f. Prawdopodobieństwo zagrania pary sekwencji pustych będzie

analogicznie jak w przypadku planów realizacji równe 1:

$$p(\varnothing, \varnothing) = 1 \tag{3.74}$$

$$p(\sigma_l(I), \sigma_f) = \sum_{a \in A(I)} p(\sigma_l(I), \sigma_f) \qquad \forall I \in \mathcal{I}_l \quad \forall \sigma_f \in \Sigma_f \tag{3.75}$$

$$p(\sigma_l, \sigma_f(I)) = \sum_{a \in A(I)} p(\sigma_l, \sigma_f(I)) \qquad \forall I \in \mathcal{I}_f \quad \forall \sigma_l \in \Sigma_l \tag{3.76}$$

Następnie definiowane są ograniczenia wynikające z definicji SEFCE — zapewnienie, że w każdym zbiorze informacyjnym rekomendacja dla naśladowcy jest optymalna — naśladowcy nie opłaca się grać inaczej niż według rekomendacji. W tym celu w programie linowym pojawią się dwie rodziny zmiennych v. Koncepcyjne te zmienne są zbliżone do zmiennych v z programu *BC2015*, gdzie konieczna była gwarancja optymalnej odpowiedzi naśladowcy. Pierwsza rodzina zmiennych to $v(\sigma_f) \in \mathbb{R} \quad \forall \sigma_f ß \Sigma_f$ to wypłata naśladowcy przemnożona przez prawdopodobieństwo sekwencji σ_f, jeśli naśladowca będzie się przez cały czas kierował otrzymywanymi rekomendacjami. Konstrukcja ograniczenia, która gwarantuje właśnie takie wartości zmiennych $v(\sigma_f)$ jest analogiczna to ograniczenia (3.71) w programie *BC2015*, gdy zmienna slack jest wyzerowana:

$$v(\sigma_f) = \sum_{\sigma_l \in \Sigma_l} p(\sigma_l, \sigma_f) u_f(\sigma_l, \sigma_f) + \sum_{I \in \mathcal{I}_f | \sigma_f(I) = \sigma_f} \sum_{a \in A(I)} v(\sigma_f a) \qquad \forall \sigma_f \in \Sigma_f. \tag{3.77}$$

Druga rodzina zmiennych, $v(I, \sigma_f) \quad \forall I \in \mathcal{I}_f, \sigma_f \in Sigma_f$, jest większa lub równa wypłacie atakującego, gdy naśladowca zagra najlepszą możliwą dla siebie akcję w zbiorze informacyjnym I przemnożoną przez prawdopodobieństwo zagrania sekwencji σ_f grając według rekomendacji.

$$v(I, \sigma_f) \geqslant \sum_{\sigma_l \in \Sigma_l} p(\sigma_l, \sigma_f) u_f(\sigma_l, \sigma_f(I)a) + \sum_{I' \in \mathcal{I}_f | \sigma_f(I') = \sigma_f(I)a} v(I', \sigma_f) \quad \forall I \in \mathcal{I}_f, \forall \sigma_f \Sigma_f, \forall a \in A(I) \tag{3.78}$$

W momencie, gdy plan korelacji spełnia warunki definiujące SEFCE, zachodzi

$$v(I, \sigma_f(I)a) = v(\sigma_f(I)a) \qquad \forall I \in \mathcal{I}_f, \sigma_f \in \Sigma_f, \forall a \in A(I). \tag{3.79}$$

Program liniowy opisany przez wzory (3.73)–(3.79) pozwala znaleźć SEFCE. Warto w tym momencie zaobserwować że ten program liniowy nie zawiera żadnych zmiennych binarnych. Dodatkowo, autorzy metody proponują jeszcze uproszczenie tego programu, polegające na obserwacji, że część par sekwencji lidera i naśladowcy nie prowadzi do żadnego węzła w drzewie, bo zagrania jednego z graczy wykluczają wystąpienie oczekiwanych zbiorów informacyjnych. Zmienne $p(\sigma_l, \sigma_f)$ odpowiadające takim parom sekwencji będą zawsze wyzerowane. Można więc pominąć te zmienne i wszystkie ograniczenia, które odnoszą się tylko do tego zbioru zmiennych przy generowaniu programu liniowego, zmniejszając jego rozmiar.

Aby w algorytmie generującym program liniowy wskazać, które z par sekwencji (σ_l, σ_f) należy uwzględnić, a które można bez straty dla wyniku pominąć, autorzy definiują pomocnicza relację Par istotnych sekwencji (ang. Relevant Sequence Pairs), będącą adaptacją relacji zaproponowanej w pracy [83]. W programie liniowym zostaną uwzględnione tylko te zmienne $p(\sigma_l, \sigma_f)$, dla których odpowiadające im sekwencje są w relacji.

Definicja 3.4. Pary istotnych sekwencji. *[87] Parę $(\sigma_l, \sigma_f) \in \Sigma_l \times \Sigma_f$ będziemy nazywać istotną wtedy i tylko wtedy, gdy:*

- *$\sigma_f = \varnothing$ lub*
- *istnieje para węzłów w drzewie gry $s, s' \in S$ że s' jest poprzednikiem s lub $s' = s$ oraz sekwencja ruchów lidera prowadząca do s to σ_l ($\sigma_l(s) = \sigma_l$) i a s' należy do zbioru informacyjnego z którego został wykonany ostatni ruch σ_f ($s' \in I_f(\sigma_f)$).*

Przedstawiony przed chwilą program liniowy znajduje SEFCE, ale nie gwarantuje znalezienia Równowagi Stackelberga. Problemem jest fakt, że w SEFCE naśladowca może w swoim zbiorze informacyjnym otrzymać różne rekomendacje w zależności od zbioru informacyjnego widzianego przez lidera, podczas gdy w stanie Równowagi Stackelberga decyzja naśladowcy musi być zależna tylko od zbioru informacyjnego widzianego przez naśladowcę. W linii 11 Algorytmu 3.5 wyszukiwane są właśnie te przypadki. Następnie stosowane jest jedna procedura `RestrictionsToTry` jedna z trzech metod uzupełniających program liniowy o dodatkowe ograniczenia w celu wyeliminowania niespójnych rekomendacji. Nazwy procedur podane są zgodnie z oryginalną pracą:

- **SI-LP** Spośród wszystkich zbiorów informacyjnych z niespójnymi rekomendacjami wybierany jest ten, który jest najbliżej korzenia drzewa gry. Jeśli jest ich kilka równie blisko, arbitralnie wybierany jest jeden, ozn. I. Następnie dla każdego ruchu a dla którego naśladowca mógł otrzymać rekomendację z niezerowym prawdopodobieństwem, do kolejki M ograniczeń do rozważenia dodawane są aktualne ograniczenia rozszerzone o ograniczenie wymuszające prawdopodobieństwo 1 dla wszystkich rekomendacji ruchu a z tego zbioru informacyjnego (czyli ograniczenie $p(\sigma_l, \sigma_f(I)a) = 0$ dla wszystkich σ_l).

  Takie postępowanie koncepcyjnie przypomina podejście Mutilple-LP w poszukiwaniu Równowagi Stackelberga w grach w postaci normalnej, gdzie y kolejno wybrać odpowiedź naśladowcy i dobrać do niej strategię lidera. Heurystyka wyboru węzła znajdującego się najwyżej w drzewie wynika z faktu, że krok ustawienia na 1 rekomendacji naśladowcy dla ruchu a zeruje zmienne $p(\sigma_l, \sigma_f)$ w poddrzewach, do których prowadzą inne ruchy niż a, a co za tym idzie może wyzerować pewne zbiory informacyjne, gdzie wystąpiły niespójne rekomendacje.

- **SI-MILP** W poprzednim wariancie każdy krok powodował dodanie do kolejki M tylu wariantów ograniczeń ile ruchów potencjalnie mogło być rekomendacją odpowiadającą

strategii w stanie Równowagi Stackelberga. Drugi wariant różni się tym, czym DOBSS od podejścia Multiple-LP — zastępuje wiele wariantów programu jednym, w którym występują zmienne binarne.

Zbiór informacyjny I jest wybierany tak, jak w wariancie $SI - MILP$. Następnie zestaw aktualnie rozważanych ograniczeń rozszerzany jest w pierwszej kolejności o rodzinę zmiennych binarnych $q_a \in \{0, 1\}$ dla każdego ruchu a, dla którego naśladowca mógł otrzymać rekomendację z niezerowym prawdopodobieństwem. Do zbioru ograniczeń dodawane jest ograniczenie wybierające dokładnie jeden z możliwych ruchów – $\sum_{a \in A} q_a = 1$, gdzie A jest zbiorem wszystkich potencjalnych a opisanych w poprzednim zdaniu. Na koniec dodawane są ograniczenia wymuszające zaproponowanie rekomendacji a dla której q_a jest niezerowe: $p(\sigma_l, \sigma_f(I)a) = q_a$, dla wszystkich a i wszystkich σ_l.

Podobnie jak przy przejściu z Multiple-LP do DOBSS przeszukiwanie przestrzeni metodą podziału i ograniczeń jest teraz realizowane przez solwer, a nie przez pętlę Algorytmu 3.5, ponieważ kolejka M zawiera zawsze nie więcej niż jeden zestaw ograniczeń.

- **AI-MILP** Ten wariant to odejście od heurystyki wyboru zbioru informacyjnego możliwie blisko korzenia na rzecz wykorzystania heurystyk solwera MILP. Postępowanie jest podobne jak w SI-MILP, jednak zamiast wybierać jeden zbiór I, zestaw ograniczeń dodawany jest dla każdego zbioru, gdzie występuje niespójna rekomendacja na raz. To minimalizuje liczbę pętli Algorytmu 3.5 na rzecz większych MILP do rozwiązania.

Następnie, w pracy [87] autorzy porównują eksperymentalnie poszczególne warianty metod. Wyniki pokazują, że w zależności od konkretnej wielkości i struktury gry inny wariant metody może okazać się najskuteczniejszy. W grach o mniej skomplikowanej strukturze zbiorów informacyjnych najlepszy wydaje się być wariant SI-LP, podczas gdy AI-MILP działa szybciej dla pozostałych instancji testowych.

3.7.3 Metoda opata o rozwiązywanie uproszczonych wersji gry

Dwie zaprezentowane wcześniej metody do gier wielokrokowych charakteryzowały się poszukiwaniem profilu strategii będącego dokładnie Równowagą Stackelberga. Jednak, ze względu na fakt, że w ogólnym przypadku poszukiwanie tej równowagi jest problemem NP-trudnym, to, o ile tylko $P \neq NP$, metody dokładne nie będą w stanie osiągnąć zadowalającej wydajności dla dużych gier. Metoda opublikowana w 2018 roku w pracy [95] jest rozszerzeniem poprzednich metod o mechanizm konstruowania mniejszych gier z gry wejściowej i poszukiwania Równowagi Stackelberga w tych uproszczonych grach stosując metodę zaprezentowaną w poprzednim podrozdziale. W anglojęzycznej literaturze na takie podejście stosuje się nazwę abstrahowanie gry (game abstraction).

Główna idea metody polega na zwijaniu wielostopniowego poddrzewa gry do drzewa jednopoziomowego z ruchami odpowiadającymi wszystkim możliwym sekwencjom ruchów lidera.

Wypłaty lidera w tym drzewie są górnymi oszacowaniami wypłaty lidera. Tak zwinięte poddrzewo, odpowiadające ruchowi i jego kontynuacjom, autorzy pracy nazywają gadżetem (ang. gadget). Dla drzewa gry ze zwiniętymi fragmentami wyliczana jest Równowaga Stackelberga, a następnie rozwijane są tylko te gadżety, które weszły z niezerowym prawdopodobieństwem do stanu równowagi. W ten sposób część drzewa gry może nie być rozważana, jeśli potencjalny wynik lidera w danym poddrzewie nie ma szans dać odpowiednio wysokiej wypłaty.

Algorytm 3.6: Metoda oparta o rozwiązywanie uproszczonych gier (Algorytm 2 w [95])

```
   input: G — gra
 1 (M, Σ') ← expand(root(G), ∅)// Dostępne ruchy i stan (węzeł
       drzewa) w korzeniu gry.
 2 expansionNeeded ← size(M) > 0;
 3 while expansionNeeded do
 4 |   for stateh ∈ M do
 5 |   |   Σ' ← Σ' ∪ createGadget(h)// Uproszczenie notacji: ta suma
 |   |   |      oznacza też usunięcie liści, które zostały
 |   |   |      zastąpione węzłami z potomkami
 6 |   end
 7 |   p ← solve(σ')// Wyznaczenie strategii w stanie Równowagi
 |   |     Stackelberga
 8 |   M ← ∅;
 9 |   E ← getGadgetsToExpand(p);
10 |   for stateh ∈ M do
11 |   |   (M', Σ') ← expand(h, Σ');
12 |   |   M ← M ∪ M';
13 |   end
14 |   expansionNeeded ← size(M) > 0;
15 end
```

Ramowy pseudokod tego podejścia przedstawiony jest w Algorytmie 3.6. Na początku tworzona jest gra składające się z jednego gadżetu – stanu-korzenia, wszystkich dopuszczalnych ruchów w tym stanie i, bezpośrednio po tych ruchach, liści będących terminalami z górnym oszacowaniem wypłaty lidera po danych ruchu. Następnie, w linii 7, uproszczona gra jest rozwiązywana. Na podstawie rozwiązania wybierane są te stany, których przeszacowanie, (jako, że wypłaty w gadżetach, to oszacowania górne), może mieć największy wpływ na wynik. Wybrane stany są rozbudowywane do gadżetów i uruchamiany jest kolejny przebieg pętli rozwiązującej. Całość kończy się, gdy nie ma już stanów ro rozbudowania. W dalszej części tym podrozdziale zostanie zaprezentowanych sposób budowania gadżetu i wyboru węzłów do rozbudowy.

Budowa gadżetu. Budowa gadżetu zastępującego poddrzewo pod pewnym węzłem decyzyjnym h lidera odbywa się w następujący sposób:

1. Przejdź przez wszystkie liście poddrzewa zakorzenionego w h, dla każdego liścia z_i za-

pamiętaj parę wypłat graczy (u_l^i, u_f^i).

2. Znalezione pary są wszystkimi możliwymi wynikami gry dla graczy, można je zobrazować na płaszczyźnie, gdzie współrzędna y odpowiada wypłacie lidera, a współrzędna x wypłacie naśladowcy (takie mapowanie wypłat na współrzędne ma znaczenie przy konstruowani otoczki wypukłej). Ze względu na to, że naśladowca maksymalizuje swój wynik, aby otrzymać górne oszacowanie, możemy rozważać jedynie górną otoczkę wypukłą tak otrzymanych punków. Przez otoczkę rozumiemy tu łamaną, w której końce odcinków, to znalezione punkty i wszystkie punkty znajdują się pod (w sensie wartości y) łamaną. Wizualizację i algorytm wyliczający łamaną można znaleźć w na Rysunku 4 w przytoczonej pracy.

3. Zbuduj poddrzewo złożone z węzła od którego bezpośrednio odchodzą liście odpowiadające wierzchołkom łamanej. Do każdego z liści prowadzi inna akcja lidera.

Uwaga: wybrany węzeł h może być częścią większego zbioru informacyjnego. W takiej sytuacji w zredukowanej grze nowy węzeł nie będzie częścią tego zbioru, to może prowadzić do chwilowego przeszacowania wyniku.

Wybór węzłów do rozbudowy. Jeśli w strategii będącej Równowagą Stackelberga występują jakiekolwiek ruchy z węzłów będących gadżetami lub jeśli w drzewie gry występują (niekoniecznie w strategii) węzły ze zbiorów informacyjnych, z których pochodzą węzły, z których zbudowano gadżety, do wyznaczenia Równowagi Stackelberga w kolejnej grze, konieczne jest rozważenie mniej zredukowanej gry. Dzieje się to poprzez ekspansję gadżetów z powrotem do oryginalnych poddrzew, (linia 9 i kolejne w Algorytmie 3.6). Jako gadżety do ekspansji wybierane są wszystkie węzły w drzewie, które spełniają warunki z początku akapitu i żaden z poprzedników węzła nie spełnia wspomnianych warunków. Formalną definicję gadżetów do wyboru można znaleźć we wzorach (17) i (18) w przytoczonej pracy.

Poprawa wydajności poprzez dodanie heurystyk. Jak podkreślają autorzy metody, zastosowanie wprost proponowanego podejścia bardzo często będzie zbyt wolne i zbyt kosztowne pamięciowo, ze względu na rozwijanie bardzo wielu gadżetów mimo potencjalnie niewielkiego zysku i niejednokrotnie bardzo dużej liczbie punktów we wspomnianej otoczce wypukłej. W związku z tym autorzy wprowadzili następujące usprawnienia heurystycznie.

- *Otoczka wypukła* nie musi pokrywać wszystkich punktów, mogą być punktów, które są ponad otoczką, jednak nie wyżej niż o zadany współczynnik δ.
- Aby złagodzić skutki przeszacowania wypłat spowodowane przez poprzedni punkt, dodana jest niewielka kara dla lidera, która odejmowana jest od wypłat w gadżetach.
- Rezygnacja z warunku, żeby wszystkie gadżety, których węzły pochodzą ze zbiorów informacyjnych niebędących singletonami zostały rozwinięte.

Szczegóły tych heurystycznych modyfikacji, które znacząco przyspieszają algorytm są opisane w przytaczanej pracy.

3.8 Podsumowanie technik stosowanych w literaturze

W istniejących publikacjach na temat obliczania Równowagi Stackelberga techniki, które pozwalają zmniejszyć czas i pamięć niezbędne do uzyskania wyniku są jednym z podstawowych problemów. Nie jest to zaskoczeniem, biorąc pod uwagę fakt, że w ogólnym przypadku problem znalezienia Równowagi jest NP-trudny. Techniki zaproponowane w literaturze możemy podzielić na dwie kategorie: związane ze szczególną postacią gry dla zastosowania i ogólne. Z punktu widzenia rozprawy bardziej istotne są techniki mające zastosowanie w szerokich klasach gier, niemniej jednak poniżej wymienimy propozycje z obu kategorii.

Techniki ogólne:

TO1 Generowanie ograniczeń — o ile efektywne generowanie ograniczeń zwykle wymaga dziedzinowych usprawnień, sama idea tego podejścia jest ogólna i w przypadku poszukiwania Równowagi Stackelberga oznacza selektywne przeglądanie możliwych strategii naśladowcy.

TO2 Generowanie kolumn — technika ogólna w tym samym sensie, co poprzednia, tym razem wiąże się z selektywnym przeglądaniem strategii lidera.

TO3 Rozważanie zredukowanych gier — technika, którą ideowo można powiązać z generowaniem kolumn i ograniczeń, ale ściśle związana z postacią drzewa gry, a nie programu liniowego.

TO4 Metoda podwójnej wyroczni.

TO5 Metoda oparta o SEFCE — stosowalna w ogólnej klasie gier, ale jej skuteczność jest ściśle związana z postacią programu liniowego i trudno jest przenieść te ideę na metody rozwiązywania nie oparte o programy liniowe.

TO6 Metoda podziału i ograniczeń — jest szczególnie efektywna w przypadku mieszanych programów liniowych, gdzie występują zmienne całkowite, ale nie ma oczywistego przeniesienia na inne techniki.

Techniki dziedzinowe:

- Stosowanie strategii brzegowych

Celem badań prezentowanych w tej rozprawie było stworzenie rozwiązań opartych o metody inteligencji obliczeniowej, która będzie mogła być stosowana w ogólnej klasie gier. Efektem przeglądu literatury było wskazanie czterech technik oznaczonych numerami TO1–TO4

jako takich, które mogą być inspiracją dla budowy ogólnych metod nieopartych o programowanie liniowe. Skonstruowane na tej badzie metody są głównym efektem prezentowanych badań i są szczegółowo opisane w kolejnym rozdziale rozprawy.

Warto tez podkreślić, że w trakcie badań zdecydowaliśmy się nie korzystać z metod dziedzinowych takich, jak strategie brzegowe. Powodem tej decyzji był fakt, ze takie metody trudno jest dostosować do szerokiej klasy gier. Nawet w przypadku względnie prostej zmiany gry SMOS z sumy zerowej na sumę ogólną (opisanej w Rozdziale 3.5), wymagane było gruntowne przebudowanie programu liniowego.

3.9 Istniejące klasy gier testowych

Rozwijanie metod obliczeniowych do rozwiązywania gier, szczególnie metod heurystycznych, wymaga również budowy zbiorów testowych, które posłużą do testowania skuteczności i szybkości tych metod i pozwolą na porównanie metod obliczeniowych między sobą. W tym podrozodziale zostaną zaprezentowane zbiory gier testowych z literatury, które są sekwencyjnymi grami Stackelberga.

3.9.1 Search Games

Gry on nazwie Search Games (w wolnym tłumaczeniu gry z poszukiwaniem [naśladowcy]) zostały zaproponowane w pracy [16] i początkowo zostały użyte do ewaluacji metody wyliczania Równowagi Nasha w grach z niepełną informacją. W późniejszych pracach tych samych autorów [17, 87] te gry są również wykorzystane do badania szybkości działania metod obliczających Równowagę Stackelberga. Metody opisane we wspomnianych pracach zostały omówione wcześniej w rozdziałach 3.7.1 i 3.7.2.

Gra jest zbudowana na bazie skierowanego grafu o specyficznej strukturze. Dany jest graf skierowany. Gra polega na przemieszczaniu przez graczy swoich jednostek (pionków) pomiędzy wierzchołkami grafu. Gra podzielona jest na kroki czasowe, w każdym kroku czasowym gracze wykonują ruch jednocześnie. W każdym kroku czasowym gracz może pozostawić każdą ze swoich jednostek w tym samym punkcie lub przesunąć ją na sąsiedni wierzchołek grafu (uwzględniając skierowanie krawędzi). Dodatkowo, w przypadku lidera, jednostki mogą się poruszać tylko w obrębie ustalonego podzbioru wierzchołków grafu. W grafie są wyróżnione 4 wierzchołki: jeden wierzchołek startowy naśladowcy i 3 wierzchołki docelowe. Gra kończy się w jednym z trzech przypadków:

1. gdy naśladowca zostanie *złapany* (procedura łapania opisana jest poniżej),
2. gdy naśladowca dojdzie do jednego z wierzchołków oznaczonych jako cel,
3. gdy zostanie osiągnięty z góry założony limit kroków czasowych.

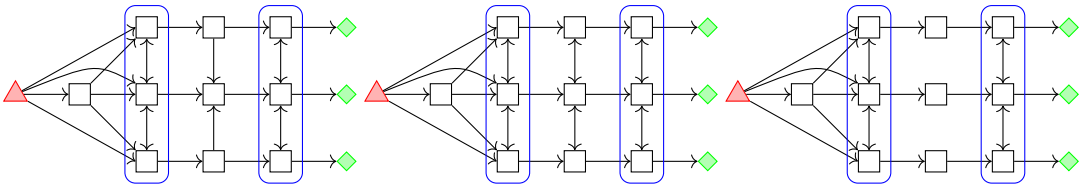
Każdej z powyższych sytuacji towarzyszą inne wartości wypłat, które otrzymują gracze na koniec gry. W pierwszym przypadku, gdy naśladowca zostaje złapany, lider otrzymuje wypłatę 1, a naśladowca -2, w przypadku dojścia naśladowcy do celu lider otrzymuje wypłatę -1, a naśladowca otrzymuje wypłatę dodatnią, która jest specyficzna dla każdego z celów i może być inna w każdej instancji gry. W przypadku osiągnięcia limitu kroków czasowych obaj gracze otrzymują wypłatę 0.

Złapanie naśladowcy. Złapanie naśladowcy następuje w jednym z dwóch przypadków: gdy po wykonaniu ruchów zachodzi sytuacja, że jednostka lidera i jednostka naśladowcy znajdują się w tym samym wierzchołku grafu lub w przypadku, gdy istnieją dwa wierzchołki grafu a,b takie, że jednostka lidera znajduje się w wierzchołku a i lider zdecydował się przesunąć tę jednostkę do b, w b znajduje się jednostka naśladowcy i naśladowca zdecydował się ją przesunąć do a (gracze spotkali się na krawędzi w trakcie przemieszczania się).

Informacja w grze. Gra charakteryzuje się niepełną informacją. Gracze znają swoje początkowe pozycje, ale nie znają swoich pozycji w grafie. Ponadto naśladowca zostawia ślady. Zostawienie śladów polega na tym, że lider, gdy wejdzie na wierzchołek, w którym był wcześniej naśladowca otrzymuje informację o tym, że wierzchołek był odwiedzony przez naśladowcę (tylko fakt odwiedzenia, bez informacji o czasie). Ponadto gra może być rozgrywana w dwóch wariantach:

- naśladowca, gdy zdecyduje się nie ruszać do sąsiedniego wierzchołka i pozostaje w bieżącym na kolejny krok czasowy, zaciera ślady i lider nie otrzyma informacji o śladach w momencie wejścia do wierzchołka,
- naśladowca nie może pozostać w wierzchołku i nie ma możliwości zacierania śladów.

Struktura grafu gry. Grafy wykorzystywane w Seach Games mają bardzo specyficzną strukturę, która pozwala wykorzystać możliwość obserwacji śladów przez lidera. Struktura grafu na którym toczy się rozgrywka występuje w trzech wariantach, wszystkie są zaprezentowane na Rysunku 3.2. Grafy te składają się z trzech równoległych jednokierunkowych ścieżek o równej długości, na rysunku ułożonych poziomo, z lewej do prawej. Na lewo od ścieżek znajdują się dwa wierzchołki. Wierzchołek startowy naśladowcy i wierzchołek pośredni. Wierzchołek startowy naśladowcy jest połączony w wierzchołkiem pośrednim i wszystkimi wierzchołkami startowymi ścieżek (krawędzie tylko w jedną stronę) Wierzchołek pośredni jest połączony, również jednokierunkowo ze wszystkimi wierzchołkami początkowymi ścieżek. Każda ze ścieżek składa się z czterech wierzchołków. Skrajny prawy, czwarty, wierzchołek każdej ścieżki jest wierzchołkiem docelowym naśladowcy. Odpowiadające sobie wierzchołki ścieżek tworzą zbiory wierzchołków, które ograniczają możliwości ruchu jednostek lidera: pierwsza jednostka lidera jest ograniczona do pierwszych wierzchołków każdej ze ścieżek. Druga jednostka lidera



Rysunek 3.2: Trzy warianty grafu Search Games. Czerwonym trójkątem oznaczony wierzchołek startowy naśladowcy. Zielone kwadraty to cele naśladowcy. Wierzchołki otoczone niebieską linią to ograniczenie ruchów poszczególnych jednostek lidera.

jest ograniczona do trzeciego wierzchołka każdej ze ścieżek. Do uproszczenia opisu grafu dalej będzie użyta następująca notacja (x,y) do identyfikowania wierzchołków, gdzie x będzie numerem ścieżki, a y numerem wierzchołka w ścieżce. Pary wierzchołków $(1,1)$,$(2,1)$; $(2,1)$,$(3,1)$; $(1,3)$,$(2,3)$; $(2,3)$,$(3,3)$ są połączone krawędziami w obie strony. Połączenia pomiędzy $(1,2)$, $(2,2)$ oraz $(3,2)$ zależą od wariantu grafu. Autorzy wspomnianej pracy zaproponowali trzy warianty:

1. połączenie jednokierunkowe $(1,2)$, $(2,2)$ oraz połączenie jednokierunkowe $(3,2)$, $(2,2)$,
2. połączenie dwukierunkowe $(1,2)$, $(2,2)$ oraz połączenie dwukierunkowe $(3,2)$, $(2,2)$,
3. brak połączeń w obrębie tych trzech wierzchołków.

3.9.2 Gry z przejmowaniem węzłów (Flip-it game)

Gra Flip-it nazywana też grą z sekretnym przejmowaniem (ang. game of stealthy takeover), została zaproponowana jako gra do badania różnorodnych zastosowań związanych z bezpieczeństwem sieci komputerowych i bezpieczeństwem informacji [29]. Gra jest rozgrywana na skierowanym grafie, a w rozgrywce bierze udział dwóch graczy: obrońca i atakujący. Z każdm z wierzchołków związane są dwie liczby dodatnie liczby rzeczywiste: wypłata za posiadanie wierzchołka oraz koszt przejęcia wierzchołka. Niepusty podzbiór wierzchołków definiuje wierzchołki wejściowe tego grafu. Gra rozgrywana jest przez t rund. Każdy wierzchołek w każdej rundzie należy do jednego z graczy, przed pierwszą rundą wszystkie wierzchołki są kontrolowane przez obrońcę. Gra nazywana jest flip-it w związku z koncepcją posiadania wierzchołków analogiczną do gier, gdzie przejmuje się kamienie i odwraca się je odpowiednio czarną lub białą stroną do góry.

Przebieg rundy. W każdej rundzie każdy z graczy wybiera jeden z wierzchołków, który chce przejąć. Jeśli spełnione są wszystkie następujące warunki:

- gracz kontroluje przynajmniej jeden wierzchołek, z którego jest krawędź do wierzchołka przejmowanego lub przejmowany wierzchołek jest wierzchołkiem wejściowym,
- przeciwnik nie próbuje przejąć tego samego wierzchołka,

- gracz nie kontroluje tego wierzchołka w tej chwili,

to gracz przejmuje wierzchołek w tej rundzie. Po każdej rundzie do wypłaty kazdego z graczy jest dodawana suma wypłat za posiadanie wszystkich wierzchołków, które gracz aktualnie kotroluje, a następnie odejmowany koszt przejęcia wierzchołka, który gracz próbuje przejąć w tej rundzie (niezależnie od tego, czy przejęcie zakończyło się powodzeniem). W trakcie gry gracze nie są informowani o tym, które z wierzchołków kontrolują w danej chwili.

Wymieniona gra, w dwóch wariantach, została użyta do ewaluacji metody opartej o upraszczanie drzewa gry [95] opisanej w Rozdziale 3.7.3. Atakujący pełnił role naśladowcy, a obrońca — lidera. Autorzy wspomnianej metody rozważyli dwa warianty gry: bez informacji (No-Info), gdzie gracze dowiadują się o sumie swoich wypłat dopiero na końcu gry oraz z informacja o wypłatach (All-Points), gdzie struktura zbiorów informacyjnych podaje graczom informację o sumie wypłat uzyskanych w danej rundzie.

W porównaniu z poprzednią grą testową, struktura drzewa tej gry jest bardziej skomplikowana: występuje dużo więcej zbiorów informacyjnych (szczególnie w wariancie All-Points), mimo, że liczba akcji w poszczególnych zbiorach informacyjnych jest względnie mała. Taka struktura pozwala testować to, jak dobrze metoda poszukiwania strategii radzi sobie z częściowo ujawnianą informacją.

Rozdział 4

Proponowane podejścia do aproksymacji Równowagi Stackelberga i sposoby ich ewaluacji

Ten rozdział zawiera treści kluczowe z punktu widzenia rozprawy doktorskiej. Zaprezentowano tu metody obliczeniowe będące oryginalnymi wynikami badań prowadzonych przez autora rozprawy pod kierunkiem promotora. W ramach prowadzonych prac powstały dwie różne metody obliczeniowe. Pierwsza z nich, nazwana *Mixed-UCT* jest prostsza, da się zastosować tylko do pewnego podzbioru gier wielokrokowych. Metoda ta powstała jako pierwsza, w ramach wstępnych badań, które mały na celu sprawdzić przydatność metody ukierunkowanego próbkowania drzew gry *Upper confidence bound applied to trees* (UCT) [50], która sprawdza się w poszukiwaniu optymalnych ruchów w grach planszowych, do poszukiwania strategii mieszanej lidera w stanie Równowagi Stackelberga. Druga z metod, nazwana *O2-UCT*, jest metodą zbudowaną na podstawie wniosków z analizy problemów i ograniczeń metody *Mixed-UCT*. W porównaniu z poprzedniczką, *O2-UCT* może być stosowana do całej klasy gier z doskonałą pamięcią. W porównaniu z poprzedniczką, ta metoda w mniejszym stopniu opiera się na próbkowaniu UCT i zbudowana jest z trzech oddzielnych modułów, co czyni ją bardziej skomplikowaną od *Mixed-UCT*.

Układ tego rozdziału jest następujący. W Rozdziale 4.1 zaprezentowane są rodziny gier użyte do ewaluacji proponowanych metod, ta sekcja zawiera przede wszystkim opis rodziny gier testowych powstałych w trakcie prac nad rozprawą. Ponadto przypomniane są gry testowe, które zostały omówione w poprzednim rozdziale i zostały wykorzystane do analizy skuteczności i szybkości proponowanych metod. Następnie, w Rozdziale 4.2 zaprezentowana jest pierwsza z metod, *Mixed-UCT*. Sekcja ta zawiera motywację i opis metody oraz wyniki eksperymentalne pokazujące jej przydatność. Następnie, w podobnym układzie, w Rozdziale 4.3, prezentowana jest metoda *O2-UCT*. Rozdział kończy się podrozdziałem 4.4, w której możliwości i ograniczenia obu metod są ze sobą zestawione i całość wyników jest podsumowana.

4.1 Wykorzystane rodziny gier testowych

Ocena skuteczności metod heurystycznych zwykle wymaga przygotowania zbiorów instancji testowych, na których różne metody zostaną uruchomione i porównane na podstawie otrzymanych wyników. W niektórych obszarach, na przykład w optymalizacji globalnej funkcji wielu zmiennych [57], istnieją popularne zbiory, na których testowane są metody z danego obszaru. W przypadku ogólnej klasy gier Stackelberga nie ma, jak dotąd żadnego zbioru tego typu. Przedstawiony wcześniej przegląd literatury pokazuje, że wiele metod w obszarze poszukiwania Równowagi Stackelberga skupia się na jednej podklasie gier. W związku z tym wiele metod zaproponowanych w literaturze nie jest ze sobą w ogóle porównywana, ze względu na to, że są przeznaczone dla rozłącznych podklas gier. Brak zbiorów testowych powoduje, że autorzy, którzy proponują metody, które działają dla szerokich klas gier, takie jak proponowane w tej rozprawie, musza proponować własne zbiory gier, które pozwolą ocenić skuteczność tych metod.

Do ewaluacji metod zaproponowanych w tej rozprawie posłużyły dwie rodziny gier:

- *Warehouse Games* — gry rozgrywane na grafie, w których naśladowca próbuje dotrzeć do jednego z kilku celów, a lider próbuje go przechwycić. Są to gry zaprojektowane przez autora rozprawy na potrzeby porównania proponowanych metod z innymi metodami występującymi w literaturze.

- *Search Games* — opisane w Rozdziale 3.9.1, zaproponowane i użyte przez autorów wspomnianych w poprzednim rozdziale metod działających w ogólnej klasie gier Stackelberga [17].

W dalszej części tego rozdziału zaprezentowany jest sposób konstrukcji rodziny gier *Warehouse Games*.

4.1.1 Warehouse Games (gry z obroną magazynu)

Wstępne wersje zbioru gier, budowanego of 2014 roku, który potem został nazwany *Warehouse Games* zostały opublikowane w pracach [40, 41, 42], zbiór w ostatecznej formie, prezentowanej tutaj został opublikowany w pracy [43] i wykorzystany do ewaluacji metody *Mixed-UCT*. Różnice pomiędzy wcześniejszymi wersjami, a wersją końcową dotyczą przede wszystkim metody losowania struktury grafu. W tej rozprawie jest zaprezentowana tylko ostateczna wersja. Zbiór ten został w niezmienionej postaci wykorzystany w późniejszych pracach opisujących metodę *O2UCT* prezentowaną dalej[45, 44].

Motywacja. Rozpoczęcie prac nad metodą do poszukiwania Równowagi Stackelberga w wielokrokowych grach o sumie niezerowej wymagało użycia zbioru testowego gier, który ma następujące własności:

- zawiera gry o zróżnicowanej liczbie kroków, dzięki czemu można analizować skalowalność metody dla gier o różnej długości,
- pozwala na elastyczny dobór wypłat w grze, tak żeby można było sprawdzać działanie dla gier bliższych i dalszych sumie zerowej,
- ekstensywna postać gry posada strukturę. Ten element jest istotny, ze względu na to, że część prac wykorzystuje całkowicie losowe drzewa gier do ewaluacji metod. Całkowicie losowe drzewa gier pozwalają ocenić skuteczność metod, ale scenariusz testowy, gdzie drzewo jest całkowicie losowe jest oderwany od rzeczywistości, gdzie gry wynikające z zastosowań często mają dość regularną strukturę.

Koncepcja. Gry *Warehouse Games* mają przypominać sytuację ochrony budynku przemysłowego, np. magazynu, w którym znajdują się cenne przedmioty. Gra rozgrywa się grafie, będącym prostokątną siatką wierzchołków, gdzie nie wszystkie sąsiadujące wierzchołki są ze sobą połączone. Kilka wierzchołków na grafie zawiera cenne przedmioty, które chce przejąć atakujący (naśladowca). Obrońca (lider) próbuje mu w tym przeszkodzić. Obaj gracze posiadają po jednej jednostce, która porusza się po grafie. W każdej kolejce gracz może przemieścić się do sąsiedniego wierzchołka grafu.

Definicja

Dany jest skierowany graf $G = (V,E)$, gdzie V zbiór wierzchołków grafu, a E zbiór krawędzi grafu oraz wyróżnione wierzchołki:

- $s_l \in V$ – wierzchołek startowy lidera,
- $s_f \in V$ – wierzchołek startowy naśladowcy,
- $Z \subseteq V \backslash \{s_f\}$ – wierzchołki zawierające cenne zasoby,

funkcje wypłat graczy:

- $U_f^+ : Z \to \mathbb{R}$ – wypłaty, zwykle dodatnie, naśladowcy za przejęcie cennego zasobu,
- $U_l^- : Z \to \mathbb{R}$ – wypłaty, zwykle ujemne, lidera za utratę cennego zasobu,
- $U_f^- : V \to \mathbb{R}$ – wypłaty, zwykle ujemne, naśladowcy w momencie złapania przez lidera,
- $U_l^+ : V \to \mathbb{R}$ – wypłaty, zwykle dodatnie, lidera, gdy złapie naśladowcę.

oraz $T \in \mathbb{N}$ – limit kroków czasowych gry.

Gra rozgrywana jest w dyskretnych krokach czasowych (rundach). Każdy z graczy dysponuje jedną jednostką, która w każdej rundzie przebywa w jakimś wierzchołku grafu. Przed

pierwszą rundą jednostka lidera znajduje się w wierzchołku s_l, a jednostka naśladowcy w wierzchołku s_f. Przed rozpoczęciem każdej rundy każdy z graczy wybiera nową pozycję dla swojej jednostki. Może to być albo pozycja bieżąca (gracz pozostaje w tym samym miejscu, w którym był) albo wierzchołek do którego istnieje skierowana krawędź z pozycji bieżącej (przejście do sąsiedniego wierzchołka). Po wyborze pozycji rozpoczyna się runda gry:

1. Jednostki przesuwane są do żądanych pozycji.
2. Sprawdzane są warunki końca gry (dokładnie w tej kolejności, pierwszy pasujący kończy grę z podanym wynikiem):
   (a) jeśli jednostka lidera i jednostka naśladowcy znajdują się w tym samym wierzchołku v, to gra się kończy, lider otrzymuje wypłatę $U_l^+(v)$, naśladowca otrzymuje wypłatę $U_f^-(v)$,
   (b) jeśli jednostka naśladowcy znajduje się w którymś wierzchołków ze zbioru Z, $z \in Z$, to gra się kończy, lider otrzymuje wypłatę $U_l^-(z)$, naśladowca otrzymuje wypłatę $U_f^+(z)$,
   (c) jeśli bieżąca runda ma numer T (został osiągnięty limit), to gra się kończy, lider i naśladowca otrzymują wypłatę równą 0.

Generowanie i warianty zbioru

Zaproponowana definicja gry daje dużą swobodę w budowaniu instancji takich gier, szczególnie dużą swobodę daje wybór grafu, na którym rozgrywa się gra. Aby stworzyć jednorodną rodzinę gier, oprócz samej definicji, powstał generator gier rozgrywanych n grafach o specyficznej strukturze. Generator, tak samo jak definicja gry, został opublikowany przez autora rozprawy i jego promotora w pracy [43]. Wszystkie wygenerowane zbiory gier są dostępne w opisanym dalej repozytorium z kodem źródłowym metod *Mixed-UCT* i *O2-UCT* dostępnym pod adresem `https://gitlab.com/jan-karwowski/petrus`.

Nazwa gier testowych – *Warehouse games* – wynika z wybranego sposobu generowania grafu dla rozgrywki. Graf ma za zadanie przypominać układ kompleksu przemysłowego, np. magazynu. Ogólna idea generowania grafu jest następująca: układamy wierzchołki grafu na płaszczyźnie w siatce prostokątnej, potencjalne połączenia będą możliwe tylko pomiędzy wierzchołkami sąsiadującymi w pionie lub w poziomie. Następnie wyznaczamy korytarze. Wierzchołki niewchodzące w skład korytarzy są w losowy sposób łączone krawędziami ze sobą i z korytarzami. W losowych wierzchołkach niebędących korytarzami umieszczane są cenne zasoby. W dalszych akapitach przedstawiony zostanie szczegółowy opis generowania gier. Generator przyjmuje zestaw parametrów przedstawiony w Tabeli 4.1.

Uwaga wstępna: w przypadku, gdy nie da się spełnić warunków narzuconych przez parametry, cały proces kończy się błędem.

Tabela 4.1: Zestawienie parametrów generatora *Warehouse games*.

parametr	znaczenie
f	liczba pięter
n	szerokość budynku
m	długość budynku
c	liczba skrzyżowań korytarzy
p_d	prawdopodobieństwo wygenerowania drzwi z korytarza do pomieszczenia
p_s	prawdopodobieństwo wygenerowania drzwi pomiędzy pomieszczeniami
N_z	liczba wierzchołków, gdzie znajdują się cenne zasoby (liczność zbioru Z)
$f_{min}^{-,Z}$	dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \in Z$
$f_{max}^{-,Z}$	górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \in Z$
$f_{min}^{-,N}$	dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \notin Z$
$f_{max}^{-,N}$	górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \notin Z$
f_{max}	górne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru Z
f_{min}	dolne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru Z
$u_l^{+,Z}$	wypłata lidera za złapanie naśladowcy w wierzchołku $i \in Z$
$u_l^{+,N}$	wypłata lidera za złapanie naśladowcy w wierzchołku $i \notin Z$
l_{max}	górne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru Z
l_{min}	dolne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru Z

Wszystkie generowane połączenia między wierzchołkami są dwukierunkowe — generowana jest para krawędzi w obie strony. Procedura generująca strukturę grafu przebiega w następujący sposób:

1. Wygeneruj siatkę prostokątną $n \times m$ wierzchołków. Początkowo wierzchołki nie są ze sobą połączone.

2. Wylosuj c punktów skrzyżowań korytarzy. Żadna para punktów nie może być w tej samej kolumnie, tym samym wierszu, sąsiadujących kolumnach, sąsiadujących wierszach. (To znaczy, że dwa równoległe korytarze nie biegną obok siebie, a zawsze są rozdzielone jakimiś pomieszczeniami).

3. Wygeneruj pionowe korytarze: wzdłuż pionowej osi w kolumnach, gdzie znajduje się wylosowane przecięcie korytarza, połącz wszystkie sąsiadujące wierzchołki w danej kolumnie.

4. Wygeneruj poziome korytarze: wzdłuż poziomej osi w kolumnach, gdzie znajduje się wylosowane przecięcie korytarza, połącz wszystkie sąsiadujące wierzchołki w danym wierszu, przykładowy graf po wykonaniu połączeń w pionie i poziomie zaprezentowany jest na Rysunku 4.1.

5. Sklonuj wygenerowaną siatkę wierzchołków z korytarzami f razy. (Każdy z klonów będzie oddzielnym piętrem budynku, wszystkie piętra mają ten sam układ korytarzy).

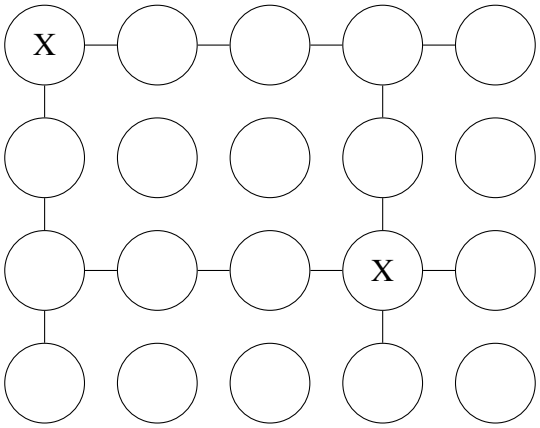
6. Połącz sąsiadujące piętra poprzez połączenie wierzchołków leżących na odpowiadających sobie przecięciach korytarzy (schody lub windy znajdują się na wszystkich przecięciach korytarzy).

7. Wykonaj na każdym piętrze z osobna:

   (a) Między każdą parą wierzchołków, gdzie jeden wierzchołek jest częścią korytarza, a drugi *nie* jest częścią korytarza, niezależnie wygeneruj krawędź z prawdopodobieństwem p_d. (Generuje przejścia z przestrzeni na wynajem do korytarzy).

   (b) Można zaobserwować, że korytarze dzielą pozostałą przestrzeń na rozłączne segmenty, co widać na Rysunku 4.2. Jeśli po poprzednim kroku, do któregoś z segmentów nie prowadzi przejście z żadnego z korytarzy, to ze wszystkich punktów, gdzie ten segment sąsiaduje z korytarzem wybierane jest z prawdopodobieństwem jednostajnym jedna para wierzchołek z segmentu, wierzchołek z korytarza, pomiędzy którymi generowana jest krawędź.

   (c) Powtarzaj następującą operację dopóki istnieją wierzchołki, do których nie istnieje ścieżka z dowolnego punktu korytarza do tego wierzchołka.



Rysunek 4.1: Krok generowania gry *Warehouse Game*: pionowe i poziome korytarze. Znakiem X oznaczone są wierzchołki, które zostały we wcześniejszym kroku wylosowane na punkty przecięcia korytarzy.

i. Z tych wierzchołków weź dowolny wierzchołek, który sąsiaduje z wierzchołkiem, do którego istnieje jakaś ścieżka z korytarza. *Uwaga: taki wierzchołek musi istnieć ze względu na to jak zostały wygenerowane przejścia w poprzednich punktach.*

ii. Połącz te wierzchołki krawędzią.

(d) Dla każdej sąsiadującej pary wierzchołków niebędących częścią korytarza, z prawdopodobieństwem p_s wygeneruj krawędź między tymi wierzchołkami, nic nie rób, jeśli ta krawędź została dodana w poprzednich krokach.

8. Wyznacz losowy wierzchołek niebędący korytarzem (na losowym z pięter) na wierzchołek startowy lidera.

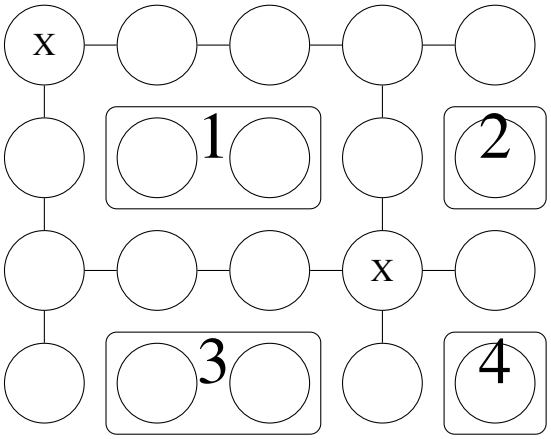
9. Wyznacz losowy wierzchołek wchodzący w skład korytarza na najniższym piętrze i będący na krawędzi siatki (pierwsza lub ostatnia kolumna albo pierwszy lub ostatni wiersz) na wejście — wierzchołek startowy naśladowcy.

Po wygenerowaniu struktury połączeń w grafie, następuje krok generowania wypłat dla poszczególnych wierzchołków:

1. Wylosuj N_Z wierzchołków, które nie są częścią korytarza i nie są punktem startowym lidera na miejsca, gdzie będą przechowywane cenne zasoby.

2. Dla każdego v z wylosowanych wierzchołków wylosuj wypłaty z rozkładu jednostajnego z podanych w parametrach zakresach: $u_f^+(v) \sim U\,(f_{min},f_{max})$, $u_l^-(v) \sim U\,(l_{min},l_{max})$, $u_f^-(v) \sim U\left(f_{min}^{-,Z}, f_{max}^{-,Z}\right)$ oraz wartość wypłaty lidera za złapanie naśladowcy na wartość stałą, będącą parametrem generatora $u_l^+(v) = u_l^{+,Z}$.

3. Dla każdego $v \notin Z$ ustal wypłaty lidera: $u_l^+(v) = u_l^{+,N}$ oraz naśladowcy $u_f^-(v) \sim U\left(f_{min}^{-,N}, f_{max}^{-,N}\right)$



Rysunek 4.2: Wygenerowane korytarze powodują podział pozostałych wierzchołków na rozłączne segmenty. W przykładzie na rysunku są 4 rozłączne segmenty.

Warto zauważyć, że poza wierzchołkami, gdzie znajdują się cenne zasoby, zmienność wypłat graczy nie występuje. Taka decyzja projektowa była kompromisem pomiędzy elastycznością generatora, a liczbą jego parametrów.

Zbiór testowy WHG (Warehouse Games). Jest to pierwszy ze zbiorów stworzonym z użyciem tego generatora, wykorzystany w eksperymentach prezentowanych w pracy [43]. Do wygenerowania użyto generatora z parametrami ustawionymi na wartości przedstawione w Tabeli 4.2. *Uwaga: efektywnie zakresy wypłat u_f^- są zbiorami jednoelementowymi. Wynika to z faktu, że wartości stałe zostały zmienione na zakresy po wygenerowaniu tego zbioru, w trakcie tworzenia zbioru WNZ opisanego dalej.* Początkowo wygenerowano 100 instancji gier na tym zbiorze. Następnie, po ustaleniu liczby kroków $T = 5$ porównano wypłaty uzyskiwane gdy lider zagra strategię mieszaną, która jest jednostajnym rozkładem nad strategiami prostymi i wypłaty dla strategii znalezionych programem liniowym przytoczonym w Rozdziale 3.7.1 (program ten daje optymalną strategię lidera). Odrzucono wszystkie gry, dla których wypłata uzyskiwania przez obie strategie była identyczna. Ten krok pozwolił odrzucić gry trywialne, np. takie, gdzie naśladowca jest blisko najcenniejszego zasobu, a lider jest zbyt daleko, żeby mu przeszkodzić. W efekcie w zbiorze pozostało 25 gier. Zbiór WHG składa się z tych 25 gier, sklonowanych siedmiokrotnie, z ustawioną długością gry $T = 3{,}4{,}5{,}6{,}7{,}8{,}9$ (to oznacza, że gry dla różnego limitu kroków czasowych mają identyczne struktury grafu i wypłat). W związku z tym, że wybrano 25 gier z początkowych 100, pliki w zbiorze testowym nie są numerowane po kolei. Początkowe 100 gier było ponumerowane liczbami od 1 do 100, a po odrzuceniu gier trywialnych numery pozostały niezmienione.

Komentarza wymagają parametry generatora użyte do utworzenia tego zbioru. Przed utworzeniem zbioru został przeprowadzony szereg eksperymentów wstępnych, który pokazał, że wspomniany program liniowy jest w stanie, mając do dyspozycji 256GB pamięci operacyjnej rozwiązywać gry o maksymalnie 6 krokach, przy czym gry sześciokrokowe często liczyły się

Tabela 4.2: Zestawienie parametrów generatora przy tworzeniu zbioru WHG. Znaczenie parametrów jest kopią informacji z Tabeli 4.1

parametr	wartość	znaczenie
f	1	liczba pięter
n	4	szerokość budynku
m	4	długość budynku
c	1	liczba skrzyżowań korytarzy
p_d	0.4	prawdopodobieństwo wygenerowania drzwi z korytarza do pomieszczenia
p_s	0.5	prawdopodobieństwo wygenerowania drzwi pomiędzy pomieszczeniami
N_z	2	liczba wierzchołków, gdzie znajdują się cenne zasoby (liczność zbioru Z)
$f_{min}^{-,Z}$	-0.15	dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \in Z$
$f_{max}^{-,Z}$	-0.15	górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \in Z$
$f_{min}^{-,N}$	-0.05	dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \notin Z$
$f_{max}^{-,N}$	-0.05	górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku $i \notin Z$
f_{min}^{+}	0	dolne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru Z
f_{max}^{+}	1	górne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru Z
$u_l^{+,Z}$	0.05	wypłata lidera za złapanie naśladowcy w wierzchołku $i \in Z$
$u_l^{+,N}$	0.10	wypłata lidera za złapanie naśladowcy w wierzchołku $i \notin Z$
l_{max}	0	górne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru Z
l_{min}	-1	dolne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru Z

wiele dni. W związku z tym przyjęto założenie, że należy losować gry, które dla 5 kroków będą miały rozsądną trudność (graf musi być na tyle mały, żeby gracze byli w stanie w ciągu 5 rund dojść ze swoich punktów startowych do zasobów oraz gęstość cennych zasobów nie powinna być zbyt duża, żeby naśladowca nie miał zbyt dużej przewagi). W związku z tym zdecydowano się na grafy na siatce 4×4, z jednym piętrem i dwoma zasobami. Ze względów praktycznych zdecydowano się na wypłaty graczy z przedziału $[-1, 1]$ (każdą grę można przeskalować, a rozsądny zakres wypłat pozwala łatwiej ustalić parametry podejść metaheurystycznych).

-

Zbiór testowy WNZ (WHG Non-Zero sum). Parametry dotyczące wypłat graczy, użyte w generatorze WHG są naturalne dla wielu zastosowań: lider otrzymuje ujemną wypłatę, gdy naśladowca otrzyma wypłatę dodatnią i na odwrót. Jednak z punktu widzenia różnorodności gier testowych, taka korelacja wypłat jest cechą niepożądaną. Aby przetestować metody dedykowane grom o sumie niezerowej, warto rozszerzyć repertuar gier, o takie gry, w których wypłaty są ze sobą mniej skorelowane. W związku z tym, w ramach dalszych prac, których efektem są metody opisane w kolejnych rozdziałach, wygenerowaliśmy drugi zbiór testowy, WNZ, w którym zmienione zostały zakresy, z których losowane są wypłaty dla graczy. Sposób tworzenia zbioru WNZ był analogiczny, jak sposób tworzenia zbioru WHG i po odrzuceniu trywialnych gier również pozostało 25 instancji[1]. Jedyną różnicą były parametry generatora dotyczące wypłat. W Tabeli 4.3 przedstawione są wykorzystane parametry. Tekstem pogrubionym zaznaczone są parametry, które różnią się w stosunku do WHG.

W celu zmierzenia różnorodności wypłat w grach został wykorzystany współczynnik korelacji Pearsona. Dla każdej gry wzięto wypłaty graczy w liściach, i pary wypłata lidera, wypłata naśladowcy z każdego liścia potraktowano jako zmienne z dwóch rozkładów, w których badana jest korelacja. Dla każdej gry z osobna wyliczono w ten sposób współczynnik korelacji Pearsona, a następnie uśredniono wyniki dla wszystkich gier ze zbioru. W ten sposób dla gier WHG uzyskano wynik -0.82 (to oznacza grę bliską sumy zerowej, bo dla gry o sumie zerowej współczynnik wyniesie -1), natomiast dla zbioru WNZ wynik ten wyniósł -0.57, gry są dalsze od sumy zerowej, co było celem tworzenia tego zbioru.

Zbiór testowy Basic. Przed zaprojektowaniem generatorów, we wczesnych eksperymentach używany był zbiór 12 gier zaprojektowanych ręcznie. Zaprojektowano trzy topologie grafów, które mają na celu sprawdzać zachowania metod w kilku różnych sytuacjach. Wszystkie gry z tego zbioru testowego były rozgrywane z limitem rund $T = 5$.

- Graf 1, przedstawiony na Rysunku 4.3, topologia gwiazdy, cele i punkty startowe graczy są w liściach. Zablokowanie przez lidera środkowego wierzchołka powoduje, że naśladowca nie ma żadnej możliwości osiągnięcia celu. Pozornie mogłoby się wydawać,

[1]Fakt, że liczności zbiorów WHG i WNZ są równe jest przypadkiem. Metoda odrzucania trywialnych gier w żaden sposób nie wymusza tej liczby.

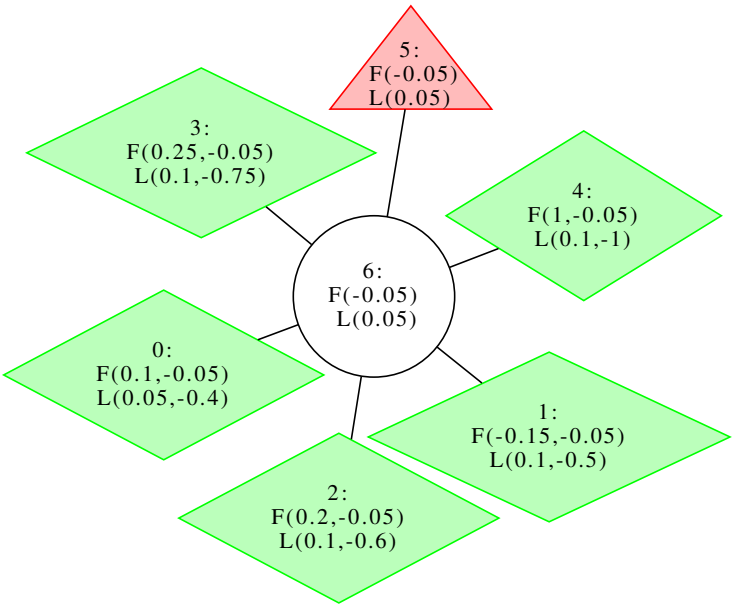
Tabela 4.3: Zestawienie parametrów generatora przy tworzeniu zbioru WNZ. Pogrubionym tekstem zaznaczono parametry zmienione względem WHG.

parametr	wartość	znaczenie
f	1	liczba pięter
n	4	szerokość budynku
m	4	długość budynku
c	1	liczba skrzyżowań korytarzy
p_d	0.4	prawdopodobieństwo wygenerowania drzwi z korytarza do pomieszczenia
p_s	0.5	prawdopodobieństwo wygenerowania drzwi pomiędzy pomieszczeniami
N_z	2	liczba wierzchołków, gdzie znajdują się cenne zasoby (liczność zbioru Z)
$f_{min}^{-,Z}$	-1	**dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku** $i \in Z$
$f_{max}^{-,Z}$	0.2	**górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku** $i \in Z$
$f_{min}^{-,N}$	-1	**dolne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku** $i \notin Z$
$f_{max}^{-,N}$	0	**górne ograniczenie na wypłatę naśladowcy za bycie złapanym w wierzchołku** $i \notin Z$
f_{min}^{+}	-0.2	**dolne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru** Z
f_{max}^{+}	1	**górne ograniczenie na wypłatę naśladowcy za dojście do wierzchołka ze zbioru** Z
$u_l^{+,Z}$	0.2	**wypłata lidera za złapanie naśladowcy w wierzchołku** $i \in Z$
$u_l^{+,N}$	0.1w	**wypłata lidera za złapanie naśladowcy w wierzchołku** $i \notin Z$
l_{max}	0.2	**górne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru** Z
l_{min}	-1	**dolne ograniczenie na wypłatę lidera, gdy naśladowca dojdzie do wierzchołka ze zbioru** Z

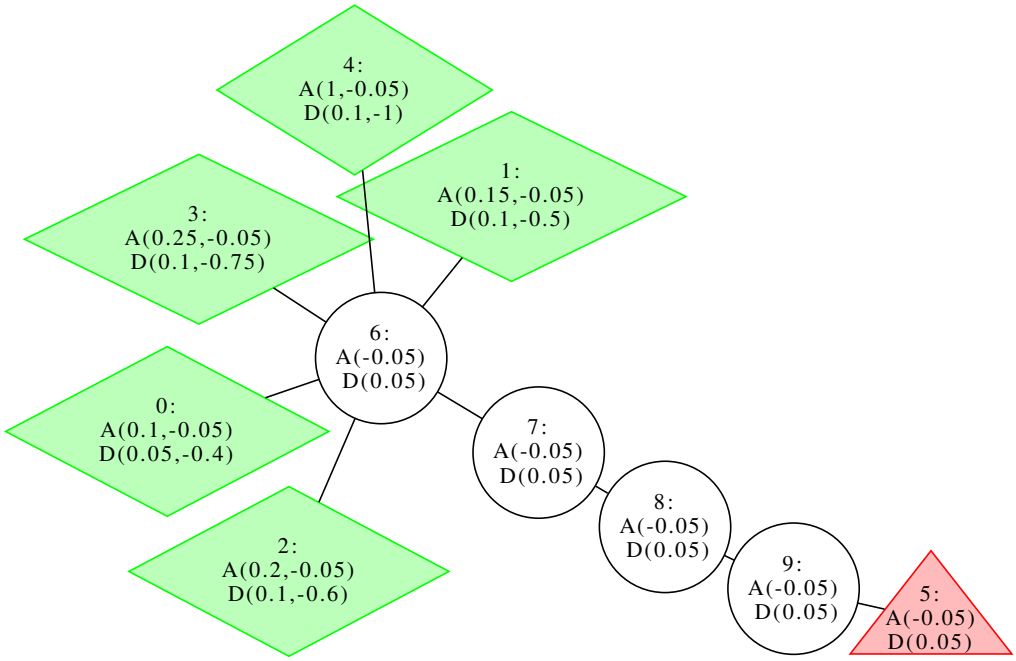
że optymalną strategię lidera jest po prostu zajęcie środkowego wierzchołka. Nie jest to jednak prawdą. Ze względu na to, że lider za złapanie naśladowcy otrzymuje niewielką wypłatę, optymalna strategia polega na zajmowaniu środkowego węzła przez wszystkie rundy z prawdopodobieństwem 50%, a w pozostałych przypadkach zajmowanie środkowego węzła przez $T - 1$ rund i przejście do punktu startowego naśladowcy w ostatnim ruchu. W takim układzie naśladowca będzie łapany z prawdopodobieństwem 50% (optymalną odpowiedzią jest zarówno przebywanie w wierzchołku startowym przez całą grę, jak i przejście w ostatnim ruchu do wierzchołka środkowego). W wynikach testów ten graf jest oznaczony jako *game1*.

- Graf 2, który powstał z grafu 1 poprzez modyfikację wierzchołka startowego naśladowcy. Zamiast wierzchołka startowego naśladowcy, do wierzchołka centralnego podłączona jest tym razem ścieżka złożona z 4 wierzchołków, a punkt startowy naśladowcy znajduje się na końcu ścieżki oddalonym od wierzchołka centralnego. Graf ten przedstawiony jest na Rysunku 4.4. Stan Równowagi w tym grafie jest bardziej skomplikowany, bo idąc ścieżką do wierzchołka 5 lider i naśladowca mogą się wyminąć, to wymusza na liderze bardziej zachowawczą strategię. W wynikach testów ten graf jest oznaczony jako *game2*.

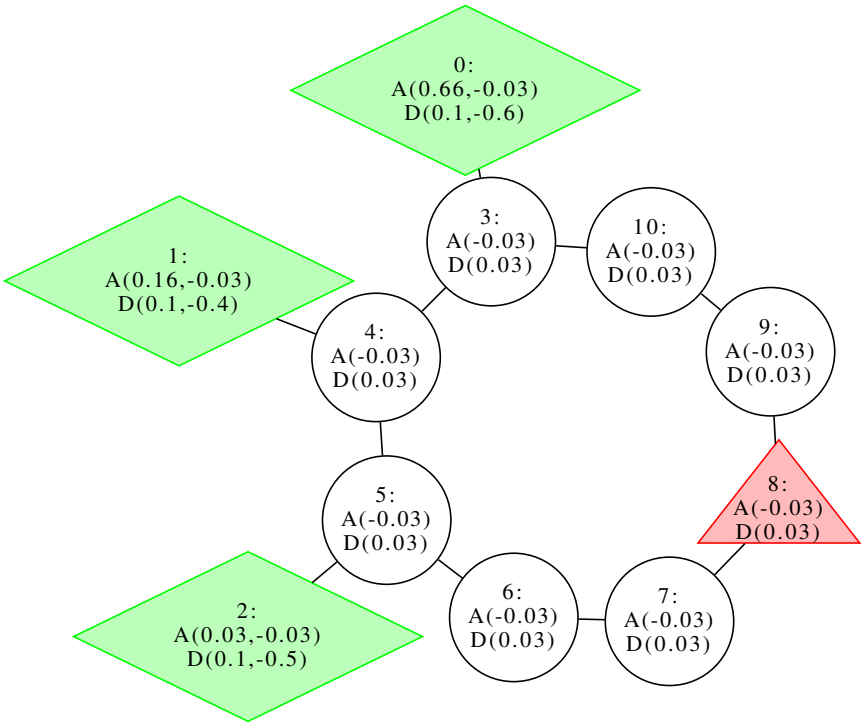
- Graf 3 to sytuacja, gdzie naśladowca ma do wyboru dwie drogi prowadzące do zasobów. Struktura grafu przedstawiona jest na Rysunku 4.5. Są 3 wierzchołki z cennymi zasobami: 0, 1, 2, przy czym wypłata naśladowcy w wierzchołku 1 jest dużo mniej atrakcyjna niż w pozostałych. Lider startuje w wierzchołku o numerze 0, naśladowca w wierzchołku o numerze 5, który jest w odległości 4 kroków od zasobów w wierzchołkach 0 i 2, a do każdego z tych zasobów prowadzi oddzielna ścieżka. W tym układzie grafu poszukiwanie strategii lidera, to dwa wyzwania: po pierwsze lider musi balansować pomiędzy blokowaniem dostępu do wierzchołka 0 i do wierzchołka 2 (wierzchołek 1 jest nieatrakcyjny i optymalna strategia naśladowcy nie będzie go atakować). Naśladowca ma rozłączne ścieżki, którymi może osiągnąć ten wierzchołek. Drugie wyzwanie jest mniej widoczne, ale równie istotne, lider otrzymuje wyższą wypłatę za złapanie naśladowcy w wierzchołkach z zasobami niż w pozostałych wierzchołkach, więc powinien dążyć do tego, żeby złapanie nastąpiło właśnie w wierzchołku z zasobem, a nie wcześniej. W przeciwieństwie do dwóch poprzednich gier, które w zbiorze basic mają jedną instancję, w przypadku tej gry znajduje się 10 gier zbudowanych na tym samym grafie, wszystkie mają dokładnie tę samą strukturę połączeń wierzchołków, a różnią się jedynie wartościami wypłat (i co za tym idzie stanem Równowagi Stackelberga). W wynikach eksperymentalnych warianty tego grafu są oznaczone jako *game3* (prezentowany na Rysunku 4.5), *game3a, game4b, game3c, game3d, game4, game4a, game4b, game4c, game4d*. Wszystkie grafy z tego zbioru zaprezentowane są w Załączniku A.



Rysunek 4.3: Graf 1 ze zbioru Basic. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 5 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



Rysunek 4.4: Graf 2 ze zbioru Basic. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 5 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



Rysunek 4.5: Graf 3 ze zbioru Basic. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.

4.2 Metoda Mixed-UCT: szybka aproksymacja stanu Równowagi Stackelberga z użyciem próbkowania strategi lidera przy ustalonej strategii naśladowcy

W tym rozdziale jest prezentowane pierwsze podejście do poszukiwania Równowagi Stackelberga w grach wielokrokowych o sumie niezerowej zaproponowane przez autora rozprawy i jego promotora, nazwane *Mixed-UCT*. Celem prac w ramach których przygotowano tę metodę było sprawdzenie przydatności podejścia *Upper Confidence Bound applied to Trees* (UCT) w problemie poszukiwania wspomnianych strategii. Wyniki badań nad metodą *Mixed-UCT* opisane poniżej, zostały opublikowane w pracach [43, 42, 41]. Dalszy ciąg tego podrozdziału jest zorganizowany w następujący sposób: najpierw opisane jest podejście *Upper Confidence Bound applied to Trees*, które jest bazą dla proponowanej metody poszukiwania Równowagi Stackelberga, następnie opisana jest właściwa metoda *Mixed-UCT* — budowa i działanie. W kolejnej sekcji zaprezentowane są przeprowadzone eksperymenty i wyniki, które oceniają szybkość i skuteczność działania zaproponowanej metody. Sekcja zakończona jest wnioskami i posumowaniem możliwości i ograniczeń metody.

Głównym celem budowy metody *Mixed-UCT* było sprawdzenie, czy można zastosować bezpośrednio podejście UCT do poszukiwania Równowagi Stackelberga. W związku z tym, że metoda powstała w ramach początkowych badań, jest mniej ogólna. Gra musi, poza cechami wymienionymi we wstępie, czyli gra Stackelberga z niepełną informacją o sumie niezerowej, z doskonałą pamięcią, spełniać dodatkową własność. Cechą, która jest wymagana, jest specyficzna struktura zbiorów informacyjnych: (†) **zbiory informacyjne są jednoznacznie zdefiniowane przez sekwencję ruchów wykonanych przez gracza, dla którego ten zbiór informacyjny jest punktem decyzyjnym**.

Podejścia z rodziny MCTS, w szczególności metoda UCT, zostały stworzone z myślą o grach z pełną informacją, na przykład grach planszowych typu Havannah, czy warcaby. Typowo są wykorzystywane do znalezienia jednego, najlepszego ruchu w danym stanie W przypadku strategii lidera w grach Stackelberga konieczne jest rozważanie strategii mieszanej, a nie pojedynczego ruchu, co zostało omówione w Rozdziale 2.5. To oznacza, że konieczna jest modyfikacja podejścia, która pozwoli uzyskiwać strategię mieszaną zamiast pojedynczego ruchu. Drugim problemem jest fakt, że najpopularniejsze podejście do stosowania UCT w grach dla dwu graczy, to stosowanie rozgrywki, gdzie każdy z graczy podejmuje decyzję na podstawie UCT. Równowaga Stackelberga jest zdefiniowana w sposób asymetryczny i sposoby podejmowania decyzji przez lidera i naśladowcę istotnie się od siebie różnią. Zamiast tego zostanie zastosowane podejście alternatywne, gdzie lider znajduje swoją strategię używając UCT w jednoosobowej grze powstałej przez ustalenie strategii naśladowcy. Strategia naśladowcy jest znajdowana poza UCT. Największym wyzwaniem przy budowie *Mixed-UCT* było opracowanie techniki, która pozwoli na podstawie statystyk symulacji zebranych w drzewie UCT zaproponować strategię

mieszaną dla lidera. W ramach prac zostały zaproponowane i sprawdzone 3 warianty, które są prezentowane w dalszej części tego rozdziału.

Pierwszym elementem *Mixed-UCT* jest jednoosobowa gra dla lidera, gdy jest ustalona strategia naśladowcy. Idea gry polega na tym, że rozgrywamy oryginalną grę, dla której poszukujemy Równowagi Stackelberga. W przypadku, gdy osiągnięto stan, który jest punktem decyzyjnym lidera, lider wybiera ruch. W przypadku, gdy osiągnięto punkt decyzyjny, w którym ruch wybiera naśladowca, to wykonywany jest ruch zgodnie z ustaloną strategią naśladowcy. Strategia naśladowcy może być zarówno strategia prostą, jak i strategią mieszaną. Jeśli jest to strategia mieszana, to z punktu widzenia lidera gra może być niedeterministyczna. Powoduje to konieczność nieznacznej modyfikacji podstawowej metody UCT. W poniższych opisach metody UCT i kolejnych, jeśli jest mowa o *ustalonej strategii naśladowcy*, to sformułowanie to dotyczy właśnie strategii, która została wykorzystana do definicji tej gry dla jednego gracza.

4.2.1 Metoda Upper Confidence Bound applied to Trees

W 2006 roku Kocsis i Szepesvári zaproponowali metodę poszukiwania najlepszych ruchach w grach dla dwóch graczy z pełną informacją [50]. Pokazano, że dla dużej liczby iteracji, ruch proponowany przez metodę zbiega do zagrania dającego najwyższą wypłatę (przy założeniu kontynuacji optymalnej gry) [50]. Metoda ta opiera się o ukierunkowane próbkowanie ruchów w drzewie gry. Metoda ta opiera się o wielokrotne symulowanie rozgrywki. Początkowe symulacje są losowe, w trakcie symulacji zbierane są statystyki, które są następnie używane do ukierunkowania kolejnych rozgrywek na sprawdzanie ruchów, które są bardziej obiecujące. To podejście jest przedstawicielem grupy metod określanych terminem *best-first search* (przeszukiwanie od najlepszych). Metoda UCT jest wariantem podejścia określanego mianem *Monte Carlo Tree Search* (MCTS, Przeszukiwanie drzew z użyciem Monte Carlo) [23]. Metoda służy do znalezienia dobrego ruchu w dowolnym stanie gry z pełną informacją. Istnienie metod z rodziny MCTS i ich znaczenie zostało już opisane we wstępie, w Rozdziale 1.2.

Zasada działania MCTS/UCT. Na wejściu metody podana jest gra dla jednego gracza i stan, dla którego poszukiwany jest ruch. Główną strukturą danych jest drzewo, w którym przechowywane są oszacowania wypłaty, którą gracz uzyska po zagraniu tego ruchu. Wierzchołki drzewa są etykietowane stanami gry, a krawędzie są etykietowane ruchami. Struktura drzewa jest analogiczna do struktury gry: korzeniem drzewa jest stan gry, dla którego puszkujemy ruchu, krawędzie odpowiadają dopuszczalnym ruchom w grze i łączą stan, z którego wykonano ruch ze stanem wynikowym po danym ruchu. Z każdą krawędzią drzewa są związanie dwie wartości: liczba iteracji algorytmu, w których został odwiedzony dany ruch oraz średnia wartość wypłaty uzyskana w trakcie tych iteracji. Drzewo jest rozbudowywane wraz z kolejnymi iteracjami. Początkowo drzewo składa się wyłącznie z wierzchołka korzenia, a w każdej iteracji może zostać dodany jeden nowy wierzchołek. Każda iteracja procedury jest złożona z czterech faz:

Selekcja W tej fazie następuje przejście przez stany i ruchy znajdujące się w strukturze danych. Dla każdego węzła, jeśli w drzewie znajdują się wszystkie ruchy, które można wykonać z odpowiadającego temu węzłowi stanu, wybierz ruch na podstawie *polityki selekcji*, która bazuje na liczbie odwiedzin i średniej wartości wypłaty danego ruchu. Przejdź do po krawędzi do węzła wskazanego przez ten ruch. Powtarzaj aż do osiągnięcia węzła, który nie ma wszystkich następników wynikających z możliwych ruchów w grze lub węzła terminalnego.

Ekspansja Jeśli został osiągnięty węzeł terminalny, przejdź do kolejnego punktu. W przeciwnym wypadku wybierz, zgodnie z *polityką domyślną* jeden z ruchów dopuszczalnych w grze, które nie mają odpowiadającej krawędzi w drzewie i dodaj go do drzewa. Przejdź do kolejnego punktu.

Symulacja Od stanu, który odpowiada węzłowi dodanemu w ostatnim punkcie symuluj rozgrywkę (bez dodawania węzłów do drzewa ze statystykami) za każdym razem wybierając ruch z użyciem *polityki domyślnej*, aż do momentu, gdy został osiągnięty stan terminalny.

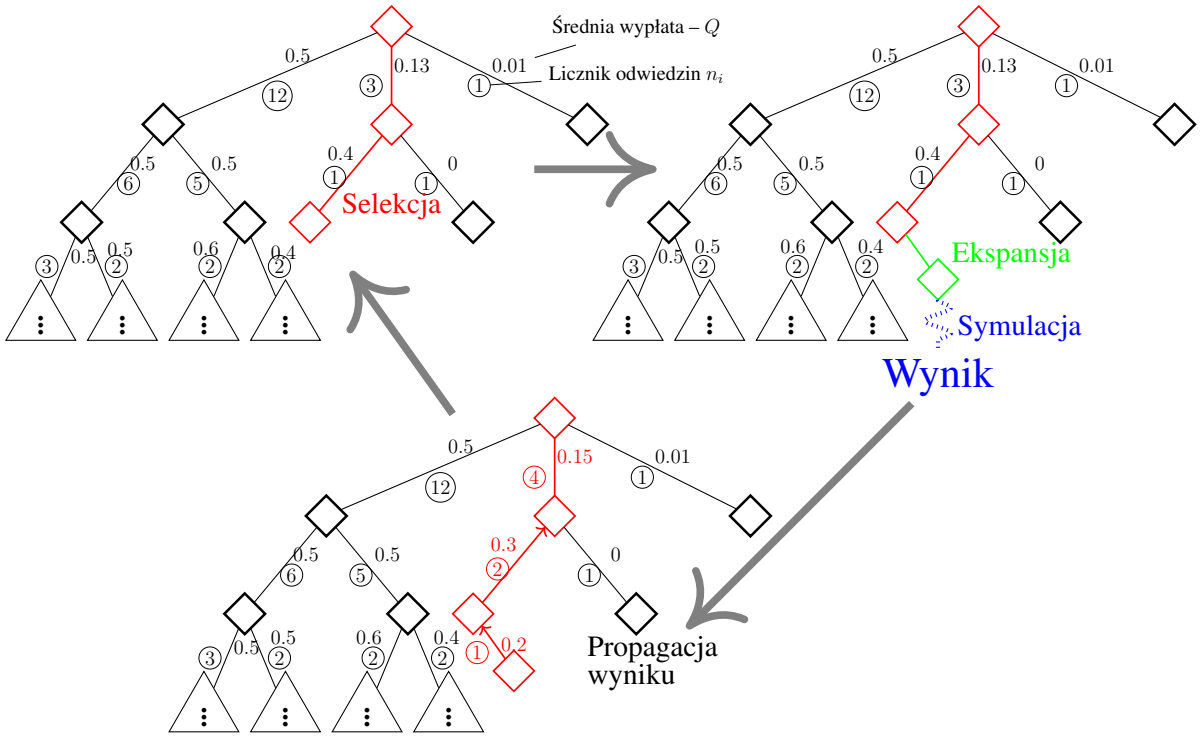
Propagacja wsteczna W stanie terminalnym gracz otrzymuje pewną wypłatę. Dla wszystkich krawędzi odpowiadających ruchom wybranym w dwóch pierwszych fazach zaktualizuj średnią wypłatę, uwzględniając wypłatę z tej symulacji, oraz zwiększ licznik odwiedzin o 1. Następnie przejdź do kolejnej iteracji metody.

Proces iteracyjny jest powtarzany aż do osiągnięcia warunku stopu (zwykle osiągnięcie limitu czasu na znalezienie ruchu). Po zakończeniu metody wynikiem jest ruch, dla którego liczba odwiedzin jest największa. Przebieg metody MCTS podsumowany jest na Rysunku 4.6.

Powyższy schemat, wspólny dla wielu metod, klasyfikowanych jako MCTS, używa pojęcia *polityki selekcji* oraz *polityki domyślnej*. Te polityki ustalane są w każdej z konkretnych metod osobno. W przypadku metody UCT *polityka selekcji* wykorzystuje wzór UCB1 [4]:

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}, \tag{4.1}$$

gdzie: s – bieżący stan gry, $A(s)$ — zbiór wszystkich dostępnych akcji, $N(s,a)$ licznik odwiedzin ruchu a ze stanu s w drzewie, $Q(s,a)$ – średnia wypłata danego ruchu a w drzewie, $N(s) = \sum_{a \in A(s)} N(s,a)$, C – parametr metody. W ramach selekcji wybierany jest ruch a^*. *Polityka domyślna* jest zaimplementowana jako wybór z rozkładu jednostajnego spośród wszystkich dostępnych ruchów. Wspomniana *polityka selekcji*, przedstawiona w równaniu (4.1) jest typowym przykładem na zachowanie równowagi pomiędzy eksploracją i eksploatacją. Pierwszy składnik wartości pod $\arg\max$ to średnia wypłata (szacowana jakość) zadanego ruchu, stosowanie tylko tej wartości byłoby postępowaniem zachłannym, gdzie algorytm zawsze testuje najbardziej obiecujący ruch wedle aktualnego stanu wiedzy. Drugi składnik jest tym większy im



Rysunek 4.6: Schemat działania metody *Monte Carlo Tree Search* w wersji stosowanej w grze dla jednego gracza. Zaprezentowane drzewo, to drzewo przechowujące statystyki na tematy ruchów. W kółku liczniki odwiedzin, obok średnia wartość wypłaty. Na czerwono zaznaczony ścieżkę wybraną w ramach selekcji, na zielono węzeł dodany w ramach ekspansji, a na niebiesko fasę symulacji. W dolnej części rysunku znajduje się wizualizacja propagacji wyniku, w której zostały uaktualnione statystyki na wybranej wcześniej ścieżce. Szare strzałki oznaczają przejście między fasami w ramach iteracyjnego przebiegu MCTS.

rzadziej, w stosunku do innych ruchów z danego stanu, był odwiedzany dany ruch. Ten składnik odpowiada za eksplorację, preferuje sprawdzenie ruchów, które były rzadko odwiedzane i wiedza o ich wypłacie jest mniej pewna. Za balans pomiędzy eksploatacją (pierwszym składnikiem) i eksploracją (drugim składnikiem) odpowiada współczynnik C, który typowo, dla gier z wypłatami z przedziału $[0,1]$ ustawia się na wartość $\sqrt{2}$ (warto zwrócić uwagę, że zakresy wypłat w grze wpływają na możliwe wartości składnika $Q(s,a)$, stąd związek współczynnika C z rozpiętością wypłat w grze).

Opis powyżej zakłada grę dla jednego gracza z pełną informacją. W wielu przypadkach rozważane gry są grami dla dwu lub więcej graczy, a powyższe podejście można bardzo łatwo uogólnić. Zmiana, pozwalająca na poszukiwanie ruchu w grze dla większej liczby graczy opiera się o założenie, że każdy z przeciwników zagra optymalny ruch. W takiej sytuacji w drzewie przechowujemy nie jedną średnią wartość wypłaty, ale wektor wartości średnich, dla każdego gracza z osobna. Polityka selekcji w danym węźle wykorzystuje element wektora odpowiadający graczowi, który rusza się w zadanym węźle, a wartości dla pozostałych graczy pomija.

4.2.2 I2UCT — modyfikacja UCT dla gier z niepełną informacją

Podejścia MCTS wymagają, żeby możliwe było symulowanie rozgrywki z dowolnego stanu gry. Jest to problematyczne, gdy chcemy zastosować MCTS do gier z niepełną informacją, gdzie dysponujemy tylko projekcją stanu gry widzianą przez gracza, który aktualnie podejmuje decyzję i nie wiemy jak wygląda pełny stan gry. W przypadku metody *Mixed-UCT* ten problem pojawia się, gdy poszukujemy strategii w grze dla jednego gracza, gdzie lider gra przeciwko ustalonej strategii mieszanej naśladowcy. W związku z tym w ramach projektowania *Mixed-UCT* powstało podejście *I2UCT* (ang. Imperfect Information UCT, UCT dla niepełnej informacji). Jest to modyfikacja metody UCT, która pozwala na zastosowanie UCT do gry pomocniczej tworzonej w trakcie przebiegu *Mixed-UCT*. Podejście I2UCT wymaga, aby stan początkowy gry był dobrze znany graczowi oraz aby było możliwe symulowanie gry i uzyskiwanie projekcji stanu widzianej przez gracza w kolejnych ruchach. Ważne jest też, żeby spełnione było założenie (†) ze strony 115. Podejście *I2UCT* nie jest metodą ogólną i nie da się go stosować do dowolnej gry z niepełną informacją (np. pokera), w zamian za to, dzięki specjalizacji do konkretnego rodzaju gier, może działać szybciej.

Działanie metody *I2UCT* zostało przedstawione w Algorytmie 4.1. Główna idea algorytmu to rozpoczęcie, niezależnie od stanu, w jakim się znajdujemy, od korzenia gry (który jest dobrze znany i nie ma tu problemu z niepełną informacją), a następnie przeprowadzenie rozgrywki zgodnie z ruchami wykonanymi przez lidera, które wykonał od początku rozgrywki do bieżącego punktu decyzyjnego i zgodnie z ruchami próbkowanymi ze strategii mieszanej naśladowcy. W ten sposób uzyskujemy stan gry, który potencjalnie może być stanem, którego projekcję widzi lider. W tym miejscu ważne jest wspomniane założenie (†), które gwarantuje, że widziana po takiej symulacji projekcja stanu lidera będzie zgodna z tą faktycznie obserwowaną. Ma-

jąc dane: procedurę, która potrafi symulować grę, ustaloną strategię lidera oraz wektor ruchów lidera, które doprowadziły do bieżącego stanu każdą symulację UCT przeprowadzamy w następujący sposób: w linii 2 rozpoczyna się przejście od startowego stanu gry. Pętla następująca po tej linii służy do dojścia do stanu widzianego przez lidera, z którego chcemy wyznaczyć najlepszy możliwy ruch. Biorąc kolejne, od najwcześniejszych, ruchy z wektora ruchów wykonanych przez lidera, prowadzimy symulację. W linii 4 symulujemy zagranie przez lidera kolejnego ruchu i przejście do kolejnego stanu. Kolejny stan, który jest punktem decyzyjnym lidera jest osiągany poprzez symulację rozgrywki, wykorzystując ustaloną strategię naśladowcy. W efekcie w linii 6 uzyskany jest stan gry, który należy do tego samego zbioru informacyjnego, co stan widziany przez lidera, a prawdopodobieństwo uzyskania konkretnego stanu (pełnego, niewidocznego dla lidera) jest zgodne z prawdopodobieństwami ruchów z ustalonej strategii naśladowcy. Ten stan użyty jest jako stan startowy do pojedynczej iteracji niezmodyfikowanej metody UCT. Całość jest powtarzana w pętli w linii 1 aż do osiągnięcia warunku stopu. Ta pętla jest odpowiednikiem głównej pętli w oryginalnym algorytmie UCT.

Algorytm 4.1: I2-UCT — Imperfect-Information UCT

Input: moves— wektor ruchów lidera, który doprowadził go do bieżącego stanu
depth— głębokość stanu w drzewie gry (długość wektora moves))

```
1 while ¬stopCondition() do
2 |   state ← InitialState
3 |   for d ← 1 ··· depth − 1 do
4 |   |   MakeMove(state, moves [d]) // Wykonaj ruch lidera i
  |   |      zasymuluj odpowiedź naśladowcy
5 |   end
6 |   IteracjaUCT(state)// Pojedyncza iteracja UCT (selekcja,
  |      ekspansja, symulacja, propagacja)
7 end
```

4.2.3 Budowa metody Mixed-UCT

Metoda *Mixed-UCT* działa w sposób iteracyjny, przedstawiony w Algorytmie 4.2. W algorytmie zdefiniowane są dwie zmienne, które przenoszą informację pomiędzy iteracjami:

- strategia naśladowcy (δ_f) — strategia używana przy wywołaniach metody *I2UCT*, strategia ta może być inicjowana w różny sposób, a sposoby inicjacji opisane są w dalszej części rozdziału,
- drzewo statystyk UCT ($UCTTree$) — stan drzewa od którego I2UCT rozpoczyna działanie w kolejnych iteracjach metody. Początkowo jest to puste drzewo składające się z samego korzenia. W kolejnych iteracjach część statystyk jest zapisywana i przenoszona dalej. Sposób przenoszenia informacji zależy od parametrów metody, zaimplementowane sposoby są opisane w dalszej części rozdziału.

Algorytm 4.2: Główna pętla metody *Mixed-UCT*

```
   input: G – gra Stackelberga
 1 δ_f ← initializeFollowerStaregy();
 2 δ_l^* ← ∅;
 3 UCTtree ← rootNode(G);
 4 while ¬stopCondition() do
 5 |   moves ← []// Pusty wektor
 6 |   UCTsubtree ← UCTtree while notTerminal(UCTsubtree) do
 7 |   |   I2UCT(UCTsubtree, singlePlayerGame(G, delta_f),moves);
 8 |   |   move ← bestMove(poddrzewoUCT);
 9 |   |   moves ← append(moves, move);
10 |   |   UCTsubtree ← accessSubtree(UCTsubtree, move);
11 |   end
12 |   δ_l ← extractLeaderStrategy(UCTtree);
13 |   UCTtree ← prepareForNextIteration(UCTtree);
14 |   δ_f ← updateFollowerStategy(δ_f, δ_l);
15 |   if u_l(δ_l^*, BR(δ_l^*)) < u_l(δ_l, BR(δ_l)) then
16 |   |   δ_l^* ← δ_l;
17 |   end
18 end
```

Główną częścią algorytmu jest zewnętrzna pętla (linie 6–17), w której każdorazowo:

1. wykonywane jest próbkowanie *I2UCT* przeciw strategii naśladowcy przechowywanej w zmiennej δ_f,

2. drzewo statystyk zebrane przez *I2UCT* zamieniane jest w strategię mieszaną lidera (linia 12),

3. statystyki zebrane w tej iteracji przenoszone są w pewien sposób do kolejnej iteracji (linia 13),

4. na podstawie uzyskanej strategii lidera aktualizowana jest strategia naśladowcy δ_f (linia 14),

Zewnętrzna pętla powtarzana jest do osiągnięcia warunku stopu. Bieżąca implementacja używa alternatywy dwóch kryteriów: osiągnięto limit liczby iteracji lub osiągnięto limit liczby iteracji, w których strategia π_l nie poprawiała wyniku lidera.

Uwaga. W każdej zewnętrznej iteracji algorytmu znajdowana jest pewna strategia lidera δ_l. Nie ma jednak gwarancji, że strategie w kolejnych iteracjach zawsze będą dawały lepszy rezultat (mierzony przeciwko optymalnej odpowiedzi naśladowcy), niż te, które znaleziono w poprzednich iteracjach. W związku z tym utrzymywana jest zmienna δ_l^*, w której przechowywana jest najlepsza dotąd znaleziona strategia lidera. Na końcu pętli zawarte jest sprawdzenie, które aktualizuje tę najlepszą strategię.

Opisane dalej metody zamiany drzewa statystyk w strategię mieszaną lidera (wywoływane w linii 12) wymagają, żeby istniały gałęzie drzewa statystyk rozwinięte aż do stanu terminalnego. W związku z tym, zamiast wykonywać pojedynczy przebieg algorytmu *I2UCT*, stosowane jest podejście iteracyjne (pętla w linii 6). W pierwszym kroku metoda I2UCT uruchamiana jest z korzenia drzewa. W drugim kroku następuje przejście do najczęściej odwiedzanego w poprzednim kroku węzła i kolejne uruchomienie I2UCT. Krok zejścia głębiej jest powtarzany aż do osiągnięcia węzła terminalnego. W każdym pogłębionym kroku wykorzystywane jest poddrzewo tego samego drzewa UCT. Warunek stopu stosowany w metodzie *I2UCT* to osiągnięcie przynajmniej $simCount$ odwiedzin aktualnie rozpatrywanego korzenia drzewa UCT w ramach tej iteracji pętli w linii 4. Ten warunek oznacza, że w korzeniu gry wykonywane jest dokładnie $simCount$ symulacji. W przypadku kroków, gdzie rozpoczynamy od niższej partii drzewa UCT, wykonywane jest $simCount$ symulacji pomniejszone o liczbę symulacji, które w ramach fazy selekcji odwiedziły ten węzeł w poprzednich obrotach pętli 6.

Przedstawiony szkielet metody nie daje pełnego obrazu działania *Mixed-UCT*, konieczny jest wybór procedur będących parametrami metody. W pierwszej kolejności zostaną przedstawione metody konstruowania strategii lidera z użyciem statystyk z drzewa UCT.

Sposoby budowy strategii lidera na podstawie statystyk zebranych w drzewie UCT

Techniki konstruowania strategii lidera na podstawie drzewa UCT były jednym z pierwszych wyzwań badawczych przy budowie *Mixed-UCT*. Badania w tym obszarze prowadzone przez autora rozprawy wraz z promotorem zostały podsumowane w pracy [41]. Praca ta prezentuje trzy możliwe podejścia do konstrukcji strategii mieszanej lidera w metodzie *Mixed-UCT*, które zostaną przedstawione dalej:

- ekstrakcja z pojedynczego drzewa (PD, ang. single tree),
- ekstrakcja najlepszej ścieżki z każdego z drzew (NS, ang. best path).
- ekstrakcja z wycinków z wielu drzew (WD, ang. tree slice),

Ekstrakcja z pojedynczego drzewa. Podejście polega na zbudowaniu strategii behawioralnej z prawdopodobieństwami ruchów związanymi ze statystykami w drzewie UCT: licznikiem odwiedzin oraz średnią wypłatą. Drzewo UCT składa się ze stanów i ruchów, ruchy dostępne z danego stanu są krawędziami prowadzącymi do stanów będących efektem wykonania tych ruchów. Oznacza to, że struktura tego drzewa jest analogiczna do struktury strategii behawioralnej, o której mowa w Rozdziale 2.3. Jedyną różnicą jest to, że w strategii behawioralnej ruchy możliwe w danym stanie mają przypisane prawdopodobieństwa, a w przypadku drzewa UCT ruchy mają przypisane dwie wartości: licznik odwiedzin i średnią wypłatę. Metoda ekstrakcji strategii z pojedynczego drzewa polega na zamianie, dla każdego ruchu, wspomnianej pary

wartości na jeden nieujemny współczynnik. Następnym krokiem jest normalizacja współczynników, w obrębie każdego stanu z osobna, tak, żeby sumowały się do 1, czyli żeby współczynniki definiowały rozkład prawdopodobieństwa nad ruchami w danym stanie. Wszystkie gałęzie drzewa, które nie kończą się stanem terminalnym są pomijane. W ten sposób uzyskiwana jest strategia behawioralna. Przed zwróceniem ostatecznej strategii podejmowany jest jeszcze jeden krok. Z racji tego, że integralną częścią metody UCT jest eksploracja wyrażona drugim składnikiem wzoru (4.1), w ramach drzewa UCT znajdują się ruchy, które są dla gracza niekorzystne, ale zostały kilkukrotnie odwiedzone ze względu na eksplorację właśnie. Na koniec do każdego rozkładu prawdopodobieństwa aplikowany jest filtr, który usuwa z rozkładu ruchy o prawdopodobieństwie, które jest poniżej progu będącego parametrem metody. Po zastosowaniu filtra ponownie stosowana jest normalizacja wag tak, aby sumowały się do 1.

Istotnym pytaniem pozostaje to, w jaki sposób zamieniać współczynniki w drzewie UCT na nieujemne wagi do rozkładu prawdopodobieństwa. Analiza tego problemu była przedmiotem wstępnych, przeprowadzonych na niedużą skalę, eksperymentów na zbiorze gier *Basic*. Sprawdzano następujące podejścia, w poniższych wzorach użyte są następujące oznaczenia w_i — waga proporcjonalna do prawdopodobieństwa ruchu i, Q_i — średnia wypłata w drzewie UCT dla ruchu i, n_i — licznik odwiedzin ruchu i w drzewie UCT, S — zbiór wszystkich ruchów, które są dostępne ze stanu z którego grany jest ruch i oraz znajdują się w drzewie UCT:

1. $w_i = n_i / \sum_{j \in S} n_j$
2. $w_i = Q_i - \min_{j \in S} Q_j$
3. $w_i = Q_i - Q_{min}^{global}$, gdzie Q_{min}^{global} to minimalna wartość Q w całym drzewie, a nie tylko pośród innych akcji z danego stanu
4. e^{Q_i}

We wstępnych testach najlepsze wyniki dało podejście z punktu 1, wszystkie kolejne eksperymenty były prowadzone właśnie z tym wzorem.

Istotne dla skuteczności tego podejścia jest odpowiedni wybór innego parametru *Mixed-UCT*: tego jak w jakie drzewo UCT będzie przekazywane jako drzewo startowe do kolejnej iteracji. Rozpoczynanie każdej iteracji od pustego drzewa spowodowałoby utratę wszystkich informacji z poprzednich iteracji, co skutkowałoby bardzo słabymi strategiami. W rozdziale dotyczącym możliwych sposobów tworzenia drzew dla kolejnych iteracji znajduje się informacja o tym, które sposoby zostały wybrane do użycia z tą metoda generowania strategii.

Ekstrakcja najlepszej ścieżki z każdego z drzew. To podejście jest koncepcyjnie najprostsze ze wszystkich trzech. W ramach tego podejścia wykorzystywana jest dodatkowa zmienna globalna — wektor najlepszych ścieżek znalezionych w kolejnych iteracjach. W tej metodzie ekstrakcji z drzewa UCT w każdej iteracji zostaje wybrana jedna, najlepsza ścieżka i dodawana do wektora najlepszych ścieżek znalezionych w kolejnych iteracjach. Każda ścieżka definiuje

wybór ruchu w każdym ze stanów od korzenia do liścia, jest zatem dobrą strategią prostą zredukowaną lidera (dzięki założeniu (†)). Strategia mieszana jest wyznaczana poprzez zliczenie częstotliwości poszczególnych ścieżek w wektorze i ustalenie im prawdopodobieństw proporcjonalnych do liczby wystąpień. Ruchy nie znajdujące się w ogóle w wektorze mają prawdopodobieństwo 0. Dodatkowo, podejście to ma jeden parametr, maksymalną długość wektora: wektor może być albo nieograniczony albo nie dłuższy niż limit ustawiony jako parametr programu. W przypadku ustawienia limitu na wartość l, w momencie, gdy do wektora ma zostać dodana $l+1$ ścieżka, najwcześniej dodana ścieżka jest usunięta, aby długość wektora nie przekroczyła l. Najlepsza ścieżka w drzewie jest wyznaczana w następujący sposób: w korzeniu drzewa wybierany jest ruch o największej wartości licznika odwiedzin. Następnie w stanie wskazanym przez ten ruch ponownie wybierany jest ruch o największej wartości licznika odwiedzin, następuje przejście do stanu wskazywanego przez ten ruch i w każdym kolejnym stanie wybór ruchu o największej wartości licznika odwiedzin jest powtarzany, aż do osiągnięcia węzła terminalnego.

Ekstrakcja wycinków z wielu drzew. Trzeci z proponowanych wariantów zamiany drzewa UCT na strategię łączy w sobie pewne cechy obu podejść. Pierwsze podejście — zamiana całego drzewa na strategię opiera się na przechowywaniu w drzewie zakumulowanej ze wszystkich iteracji informacji na temat ruchów i zamiana tego drzewa na strategię. Drugie podejście, niezależnie od tego co jest akumulowane w drzewie UCT, w każdej iteracji wybiera najlepszą ścieżkę i przechowuje ją poza drzewem UCT — prowadzi własną historię. To podejście rozszerza pomysł wyboru najlepszej ścieżki do wyboru całego wycinka drzewa, w którym ruchy mają wysokie wartości liczników odwiedzin, zbieranie wektora (zmiennej globalnej) takich wycinków i w ramach budowy strategii lidera, sklejania tych wycinków w jedno drzewo, które można przekształcić w strategię stosując metodę z wariantu pierwszego. Ten wariant ekstrakcji strategii jest opisany w Algorytmie 4.3. Pomiędzy kolejnymi wywołaniami ekstrakcjami strategii przechowywany jest wektor wycinków drzewa za poszczególnych iteracji, początkowo pusty (linia 2). Procedura ekstrakcji strategii składa się z wycięcia z drzewa UCT fragmentu zawierającego dobrze ocenione ruchy (linia 4), następnie do wektora wycinków drzew zostanie dodany kolejny wycinek, gdy wycinków jest więcej niż limit podany jako parametr metody, to najstarszy z wycinków jest usuwany z wektora. Po modyfikacji wektora, tak, żeby uwzględniał wycinek z ostatniej iteracji następuje połączenie wszystkich wycinków w jedno drzewo, a następnie zastosowanie metody transformacji drzewa w strategię z pierwszego wariantu metody. Wynikowa strategia jest zwracana jako rezultat metody.

W linii 4 jest wybierany najlepszy wycinek drzewa. Procedura wyboru wycinka przebiega w następujący sposób. Parametrem jest współczynnik p z przedziału $(0,1]$ mówiący jak dużą część drzewa należy zachować. Dla każdego węzła w drzewie z osobna wykonywana, idąc od góry drzewa jest następująca operacja: niech c będzie liczbą ruchów z danego stanu znajdujących się w drzewie UCT. Wylicz $r = \lceil p \cdot c \rceil$. Posortuj ruchy malejąco według średnich wypłat i wybierz r najlepszych. Pozostałych nie umieszczaj w wycinku i nie przetwarzaj ich następników.

Algorytm 4.3: Ekstrakcja strategii z użyciem wielu wycinków drzew UCT

```
1  sliceLimit// parametr -- maksymalna liczba wycinków w
       wektorze
2  treeSlices ← []// pusty wektor
3  def extractLeaderStrategy(uctTree):
4  |   slice ← getSlice(uctTree);
5  |   if length(treeSlices) = sliceLimit then
6  |   |   treeSlices ← deleteFirst(treeSlices);
7  |   end
8  |   treeSlices ← appendToEnd(treeSlices, slice);
9  |   tree ← mergeSlices(treeSlices);
10 |   return strategy(tree);
```

W linii 9 zebrane wycinki drzewa są składane w całość. Procedura przebiega w następujący sposób: wynikowe drzewo zawiera wszystkie stany i wszystkie ruchy, które występują w co najmniej jednym wycinku. Licznik odwiedzin ruchu jest sumą liczników odwiedzin ze wszystkich wycinków dla tego ruchu. Wartość średniej wypłaty jest uśrednieniem wypłaty po wszystkich ruchach. *Uwaga: w związku z tym, że wybrana procedura zamiany drzewa w strategię nie wykorzystuje średniej wypłaty, sposób wyliczenia średniej wypłaty po połączeniu nie ma znaczenia.*

Sposób przenoszenia drzewa do następnej iteracji

Drugim elementem Algorytmu 4.2, który jest parametrem i może być zaimplementowany w różny sposób, jest krok przenoszenia drzewa UCT między iteracjami (linia 13). Użyty wariant procedury zależy od tego, który sposób zamiany drzewa UCT w strategię był użyty. Metody zamiany drzewa w strategię mają istotnie różną charakterystykę przechowywania informacji z poprzednich iteracji. Metoda wyboru najlepszej ścieżki oraz metoda wycinków wielu drzew prowadzą w zmiennej globalnej historię odpowiednio ścieżek i wycinków drzew z poprzednich iteracji oraz uwzględniają te informacje przy budowie strategii. Metoda zamiany drzewa na strategię nie gromadzi żadnej historii. To spostrzeżenie oznacza, że w przypadku metod, które prowadzą historię można zastosować bardzo proste podejście do przenoszenia drzewa UCT do kolejnej iteracji, polegające na porzucaniu wszystkich statystyk i rozpoczynaniu każdej iteracji od pustego drzewa (nieprzenoszenie żadnej informacji). Taka implementacja została też użyta w tych dwóch przypadkach. W przypadku transformacji drzewa w strategię takie podejście nie jest właściwe, bo utracona zostałaby informacja z poprzednich iteracji. W związku z tym, dla tej metody, stosowany jest inny sposób przenoszenia drzewa UCT pomiędzy iteracjami *Mixed-UCT*. W tym przypadku w kolejnej iteracji metody wykorzystywane jest drzewo z poprzedniej iteracji, co znaczy, że w kolejnych iteracjach liczniki odwiedzin są coraz wyższe. Warto też zwrócić uwagę na fakt, że z iteracji na iterację zmienia się strategia naśladowcy, co oznacza, że w fazie selekcji algorytmu UCT decyzje są podejmowane na podstawie zakumulowanych średnich wypłat z poprzednich iteracji, które mogą nie odpowiadać wypłatom przeciwko bieżą-

cej strategii naśladowcy. Wraz z kolejnymi aktualizacjami wartości tych wypłat przybliżają się jednak do wartości odpowiednich dla strategii naśladowcy z tej iteracji.

Sposób aktualizacji strategii naśladowcy

Ostatnim elementem, który w metodzie *Mixed-UCT* jest możliwy do ustawienia jest sposób w jaki sposób jest uzyskiwana strategia naśladowcy, która będzie użyta w kolejnej iteracji metody. W tym celu utrzymywany jest wektor najlepszych odpowiedzi naśladowcy na strategie lidera δ_l wyliczone w kolejnych iteracjach *Mixed-UCT*. Po każdym wyznaczeniu strategii lidera wyznaczane są *wszystkie* strategie proste naśladowcy, które są najlepszą odpowiedzią na strategię δ_l lidera. Wszystkie znalezione strategie są zamieniane na strategię mieszaną poprzez ustanowienie jednostajnego rozkładu nad tymi strategiami, a ta strategia mieszana dodawana jest do wektora odpowiedzi naśladowcy. Wektor strategii naśladowcy przechowuje l_F strategii. Jeśli dodawana do wektora strategia spowoduje przekroczenie rozmiaru wektora, to najstarsza ze strategii jest usuwana z tego wektora. W następnym kroku wyznaczana jest strategia mieszana, która powstaje poprzez uśrednienie strategii mieszanych w wektorze i ta strategia jest zwracana jako strategia naśladowcy dla kolejnej iteracji *Mixed-UCT*. Poprzez uśrednienie strategii mieszanych rozumie się tu następującą operację: dla każdej strategii prostej, która wchodzi w skład przynajmniej jednej strategii mieszanej w wektorze, zsumuj prawdopodobieństwa jej zagrania we wszystkich strategiach mieszanych, w których występuje, a następnie podziel przez długość wektora. Tak wyliczona liczba jest prawdopodobieństwem zagrania tej strategii prostej w ramach uśrednionej strategii mieszanej.

Warunek stopu

Główna pętla *MixedUCT*, w linia 4 w Algorytmie 4.2 jest powtarzana aż do osiągnięcia zadanego warunku stopu. Co do zasady, warunek można formułować dowolnie, jednak we wszystkich eksperymentach przyjęto alternatywę dwóch warunków:

- liczba iteracji głównej pętli osiągnęła limit będący parametrem metody lub
- w przeciągu ostatnich M iteracji nie nastąpiła poprawa najlepszej znanej strategii (δ_l^*).

Implementacja

Metoda *Mixed-UCT* została zaimplementowana w językach Java i Scala działających na platformie JVM. Jako środowisko uruchomieniowe była wykorzystywana maszyna wirtualna Java w wersji 8. Repozytorium z kodem źródłowym, zawierające zarówno metodę *Mixed-UCT*, jak i opisaną dalej *O2UCT* jest dostępne pod adresem internetowym `https://gitlab.com/jan-karwowski/petrus`. Wspomniane repozytorium zawiera również kod związanymi z innymi, powiązanymi eksperymentami, którego autorami są Adam Żychowski i Filip Grajek, jednak kod implementujący wspomniane metody jest w całości wykonany przez autora tej

rozprawy. Szczegóły dotyczące autorstwa kodu źródłowego można uzyskać za pomocą mechanizmu `git blame`.

Wstępne eksperymenty

Zaprezentowana powyżej metoda *Mixed-UCT* zawiera podprocedury, które są parametrami metody i mogą być zamieniane na inne oraz szereg mniejszych parametrów, takich jak długości poszczególnych wektorów, parametr C we wzorze (4.1), czy warunek stopu metody. Ważnym etapem wstępnych prac nad metodą był wybór parametrów. Wybór ten przeprowadzono poprzez uruchomienie metody na zbiorze gier *Basic* opisanym w Rozdziale 4.1.1. Zostały przeprowadzone dwa wstępne eksperymenty: jeden mający na celu ustalenie który z wariantów zamiany drzewa UCT na strategię będzie najlepszy, opublikowany w pracy [41] oraz mający ustalić właściwą długość wektora przechowującego najlepsze odpowiedzi naśladowcy, opublikowany w pracy [42].

Oba eksperymenty wstępne zostały przeprowadzone w systemie Debian GNU/Linux na komputerze z procesorem Intel Core i7 CPU @ 3.40GHz.

Wybór metody budowy strategii na podstawie drzewa UCT. Powyżej zaproponowano trzy warianty metody zamiany statystyk z drzewa UCT na strategię mieszaną lidera. Aby stwierdzić, który z wariantów najlepiej zastosować w dalszych badaniach, przeprowadzono następujący eksperyment: dla każdej z gier ze zbioru *Basic* z limitem rund $T = 5$ uruchomiono poszukiwanie strategii lidera 10 razy. Dla każdej ze znalezionych strategii obliczono wypłatę lidera przeciwko najlepszej odpowiedzi naśladowcy na tę znalezioną strategię, a następnie wyniki uśredniono w obrębie każdej z gier. W celu pomocy prezentacji wyników, policzona została również wypłata jaką może otrzymać lider grając strategię jednostajną przeciwko odpowiedniej optymalnej odpowiedzi naśladowcy. Na koniec zastosowano przytoczoną wcześniej metodę *DOBSS* [69] do znalezienia dokładnej Równowagi Stackelberga. Na podstawie tych wartości, dla każdego wyznaczono współczynnik oceny wyniku gry w następujący sposób:

$$sc = \frac{u - u_{uniform}}{u_{optimal} - u_{uniform}}, \tag{4.2}$$

gdzie u — oceniana wypłata lidera, $u_{uniform}$ — wypłata lidera uzyskana dla jednostajnej strategii lidera, $u_{optimal}$ — wypłata lidera w stanie Równowagi Stackelberga. Łatwo jest zaobserwować, że $sc = 1$ oznacza optymalny wynik, a $sc = 0$ wynik taki, jak dla strategii jednostajnej. Formalnie możliwe jest otrzymanie wyniku $sc < 0$, jednak znaczyłoby to, że uzyskana strategia jest gorsza nawet od strategii jednostajnej.

Wszystkie przebiegi *Mixed-UCT* zostały uruchomione z parametrami przedstawionymi w Tabeli 4.4. W Tabeli 4.5 przedstawione zostały wyniki tego eksperymentu. Wartości miary sc dla metody PD wynoszą 1 dla każdej z gier testowych, w przypadku metody WD sc wynosi 1 dla 8 z 12 przebiegów. Dla pozostałych przebiegów wartość ta jest co najmniej 0.96. W przypadku

Tabela 4.4: Zestaw parametrów *Mixed-UCT* użyty we wstępnych eksperymentach.

Parametr	Wartość
Współczynnik eksploracji (C) we wzorze UCB1 (4.1)	1.4
Liczba symulacji w każdym węźle będącym korzeniem z którego startujemy I2UCT ($simCount$)	500
Maksymalna liczba iteracji głównej pętli *Mixed-UCT*	45 000
Liczba iteracji bez poprawy powodująca wczesne zatrzymanie	10 000
Długość wektora przechowującego historyczne strategie naśladowcy (l_F)	500
Limit wysokości prawdopodobieństwa poniżej którego ruchy są usuwane ze strategii lidera w procedurze transformacji drzewa UCT w strategię	0.05

Tabela 4.5: Wyniki wstępnego eksperymentu mającego na celu wybór sposobu zamiany drzewa UCT w strategię. Dla każdego wariantu metody i dla każdej prezentowany jest średni czas wykonania w sekundach oraz miara sc zdefiniowana wzorem (4.2). Dodatkowo, jako wynik referencyjny, prezentowany jest czas solwera DOBSS, który oblicza dokładną strategię.

	Pełne drzewo (PD)		Wiele wycinków (WD)		Najlepsze ścieżki (NS)		DOBSS	
gra	czas [s]	*sc*	czas [s]	*sc*	czas[s]	*sc*	*sc*	czas [s]
game1	2382	1	1559	0.96	1344	0.99	28327	1
game2	2614	1	2117	0.99	1587	1	110	1
game3	3188	1	7643	0.99	2486	0.97	17394	1
game3a	2115	1	3191	1	1379	1	18734	1
game3b	2325	1	3359	1	2484	1	283	1
game3c	2058	1	3183	1	2151	1	18579	1
game3d	2241	1	4062	1	2897	1	4695	1
game4	1973	1	2364	1	1579	1	5823	1
game4a	2058	1	2007	1	1261	1	5915	1
game4b	1393	1	1448	0.99	1585	0.99	5519	1
game4c	2579	1	2416	1	2093	1	5928	1
game4d	1713	1	2686	1	1775	1	5149	1
średnia	2220	1	3003	1	1885	1	9705	1

metody NS dla 9 gier testowych sc ma wartość 1, a dla pozostałych 3 przypadków wartość ta jest co najmniej 0.97. Wartości sc pokazują, że wszystkie warianty metody dają strategie lidera, które są bardzo bliskie strategii lidera ze stanu Równowagi Stackelberga. W tym aspekcie metoda PD jest nieznacznie lepsza od pozostałych. Drugim mierzonym aspektem był czasy wykonania metody *Mixed-UCT* w poszczególnych wariatach. Tu zróżnicowanie jest większe, uśredniony po wszystkich grach testowych czas wykonania metody był najniższy dla wariantu NS (1885 sekund), drugi w kolejności był wariant PD (2220 sekund), a trzeci, istotnie wolniejszy był wariant WD. Wszystkie trzy warianty metody są istotnie szybsze od metody dokładnej — DOBSS, jednak różnice pomiędzy wariantami są znaczące. Najszybszy (NS) jest o prawie 40% szybszy od najwolniejszego (WD). Celem eksperymentu był wybór jednego z wariantów, który zostanie wykorzystany w dalszych eksperymentach. Metoda wykorzystująca WD okazała się najgorsza zarówno jeśli chodzi o czas wykonania, jak i jakość wyników, co pozostawia wybór pomiędzy PD i NS. O ile PD dawała nieznacznie lepsze wyniki, to odbywało się to kosztem istotnie większego nakładu czasowego. W związku z tym, mając na uwadze fakt, że w dalszych eksperymentach metoda będzie testowana między innymi pod kątem skalowalności czasowej dla dużych gier, do dalszych eksperymentów został wybrany wariant NS.

Dobór długości wektora przechowującego historyczne strategie naśladowcy. Drugi z eksperymentów wstępnych miał na celu ustalenie jaki limit l_F długości wektora historycznych najlepszych odpowiedzi naśladowcy da najlepsze rezultaty (pod względem czasowym i pod względem jakości strategii). Eksperyment był wykonywany po eksperymencie opisywanym wcześniej i w związku z tym został przeprowadzony z użyciem metody NS (najlepszej ścieżki) do zamiany drzewa UCT w strategię mieszaną. Wyniki tego eksperymentu zostały pierwotnie opublikowane w pracy [42]. Parametry metody w eksperymencie były takie same jak w przypadku poprzedniego eksperymentu, opisane w Tabeli 4.4, z wyjątkiem parametru l_F. Eksperyment polegał na uruchomieniu metody *Mixed-UCT* z wartościami parametru $l_F \in \{1{,}100{,}500{,}1000\}$ i porównaniu wyników między sobą. Eksperyment, podobnie jak poprzedni, został przeprowadzony na zbiorze *Basic*. Dla każdej gry ze zbioru i dla każdej wartości l_F zostało przeprowadzone 10 przebiegów *Mixed-UCT*, a wyniki prezentowane w Tabeli 4.6 przedstawiają uśredniony wynik ze wszystkich przebiegów. Ponownie została wykorzystana miara sc zdefiniowana wzorem (4.2) i ponownie wyniki zostały zestawione z metodą DOBSS. Pierwszą rzeczą, która rzuca się w oczy przy analizie wyników, jest fakt, że dla $l_F = 1$ znalezione strategie lidera są dużo słabsze niż dla pozostałych wartości tego parametru. Średnia wartość sc dla tego przypadku wynosi 0.97. Dla wszystkich pozostałych przypadków średnia wartość wyniosła 1 (zaokrąglona do dwóch miejsc po przecinku). Ta obserwacja jest podstawą do odrzucenia wartości $l_F = 1$ jako kandydata na wartość parametru l_F. W przypadku pozostałych wartości jakość otrzymanych strategii jest równie dobra, różni się jednak czas obliczeń. Średni czas obliczeń dla wartości 100 wyniósł 1499 sekund, dla wartości 500 1359 sekund, a dla wartości 1000 – 1289 sekund. Można zaobserwować nieznaczny spadek średniego czasu obliczeń

Tabela 4.6: Wyniki wstępnego eksperymentu, który miał na celu wyznaczenie optymalnej wartości parametru l_F — długości wektora przechowującego optymalne odpowiedzi naśladowcy, który służy do wyliczenia strategii mieszanej naśladowcy w metodzie *Mixed-UCT*

game	1		100		500		1000		DOBSS	
	czas [s]	sc	czas [s]	sc	czas [s]	sc	czas [s]	sc	czas[s]	sc
game1	1448	0.81	1492	0.99	1592	0.99	1611	0.99	28327	1
game2	1049	0.99	1113	1	1333	1	1351	1	110	1
game3	2319	0.99	1786	1	1332	1	1135	1	17394	1
game3a	2003	1	1230	1	1358	1	1487	1	18734	1
game3b	1838	0.96	1382	1	1823	1	1639	1	283	1
game3c	1777	0.94	1358	1	1286	1	1169	1	18579	1
game3d	2129	0.99	1467	1	1186	1	1218	1	4695	1
game4	1777	1	1995	1	1151	1	1310	1	5823	1
game4a	1681	1	1637	1	1203	1	1114	1	5915	1
game4b	1788	1	1639	1	1312	1	1061	1	5519	1
game4c	1783	1	1494	1	1374	1	1203	1	5928	1
game4d	1948	1	1391	1	1356	1	1165	1	5149	1
średnia	1795	0.97	1499	1	1359	1	1289	1	9705	1

dla coraz większych wartości parametru. Jednocześnie należy podkreślić, że spadek czasu nie jest obserwowany we wszystkich grach. W przypadku gier *game1* i *game2* obserwujemy odwrotną prawidłowość – dla mniejszych wartości parametru uzyskiwane sa krótsze czasy. Mimo to ogólny wynik sugeruje, że wartość 1000 dała najlepszy rezultat i w związku z tym ta wartość została wybrana do dalszych eksperymentów.

Krótkiego komentarza wymaga fakt, że użycie dłuższego, zajmującego więcej pamięci, wektora spowodowało skrócenie czasu obliczeń. Warto zauważyć, że wektory implementowane są z użyciem tablic, w związku z czym długość wektora nie ma wpływu na szybkość indeksowania. Po drugie, ze względu na warunek wczesnego zatrzymania — metoda *Mixed-UCT* zatrzymuje się po wykonaniu odpowiednio dużej liczby iteracji, które nie przyniosły poprawy strategii lidera. W przypadku wyższych wartości testowanego parametru szybciej osiągana jest dobra strategia lidera, która nie jest już poprawiana i dzięki temu cały przebieg jest krótszy.

4.2.4 Wyniki eksperymentalne

Przedstawione wyżej eksperymenty miały na celu tylko dobór parametrów do metody. Co prawda wyniki w tabelach 4.5 i 4.6 są zestawione z metodą obliczającą dokładnie stan Równowagi Stackelberga DOBSS, jednak zbiór testowy był zbyt ograniczony, żeby można było twierdzić, że metoda *Mixed-UCT* jest bardziej wydajnym podejściem do obliczania Równowagi Stackelberga. Ponadto, w momencie prowadzenia eksperymentu wstępnego autorowi rozprawy nie były jeszcze znane szybsze niż DOBSS metody obliczania równowagi Stackelberga w grach sekwencyjnych (najstarsza z metod opisanych w Rozdziale 3.7 to *BC2015*, opubli-

kowana w roku 2015, podobnie pierwszy eksperyment wstępny, opublikowany w 2016 roku, był prowadzony w roku 2015, bardzo krótko po publikacji wspomnianej metody). W związku z tym, żeby ocenić faktyczną przydatność metody *Mixed-UCT* konieczne było przeprowadzenie eksperymentów na większej liczbie gier i zestawienie wyników z metodą szybszą niż DOBSS.

Eksperymenty zostały przeprowadzone na zbiorze *WHG*, dla gier o liczbie rund od $T = 3$ do $T = 9$ (eksperymenty wstępne były prowadzone wyłącznie dla $T = 5$). Wyniki dla metody *Mixed-UCT* są zestawione z metodą *DOBSS* (użytą już w eksperymentach wcześniejszych) oraz z metodą *BC2015* (opisaną dokładnie w ramach przeglądu literatury w Rozdziale 3.7.1). Porównany jest czas potrzebny do wyznaczenia strategii lidera oraz jakość znalezionej strategii rozumiana przez wypłatę lidera w sytuacji, gdy lider gra znaleziona strategię przeciwko najlepszej odpowiedzi naśladowcy dla tej znalezionej strategii.

Środowisko uruchomieniowe. Wszystkie eksperymenty były uruchomione na maszynie z procesorem Intel Core i7 CPU @3.40GHz and 16GB RAM działającej pod kontrolą systemu Debian GNU/Linux. W przypadku metod *DOBSS* oraz *BC2015*, które budują program liniowy, którego rozwiązanie definiuje strategie lidera, został użyty solwer programowania liniowego Gurobi 8.0 [34]. W przypadku *Mixed-UCT* ponownie była użyta maszyna wirtualna Java w wersji 8.0. W przypadku metody *Mixed-UCT*, ze względu na jej losową naturę, wykonano 10 przebiegów dla każdej z gier testowych, a wyniki uśredniono. W przypadku metod wykorzystujących programowanie liniowe, wykonany jedno powtórzenie każdego przebiegu.

Parametry metody *Mixed-UCT* są zgodne z parametrami wyznaczonymi we wstępnych eksperymentach, wybrana metoda zamiany drzewa w strategię to wybór najlepszej ścieżki (NS), długość l_F wektora zawierającego historyczne strategie naśladowcy wynosiła 1000, pozostałe parametry zgodnie z wcześniej przedstawioną Tabelą 4.4. Metody *DOBSS* i *BC2015* nie mają żadnych parametrów.

Wyniki eksperymentów. W związku z dużo szerszym zakresem gier testowych niż w poprzednich eksperymentach, 25 gier testowych w dla liczby rund od 3 do 9, eksperymenty nie będą zaprezentowane równie szczegółowo, jak eksperymenty wstępne. W przypadku metod opartych o programowanie liniowe nie było możliwe policzenie wszystkich gier testowych. W przypadku metody *DOBSS* gry mające $T = 6$ lub więcej rund wymagały zbudowania programu liniowego, którego rozwiązanie nie było możliwe w dostępnym limicie pamięci operacyjnej, w przypadku metody *BC2015* programy liniowe były mniejsze, ale nadal niemożliwe było obliczenie strategii dla $T = 7$ i większych. Tabela 4.7 przedstawia czasy obliczeń zagregowane po liczbie rund T, oprócz tego, te same dane są zaprezentowane w formie wykresu na Rysunku 4.7. Te wyniki pierwotnie zostały opublikowane w pracy [43].

Rysunek 4.7, poza punktami reprezentującymi średnie czasy obliczeń dla poszczególnych długości gier, prezentuje krzywe dopasowane do tych punktów, które są pozwalają ekstrapolować czas obliczeń dla większych długości gier, dla których nie udało się obliczyć warto-

Tabela 4.7: Czasy wykonania metody *Mixed-UCT* uśrednione po wszystkich gier o danej liczbie rund $T = 3, \ldots, 9$, na tle metod budujących programy liniowe — DOBSS i BC2015. Czasy tych dwu metod zaprezentowane tylko dla zakresu T, dla których było możliwe policzenie wartości w ramach dostępnej pamięci operacyjnej.

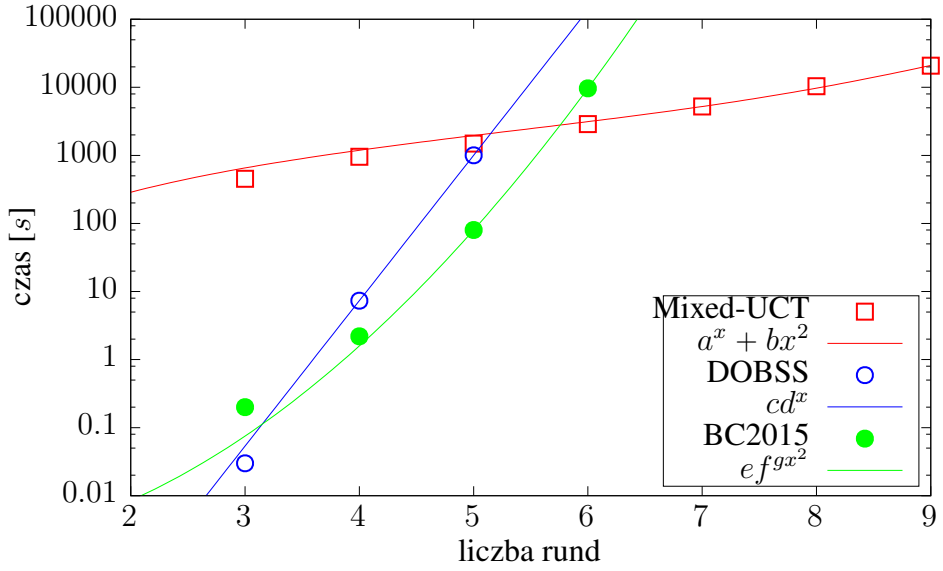
liczba rund (T)	3	4	5	6	7	8	9
Mixed-UCT: czas [s]	453	955	2240	2908	5233	10360	20827
DOBSS: czas [s]	0.03	7.33	994	—	—	—	—
BC2015: czas [s]	0.20	2.20	80	9654	—	—	—

Tabela 4.8: Współczynniki funkcji aproksymujących czas wykonania metod przedstawionych w Tabeli 4.7 dopasowane z użyciem metody najmniejszych kwadratów.

Mixed-UCT $a^x + bx^2$	DOBSS de^x	BC2015 ef^{gx^2}
$a = 2.9$	$d = 3.4e-6$	$e = 1.4e-3$
$b = 74$	$e = 49$	$f = 1.5$
		$g = 1.1$

ści. Krzywe aproksymujące czasy zostały uzyskane w następujący sposób: na podstawie analizy pseudokodu poszczególnych metod dobrane zostały funkcje z parametrami. Odpowiednio: $a^x + bx^2$ dla *Mixed-UCT*, ze względu na to, że poszukiwanie odpowiedzi lidera jest realizowane poprzez pełny przegląd strategii lidera i zajmuje liniowo dużo czasu względem tej liczby. Liczba możliwych strategii lidera z kolei rośnie wykładniczo względem liczby rund w grze. Stąd pierwszy składnik, który jest wykładniczy. Drugi składnik to pozostała część metody *Mixed-UCT*, związana z algorytmem *I2-UCT*, w którym symulacja od korzenia do liścia zajmuje liniowo wiele czasu, a liczba symulacji, ze względu na pętlę w linii 3 Algorytmu 4.1 jest liniowa względem długości gry, co daje złożoność kwadratową. W przypadku metody *DOBSS* wybrano funkcję cd^x, gdzie c,d to parametry, które będą dopasowywane. Program liniowy tworzony przez tę metodę ma liczbę kolumn proporcjonalną do liczby strategii naśladowcy oraz liczbę wierszy proporcjonalną do liczby strategii lidera, każdy z tych zbiorów strategii ma wykładniczo wiele elementów względem liczby. W przypadku metody *BC2105* wybrano funkcję ef^{gx^2}, gdzie e,f,g to parametry. Znaczenie ma przede wszystkim rozmiar sekwencyjnej reprezentacji gry, który jest liniowy względem liczby węzłów w drzewie gry. Ta z kolei jest wykładnicza względem liczby kroków w grze. Wszystkie wybrane funkcje mają 2 lub 3 parametry. Za pomocą metody najmniejszych kwadratów Levenberga-Marquardta [56] zaimplementowanej w pakiecie Gnuplot wartości współczynników funkcji zostały dopasowane do odpowiadających punktów z Tabeli 4.7. Uzyskane wartości współczynników są przedstawione w Tabeli 4.8.

Patrząc na średnie czasy obliczeń przedstawione w Tabeli 4.7 i na Rysunku 4.7 widać, że dla małych gier metoda Mixed-UCT jest dużo wolniejsza od metod opartych o programowanie liniowe. Jednak wraz ze wzrostem rozmiaru gier czas wykonania *Mixed-UCT* skaluje się



Rysunek 4.7: Czasy wykonania metody *Mixed-UCT* uśrednione po wszystkich gier o danej liczbie rund $T = 3, \ldots, 9$, na tle metod budujących programy liniowe — DOBSS i BC2015. Czasy tych dwu metod zaprezentowane tylko dla zakresu T, dla których było możliwe policzenie wartości w ramach dostępnej pamięci operacyjnej. Dane z Tabeli 4.7 z dodaną krzywą aproksymującą dalszą skalowalność czasową.

dużo lepiej. Dla gier o $T = 6$ metoda *Mixed-UCT* jest szybsza od pozostałych metod. Co więcej krzywe ekstrapolujące czas wykonania dla większych gier nie mieszczą się na wykresie pomimo zastosowania skali logarytmicznej. W przypadku gier o $T = 7{,}8{,}9$ różnica w czasie wykonania pomiędzy *Mixed-UCT*, a metodami opartymi o program liniowy wyniosłaby od kilku do kilkunastu rzędów wielkości. W sytuacji, gdy $T = 9$ przewidywany czas obliczeń metod *DOBSS* i *BC2105* byłby tak duży, że nie jest możliwe praktyczne zastosowanie tych metod, metoda *Mixed-UCT* jest nadal w stanie rozwiązać te gry w czasie liczonym w dniach.

Równie ważne, jeśli nie ważniejsze niż szybkość obliczeń, jest to, czy metoda *Mixed-UCT* oblicza strategie, które dają liderowi dobrą wypłatę. Gdyby tak nie było, to cały zysk z szybszych obliczeń byłby pozorny. Problemem w badaniu jakości znalezionej strategii jest fakt, że nie są znane rozwiązania dla wszystkich gier. Dokładny stan równowagi Stackelberga udało się policzyć, z użyciem metody *BC2015* dla gier o $T \leqslant 6$. W związku z tym porównanie jakości strategii lidera możliwe jest tylko dla tych gier. Ponadto w przypadku gier o $T = 3{,}4$ problem bardzo często miał trywialne rozwiązanie, strategia jednostajna lidera dawała dokładnie taką samą wypłatę, jak strategia ze stanu równowagi Stackelberga. Wynika to z faktu, że w wielu przypadkach 3 rundy nie były wystarczające, żeby którykolwiek z graczy miał szansę wykonać akcję, która dawała inną wypłatę końcową niż 0. Powyższe problemy spowodowały, że analiza jakości znalezionych strategii została przeprowadzona tylko dla gier o liczbie rund $T = 5$ i $T = 6$. Wyniki szczegółowe dla gier $T = 5$ są przedstawione w Tabeli 4.9, a dla gier o $T = 6$ w Tabeli 4.10. W obu tabelach prezentowana jest miara sc zdefiniowana wzorem (4.2) w dwóch wariantach, dla średniej wypłaty strategii lidera z 10 przebiegów metody (oznaczona jako sc)

oraz dla maksymalnej wartości tej wypłaty (sc_{max}). Wartość miary sc dla strategii obliczonej pozostałymi metodami jest pominięta, ponieważ obliczają one dokładny stan równowagi, a co za tym idzie wartość sc wynosi 1. Obok wartości sc prezentowane są czasy obliczeń dla każdej z gier testowych dla metod *Mixed-UCT* i *BC2015*. Czas obliczeń metody *DOBSS* został pominięty, ponieważ, co pokazano w Tabeli 4.7, dla gier o $T \geqslant 4$ jest on wyraźnie gorszy od pozostałych metod. Analizując wartości sc w Tabeli 4.9, można zaobserwować, że dla większości gier wskaźnik sc jest powyżej $0{,}9$, co oznacza, że jakość znalezionej strategii jest dobra. Są dwie instancje testowe, dla których ten wynik jest istotnie gorszy: gra 24, gdzie wynik to $0{,}71$ oraz gra 56, gdzie wynik to tylko $0{,}56$. Mimo tych dwóch przypadków, średnia wartość miary sc to $0{,}95$. Dodatkowo, patrząc na wartość sc_{max} można zauważyć, że w 14 przypadkach wartość ta jest równa 1, a tylko w jednym, gra 56, jest poniżej $0{,}98$. Oznacza, to że wyniki metody *Mixed-UCT* można poprawić poprzez kilkukrotne uruchomienie tej metody, a następnie wybór najlepszego z wyników z tych przebiegów. To, oczywiście, oznacza dodatkowy koszt czasowy. Należy jednak zwrócić uwagę na fakt, że, zakładając 10 powtórzeń, czas obliczeń wzrośnie dziesięciokrotnie, a dla gier o liczbie rund $T \leqslant 7$ wartość ta nadal byłaby dużo mniejsza niż czas obliczeń *BC2015*, co można zaobserwować na Rysunku 4.7. Czasy obserwowane w Tabeli 4.9 są zbieżne z wcześniejszymi obserwacjami w Tabeli 4.7. W przypadku $T = 5$ metoda *Mixed-UCT* potrzebuje średnio 28 razy więcej czasu na obliczenie strategii w tej samej grze. W skrajnych przypadkach, gdy metoda *BC2015* zwraca wynik bardzo szybko (gry 74 i 87) współczynnik ten jest większy, odpowiednio 1890 oraz 485. Tabela 4.10, która przedstawia wyniki dla gier o $T = 6$ rundach układ prezentacji wyników jest identyczny, jak w poprzedniej tabeli. W przypadku wartości sc oraz sc_{max} wartości są nieznacznie niższe niż w grach o $T = 5$. Wartość sc po raz kolejny jest mniejsza od $0{,}9$ tylko w grach 24 i 56, w przypadku wartości sc_{max} ponownie jest wiele wartości $1{,}0$, choć warto zwrócić uwagę, że tym razem są dwie wartości poniżej 0.98, dla gier 24 — $0{,}91$ oraz 56 — $0{,}56$. W pozostałych przypadkach jakość znalezionych strategii jest dobra. Średnie wartości sc i sc_{max} są tylko nieznacznie niższe niż wartości w poprzedniej tabeli. Średni czas obliczeń dla gier o $T = 6$ jest korzystniejszy dla metody *Mixed-UCT*. Należy jednaz zwrócić uwagę na fakt, że w kilku przepadkach testowych *BC2015* jest szybsza od *Mixed-UCT*. Są to gry: 20, 31, 39, 41, 56, 74, 87, 96. Przytoczone 8 przypadków to prawie $1/3$ zbioru testowego. Mimo to, średni czas metody *BC2015* jest ponadtrzykrotnie większy od *Mixed-UCT*. Dzieje się tak dlatego, że w niektórych przypadkach, takich jak gry 42, 82, 85, 91 *BC2015* jest dużo wolniejsza od metody metaheurystycznej. W tym miejscu warto też zauważyć, że czasy wykonania metody *Mixed-UCT* są dużo bardziej przewidywalne, dla wszystkich instancji gier o tej samej długości czasy wykonania są podobne. W przypadku metody *BC2015* czasy są bardzo zróżnicowane pomiędzy instancjami.

Przedstawione wyniki pokazują, że dla dużych gier czas obliczenia strategii przy użyciu metody *Mixed-UCT* jest dużo niższy niż w przypadku *BC2015*, starsza metoda *DOBSS* działa w rozsądnym czasie jedynie dla bardzo małych gier. Analiza jakości uzyskiwanych strategii pokazuje, ze dla większości gier ze zbioru testowego strategia policzona przez *Mixed-UCT*

jest tylko nieznacznie gorsza od dokładnych rozwiązań. Dodatkowo metoda *Mixed-UCT* potrzebuje dużo mniej pamięci niż metody oparte o programowanie liniowe. Metoda *Mixed-UCT* mogła rozwiązać na komputerze z 16GB RAM gry o kilka rzędów wielkości większe niż metody oparte o MILP. Nie można jednak nie powiedzieć to dwóch grach, dla których strategie znajdowane przez *Mixed-UCT* dają gorsze wypłaty niż strategie ze stanu Równowagi Stackelberga. Drugą rzeczą, którą należy podkreślić jest fakt, że eksperymenty zostały przeprowadzone tylko na zbiorze testowym *WHG*, w którym struktura wypłat graczy jest bliska sumie zerowej. W związku z tym nie można stwierdzić, że *Mixed-UCT* nada je się do stosowania w ogólnej klasie gier o sumie niezerowej z niepełną informacją.

4.2.5 Ograniczenia i słabe punkty metody Mixed-UCT

Po przeprowadzeniu eksperymentów przedstawionych powyżej, prace nad metodą *Mixed-UCT* zostały skierowane w dwóch kierunkach: czy da się poprawić wyniki metody na wspomnianych wcześniej grach 24 i 56 oraz czy jest możliwe uzyskanie dobrych wyników na grach dalszych od sumy zerowej. Niestety nie udało się zaproponować takich modyfikacji metody, które pozwoliłyby na uzyskanie dobrych wyników na zbiorze *WNZ*. Nie udało się też poprawić wyników dla dwóch wspomnianych gier. Ponadto głównym celem przewodnim przy budowie *Mixed-UCT* było stwierdzenie, czy jest możliwe zastosowanie wprost podejścia *UCT* do poszukiwania strategii lidera. Niestety nie udało się w pełni zrealizować tego planu, choć, co warto podkreślić, czas znajdowania strategi dla dużych gier ze zbioru *WHG* jest bezkonkurencyjny względem metod opartych o programowanie liniowe.

Tabela 4.9: Szczegółowe wyniki eksperymentów dla gier z $\mathbf{T} = \mathbf{5}$. Pierwsza kolumna podaje numer gry ze zbioru testowego (numery nie są kolejnymi liczbami, co zostało wyjaśnione w Rozdziale 4.1.1 opisującej tworzenie zbioru testowego). Następnie prezentowana jest miara sc liczona wzorem (4.2) dla strategii obliczonych przez *Mixed-UCT*. Podane są dwa warianty: sc – miara obliczona na podstawie średniej wypłaty lidera z 10 powtórzeń przebiegu oraz sc_{max}, która uwzględnia wartość maksymalną. W kolejnych dwóch kolumnach zaprezentowany jest czas obliczeń odpowiednio *Mixed-UCT* i *O2-UCT*. W ostatnim wierszu tabeli znajdują się średnie po wszystkich instancjach prezentowanych w tabeli.

	Mixed-UCT			BC2015
gra	sc	sc_{max}	czas [s]	czas
17	0,99	1,00	1705	64
2	1,00	1,00	5259	95
20	1,00	1,00	2200	63
24	0,71	0,98	2431	16
3	1,00	1,00	2111	69
35	0,98	1,00	1117	25
39	1,00	1,00	1376	32
41	0,98	0,99	1661	35
42	1,00	1,00	2149	425
43	1,00	1,00	1236	29
56	0,56	0,79	4410	37
59	0,99	0,99	1632	110
64	0,99	1,00	1793	30
7	1,00	1,00	1744	163
74	0,95	0,99	3379	2
78	0,98	1,00	1380	88
82	1,00	1,00	1820	197
85	0,90	0,99	3833	138
87	0,99	0,99	3396	7
89	0,99	1,00	1151	14
91	0,91	1,00	2096	101
96	0,98	0,98	1409	14
średnia	0,95	0,99	2240	80

Tabela 4.10: Szczegółowe wyniki eksperymentów dla gier z $\mathbf{T} = \mathbf{6}$. Pierwsza kolumna podaje numer gry ze zbioru testowego (numery nie są kolejnymi liczbami, co zostało wyjaśnione w Rodziale 4.1.1 opisującej tworzenie zbioru testowego). Następnie prezentowana jest miara sc liczona wzorem (4.2) dla strategii obliczonych przez *Mixed-UCT*. Podane są dwa warianty: sc — miara obliczona na podstawie średniej wypłaty lidera z 10 powtórzeń przebiegu oraz sc_{max}, która uwzględnia wartość maksymalną. W kolejnych dwóch kolumnach zaprezentowany jest czas obliczeń odpowiednio *Mixed-UCT* i *BC-2015*. W ostatnim wierszu tabeli znajdują się średnie po wszystkich instancjach prezentowanych w tabeli.

	Mixed-UCT			BC2015
gra	sc	sc_{max}	czas [s]	czas
17	0,99	0,99	2238	4134
2	1,00	1,00	5761	7109
20	1,00	1,00	3722	2287
24	0,70	0,91	4087	28944
3	0,99	1,00	3304	7018
31	1,00	1,00	3299	632
35	0,96	0,99	1972	3734
39	0,99	0,99	2019	1941
41	0,97	0,98	2846	1938
42	0,90	1,00	1736	71573
43	1,00	1,00	1906	4118
56	0,42	0,56	4357	3821
59	0,99	0,99	1954	11738
64	0,98	0,98	2433	2436
7	1,00	1,00	3631	6590
74	1,00	1,00	2396	1018
78	0,96	1,00	2791	5779
82	0,99	0,99	3142	31122
85	0,91	1,00	2169	11217
87	0,95	1,00	4943	479
89	0,98	1,00	2078	1417
91	1,00	1,00	1741	12149
96	1,00	1,00	2360	849
średnia	0,94	0,97	2908	9654

4.3 Metoda O2-UCT: aproksymacja wykorzystująca naprzemienne próbkowanie strategii lidera i naśladowcy

Podejście, które miało w sposób bezpośredni sposób wykorzystywać algorytm UCT, czyli opisana w poprzednim rozdziale metoda *Mixed-UCT* zakończyło się częściowym sukcesem. W pewnych kasach gier metoda sprawdziła się, jednak nie jest ona uniwersalnie skuteczna. Silnym założeniem ograniczającym swobodę projektowania metody *Mixed-UCT* był warunek, że to podejście ma bezpośrednio wykorzystywać algorytm UCT. Wszystkie dalsze elementy, na przykład konieczność zaproponowania metody zamieniającej drzewo UCT na strategię, czy konieczność budowy wypadkowej odpowiedzi naśladowcy, były podyktowane właśnie budową metody w okół centralnego punktu, którym z założenia była metoda UCT. Dodatkowo tak zbudowane podejście jest trudne do analizy teoretycznej i przeprowadzenia jakichkolwiek dowodów zbieżności. W związku z zaobserwowanymi ograniczeniami, budując drugą metodę do poszukiwania równowagi zostało zastosowane istotnie inne podejście. Tym razem powstał szkielet algorytmu, który jest inspirowany elementami podejść do poszukiwania Równowagi Stackelberga z literatury oznaczonymi symbolami TO1–TO4 na stronie 95. Szkielet ten uzupełnia się następnie implementacjami poszczególnych elementów metody i część z tych elementów może, (i w wersji prezentowanej w tej rozprawie jest), być zaimplementowana z użyciem podejścia UCT.

Metoda opisywana w tym rozdziale nosi nazwę *O2-UCT*, co jest skrótem od *Double Oracle*, podejścia do poszukiwania stanów równowagi w grach opisanego w Sekcji 3.6 oraz *UCT*, metody poszukiwania najlepszego ruchu w grach z pełną informacją, opisanego w Sekcji 4.2.1. Całość metody łączy w sobie elementy z obu tych podejść i służy do znajdowania strategii lidera, która będzie dawać możliwie dużą wypłatę w grach Stackelberga. W przeciwieństwie do *Mixed-UCT*, ta metoda nie nakłada dodatkowych ograniczeń na strukturę zbiorów informacyjnych ponad wymaganiem doskonałej pamięci. Wyniki badań nad metodą *O2-UCT* zostały pierwotnie opublikowane w pracach [45] i [44].

Praktycznie wszystkie podejścia do Równowagi Stackelberga z literatury, zarówno ogólne, opisane z Rozdziale 3.7 oraz metody działające dla specyficznych klas gier, jak na przykład metoda opisana w Rozdziale 3.3, zawierają element, w którym ustalana jest pewna strategia naśladowcy i dla niej poszukiwana jest strategia lidera, dla której ta ustalona strategia naśladowcy jest optymalną odpowiedzią. W metodach tego typu następuje przegląd strategii naśladowcy i krok poszukiwania strategii lidera dla każdej przeglądanej strategii naśladowcy. To, w jaki sposób przeglądane są strategie naśladowcy, zależy od metody. Metody ogólne zwykle wykorzystują albo pełny przegląd, albo stosują jakiś wariant metody podziału i ograniczeń wynikający z postaci programu liniowego. Niektóre z nich zawierają jawną implementację podziału i ograniczeń, np. metoda *ASPEN* opisywana w Rozdziale 3.3, inne, na przykład *BC2105*, opisywana w Rozdziale 3.7.1, budują program liniowy ze zmiennymi binarnymi i polegają na metodach wbudowanych w solwery programowania liniowego. Niemniej jednak pierwszym istotnym dla wydajności metody elementem jest właśnie efektywne przeszukiwanie przestrzeni strategii na-

śladowcy. Zewnętrzna pętla proponowanej w tej sekcji metody *O2-UCT* będzie miała za zadanie wykonywać właśnie przeszukiwanie przestrzeni strategii naśladowcy.

Rysunek 4.8 przedstawia zamysł przeszukiwania przestrzeni strategii naśladowcy w metodzie *O2-UCT*. Podejście polega na zastosowaniu heurystycznej metody próbkowania strategii naśladowcy, która wraz z kolejnymi próbkowaniami zbiera informacje o uzyskanych wypłatach *lidera* i wykorzystuje zebrane informacje do ukierunkowania próbkowania na strategie naśladowcy, które mogą dać możliwość zagrania strategii lidera, która da możliwie dużą wypłatę liderowi. Każda z wybranych strategii naśladowcy jest oceniana poprzez znalezienie strategii lidera, dla której ta strategia naśladowcy jest najlepszą odpowiedzią. Strategie lidera znalezione w tym kroku są również kandydatami na przybliżenie strategii lidera ze stanu Równowagi Stackelberga. Algorytm 4.4 przedstawia przebieg zewnętrznej pętli metody *O2-UCT*. Główna pętla algorytmu w linii 3 jest powtarzana aż do osiągnięcia ustalonego warunku stopu. W bieżącej implementacji metody jest to po prostu limit liczby iteracji. Każdy przebieg pętli rozpoczyna się od próbkowania strategii naśladowcy. Heurystyka próbkująca powinna uwzględniać wyniki próbkowań z poprzednich iteracji, i preferować strategie, które potencjalnie mogą dać lepszą wypłatę lidera. Dla każdej uzyskanej w ten sposób strategii naśladowcy dobierana jest strategia lidera, w linii 5, dla której ta strategia naśladowcy jest najlepszą odpowiedzią. Dodatkowo procedura doboru strategii lidera musi optymalizować wypłatę lidera (z zachowaniem warunku, że δ_f jest najlepszą odpowiedzią). Po dobraniu strategii lidera dysponujemy profilem strategii (δ_l, δ_f), który jest kandydatem na wynik algorytmu. Dla tego profilu obliczana jest wartość oczekiwana wypłaty lidera, a następnie, w linii 6, statystyki dla metody próbkowania są aktualizowane. Trzeba tu zwrócić uwagę na fakt, że te statystyki uwzględniają wartość oczekiwaną wypłaty **lidera**, mimo, że próbkujemy strategie naśladowcy. Jest to technika zbliżona do techniki podziału i ograniczeń we wspomnianych metodach z literatury, gdzie odrzuca się podprzestrzenie strategii naśladowcy, dla których lider z pewnością nie będzie miał wypłaty lepszej niż wypłaty znalezione wcześniej. W tym podejściu heurystyka próbkująca ma wybierać strategie naśladowcy, które umożliwią liderowi znalezienie odpowiedzi, która da możliwie dużą wartość u_l. Na końcu pętli, w linii 7 aktualizowana jest najlepsza znana strategia lidera, która na końcu algorytmu posłuży jako zwrócony wynik.

Tak sformułowany szkielet algorytmu należy uzupełnić dwoma elementami: procedurą ukierunkowanego próbkowania strategii naśladowcy (linia 4 Algorytmu 4.4) oraz procedurą poszukiwania strategii lidera dla próbkowanej strategii naśladowcy (linia 5 Algorytmu 4.4). W implementacji *O2-UCT* pierwsza z procedur będzie wykorzystywać UCT, a druga z procedur będzie rodzajem podejścia Podwójnej Wyroczni (Double Oracle). Poniżej opisane są obie te procedury.

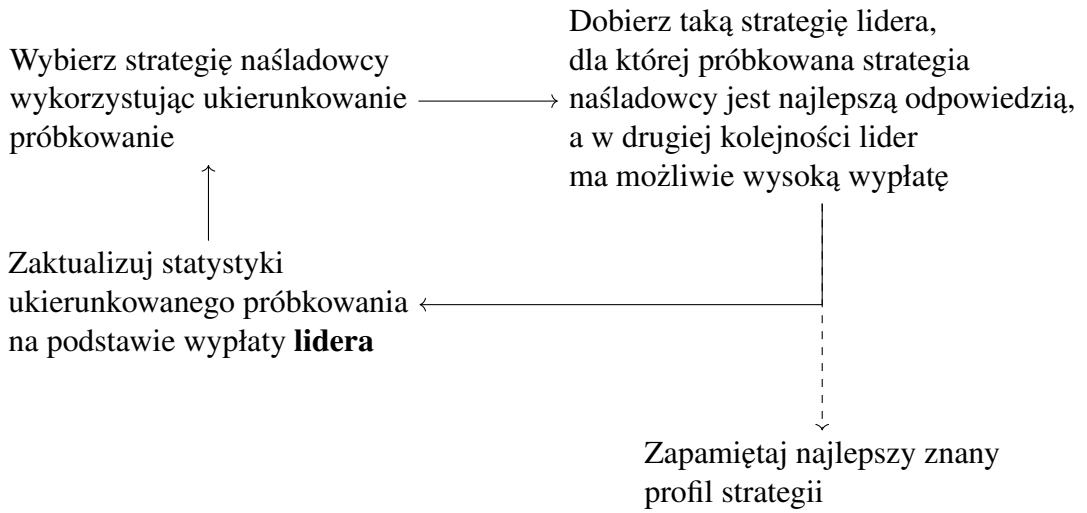
4.3.1 Próbkowanie kandydatów na strategię naśladowcy

Metoda *Upper Confidence Bound applied to Trees* (UCT) opisana w Rozdziale 4.2.1 jest metodą ukierunkowanego przeszukiwania drzew gier. W przypadku metody *O2-UCT* potrzebna

Algorytm 4.4: Zewnętrzna pętla metody *O2-UCT*

wejście: G — gra Stackelberga

$(\delta_l^*, \delta_f^*) \leftarrow null$;
$historicalDataForSampling \leftarrow \varnothing$;
while $\neg stopCondition()$ **do**
  $\delta_f \leftarrow sampleFollowerStr(G, historicalDataForSampling)$;
  $\delta_l \leftarrow findLeaderStrategy(G, \delta_f)$;
  $historicalDataForSampling \leftarrow$
    $append(historicalDataForSampling, (\delta_f, u_l(\delta_l, \delta_f)))$// W tym miejscu użyta jest wypłata lidera!
  if $u_l(\delta_l, \delta_f) > u_l(\delta_l^*, \delta_f^*)$ **then**
    $(\delta_l^*, \delta_f^*) \leftarrow (\delta_l, \delta_f)$;
  end
end
return (δ_l^*, δ_f^*);



Rysunek 4.8: Zewnętrzna pętla metody *O2-UCT*: próbkowanie strategii naśladowcy.

jest ukierunkowane przeszukiwanie przestrzeni strategii naśladowcy. Aby zaadaptować metodę UCT do procedury próbkowania strategii lidera wystarczy skonstruować grę dla jednego gracza, w której węzły terminalne będą odpowiadać strategiom naśladowcy, a nie wprost przypisywać graczowi wypłatę. Posiadając taką grę będzie można zastosować następujący schemat działania:

- Rozegraj grę pomocniczą do liścia (z użyciem symulacji UCT).
- Strategia naśladowcy będąca efektem próbkowania jest strategią naśladowcy przekazaną do głównej pętli *O2-UCT*.
- Wykonywana jest linia 5 Algorytmu 4.4, a na jej podstawie wyliczana jest wartość wypłaty lidera.
- Propagacja w drzewie UCT odbywa się jako realizacja linii 6 w głównym algorytmie *O2-UCT*.

Gra pomocnicza

W opisie gry pomocniczej, gdy mowa o grze pomocniczej lub o grze bez przymiotnika, to tyczy się to gry pomocniczej, która jest konstruowana. Gdy mowa o grze oryginalnej, to tyczy się to gry Stackelberga dla której poszukujemy strategii lidera. Zaprojektowana gra pomocnicza mająca na celu budowę strategii naśladowcy ma następującą strukturę:

- Każdy stan gry ma przypisany zbiór informacyjny z oryginalnej gry. To przypisanie nie jest jednoznaczną etykietą stanu. Może istnieć więcej niż jeden stan gry, do którego przypisany jest ten sam zbiór informacyjny z gry oryginalnej.
- Ze stanu gry możliwe jest do zagrania dokładnie tyle ruchów ile ruchów jest możliwych do zagrania w oryginalnej grze ze zbioru informacyjnego, który jest etykietą stanu. Każdy z ruchów w grze jest etykietowany odpowiadającym ruchem w grze oryginalnej.
- Zagranie ruchu w grze oznacza wybór ruchu odpowiadającego etykiecie w zbiorze informacyjnym odpowiadającym etykiecie stanu w strategii prostej będącej wynikiem rozgrywki.
- Na ścieżce od korzenia do liścia muszą znaleźć się wszystkie potencjalnie osiągalne zbiory informacyjne, gdy grane są ruchy wchodzące w skład budowanej strategii i dokładnie te zbiory informacyjne.

Ostatni z punktów gwarantuje nam, że wynikowe przypisanie ruchów do zbiorów informacyjnych będzie poprawną strategią prostą zredukowaną naśladowcy (definicje 2.20 i 2.15) w oryginalnej grze.

W konstrukcji gry będzie wykorzystana procedura pomocnicza wyznaczająca najbliższych potencjalnych następników stanu oryginalnej gry, należących do zbiorów informacyjnych, które

są punktami decyzyjnymi naśladowcy. Procedura ta opisana jest w Algorytmie 4.5. Procedura polega na schodzeniu do kolejnych stanów w drzewie gry tak długo, aż na każdej ścieżce nie natrafimy na węzeł będący punktem decyzyjnym naśladowcy lub węzeł terminalny. Procedura operuje na kolejce zdefiniowanej w linii 2. W kolejce przechowywane są stany na końcu ścieżek, na których nie napotkano jeszcze zbioru informacyjnego naśladowcy. Dopóki kolejka nie jest pusta, wyjmowany jest jeden stan z kolejki i następuje przetwarzanie. Instrukcja warunkowa w linii 5 sprawdza, czy aktualnie przetwarzany stan nie jest stanem terminalnym. Jeśli tak jest, to stan ten jest pomijany (po stanie terminalnym nie następuje już nic, w szczególności nie nastąpi punkt decyzyjny naśladowcy). W przypadku, gdy stan nie jest stanem terminalnym, są dwie możliwości: jest to punkt decyzyjny lidera lub punkt decyzyjny naśladowcy. W przypadku, gdy jest to punkt decyzyjny naśladowcy, to jest to koniec przeszukiwania bieżącej ścieżki. Dopisujemy znaleziony stan do wynikowego wektora. Wynikowy wektor, początkowo pusty w linii 1, przechowuje pary składające się ze zbioru informacyjnego oraz wektora stanów. Wektor stanów towarzyszący zbiorowi informacyjnemu jest konieczny aby pozyskać potencjalne następniki dla tego zbioru informacyjnego. Będzie on użyty w konstrukcji gry. Zbiory informacyjne w wektorze wyjściowym muszą być unikatowe. W przypadku, gdy wiele ścieżek prowadzi do tego samego zbioru informacyjnego, wszystkie napotkane stany należące do tego zbioru informacyjnego znajdą się w wektorze towarzyszącym temu stanowi. W linii 8 następuje sprawdzenie, czy znaleziony zbiór informacyjny naśladowcy już znajduje się w wyjściowym wektorze, jeśli tak, to do w istniejącej parze reprezentującej ten zbiór informacyjny, wektor stanów rozszerzany jest o stan przetwarzany w tej iteracji pętli. W przypadku, gdy zbioru informacyjnego nie ma w wektorze wyjściowym, to jest on dodawany wraz z jednoelementowym wektorem stanów. W przypadku, gdy napotkany stan jest punktem decyzyjnym lidera, to w linii 15 następuje przejście do wszystkich możliwych następników tego stanu. Dla każdej możliwej akcji symulowana jest oryginalna gra. Należy pamiętać, że gra może być niedeterministyczna i w związku z tym jeden ruch może skutkować osiągnięciem kilku różnych stanów gry. Wszystkie uzyskane w ten sposób stany, dla wszystkich akcji w linii 18 są dodawane do kolejki stanów do przetworzenia, po której iteruje zewnętrzna pętla. Efektem wykonania całej procedury jest wektor par (zbiór informacyjny, wszystkie osiągnięte stany należące do tego zbioru informacyjnego). Znalezione pary będą elementem etykiet stanów w konstruowanej grze.

Dodatkowo potrzebna jest procedura pomocnicza, która zaaplikuje powyższy algorytm do zbioru (lub wektora) stanów należących, a nie do pojedynczego stanu. Jest to proste rozszerzenie, które po prostu łączy ze sobą pary odpowiadające temu samemu zbiorowi informacyjnemu z wielu wywołań powyższej procedury. Postępowanie to jest prezentowane w Algorytmie 4.6.

Konstrukcja gry Konstrukcja gry jest przeprowadzona w następujący sposób:

- Etykietą stanu gry będzie para. W jej skład, oprócz wymaganego powyżej zbioru informacyjnego naśladowcy, będzie wchodzić kolejka zbiorów informacyjnych do przetworzenia. W teorii kolejka może być dowolnym rodzajem kolejki (FIFO, LIFO, etc.),

Algorytm 4.5: Znajdowane najbliższych następników stanu oryginalnej gry, które są punktami decyzyjnymi naśladowcy.

wejście: s_0 — stan oryginalnej gry

```
1  s_f ← ∅// Wyjściowy wektor par (zbiór informacyjny, wektor
       stanów), które są punktami decyzyjnymi naśladowcy
2  q ← [s_0]// Kolejka stanów do przetworzenia
3  while q ≠ ∅ do
4      s ← pop(q);
5      if ¬isTerminal(s) then
6          I ← informationSet(s);
7          if player(I) = follower then
8              if contains(s_f, I) then
9                  (I, v) ← get(s_f,I)// Znajdź w wektorze parę
                       zawierającą zbiór informacyjny na pierwszej
                       pozycji
10                 replace(s_f, (I, v), (I, v ⊕ s))// Zamień parę zawierającą
                       zbiór informacyjny na parę z rozszerzonym
                       wektorem stanów o bieżący stan
11             else
12                 append(s_f, (I, s));
13             end
14         else
15             for a ∈ possibleActions(I) do
16                 s' ← playActionAllPossibleSuccessors(s,a)// Wszystkie
                       możliwe stany, które następują po zagraniu
                       akcji a (w grze niedeterministycznej może być
                       więcej niż 1)
17                 for s' ∈ s' do
18                     put(q,s');
19                 end
20             end
21         end
22     end
23 end
24 return s_f;
```

Algorytm 4.6: Uogólnienie procedury z Algorytmu 4.5 dla całego wektora stanów

```
   wejście: s — wektor stanów
1  s_f ← ∅ // Wektor wyjściowy
2  for s ∈ s do
3  |  t ← findFollowerSuccessors(s) // Wywołanie procedury z
   |     Algorytmu 4.5
4  |  for (I_t, s_t) ∈ t do
5  |  |  if contains(s_f, I_t) then
6  |  |  |  (I, v) ← get(s_f,I_t)// Znajdź w wektorze parę zawierającą
   |  |  |     zbiór informacyjny na pierwszej pozycji
7  |  |  |  replace(s_f, (I, v), (I, v ⊕ s_t))// Zamień parę zawierającą
   |  |  |     zbiór informacyjny na parę z rozszerzonym
   |  |  |     wektorem stanów o elementy teraz rozważanego
   |  |  |     wyniku
8  |  |  else
9  |  |  |  append(s_f, (I_t, s_t));
10 |  |  end
11 |  end
12 end
13 return s_f
```

w implementacji została użyta kolejka FIFO. Dodatkowo zbiór informacyjny będzie rozszerzony o wektor stanów uzyskanych z powyższych algorytmów, które należą do tego zbioru. Uwaga: to nie muszą być wszystkie stany wchodzące w skład tego zbioru informacyjnego w oryginalnej grze.

- Konstrukcja stanu początkowego przebiega w następujący sposób: uruchom procedurę z Algorytmu 4.5 dla korzenia oryginalnej gry. Weź pierwszy element zwróconego wektora. Ten element będzie częścią etykiety stanu gry definiującą odpowiadający w zbiór informacyjny w oryginalnej grze. Pozostałą część zwróconego wektora umieść w drugiej częśći etykiety — kolejce elementów do przetworzenia.

- W każdym stanie do zagrania są dostępne ruchy etykietowane ruchami z oryginalnej gry, które są dostępne w zbiorze informacyjnym będącym etykietą stanu.

- Konstrukcja następnika: wejściem do procedury konstrukcji następnika jest stan poprzedzający oraz ruch, który prowadzi do tego stanu. Procedura konstrukcji jest opisana w Algorytmie 4.7 i składa się z dwóch kroków. W pierwszym kroku wyliczane są możliwe następniki stanów ze zbioru informacyjnego węzła poprzedzającego po zagraniu ruchu, który doprowadził do stanu konstruowanego. Te następniki są zbierane w wektorze $\mathbf{v}$, wypełnianym w pętli w linii 2. Stany w wektorze mogą należeć do zbioru informacyjnego dowolnego z graczy lub być stanami terminalnymi. Stosowana jest procedura z Algorytmu 4.6, aby cały wektor stanów przeprowadzić na wektor zbiorów informacyjnych

naśladowcy wraz z wektorem stanów wchodzących w skład tego zbioru informacyjnego. Wszystkie uzyskane stany są potencjalnie osiągalne po zagraniu wybranego ruchu, konieczne będzie przypisanie im decyzji, żeby skonstruować zredukowaną strategie prostą. W związku z tym wszystkie te zbiory informacyjne dodawane są do kolejki stanów do przetworzenia (pętla w linii 7). W drugim kroku konstruowany jest właściwy stan. Jeśli kolejka zbiorów informacyjnych, którym trzeba przypisać ruch jest pusta, to osiągnięty został stan terminalny. W przeciwnym wypadku tworzony jest kolejny stan nieterminalny, etykietowany pierwszym zbiorem informacyjnym z kolejki i zawierający pozostałą część kolejki jako drugą część etykiety.

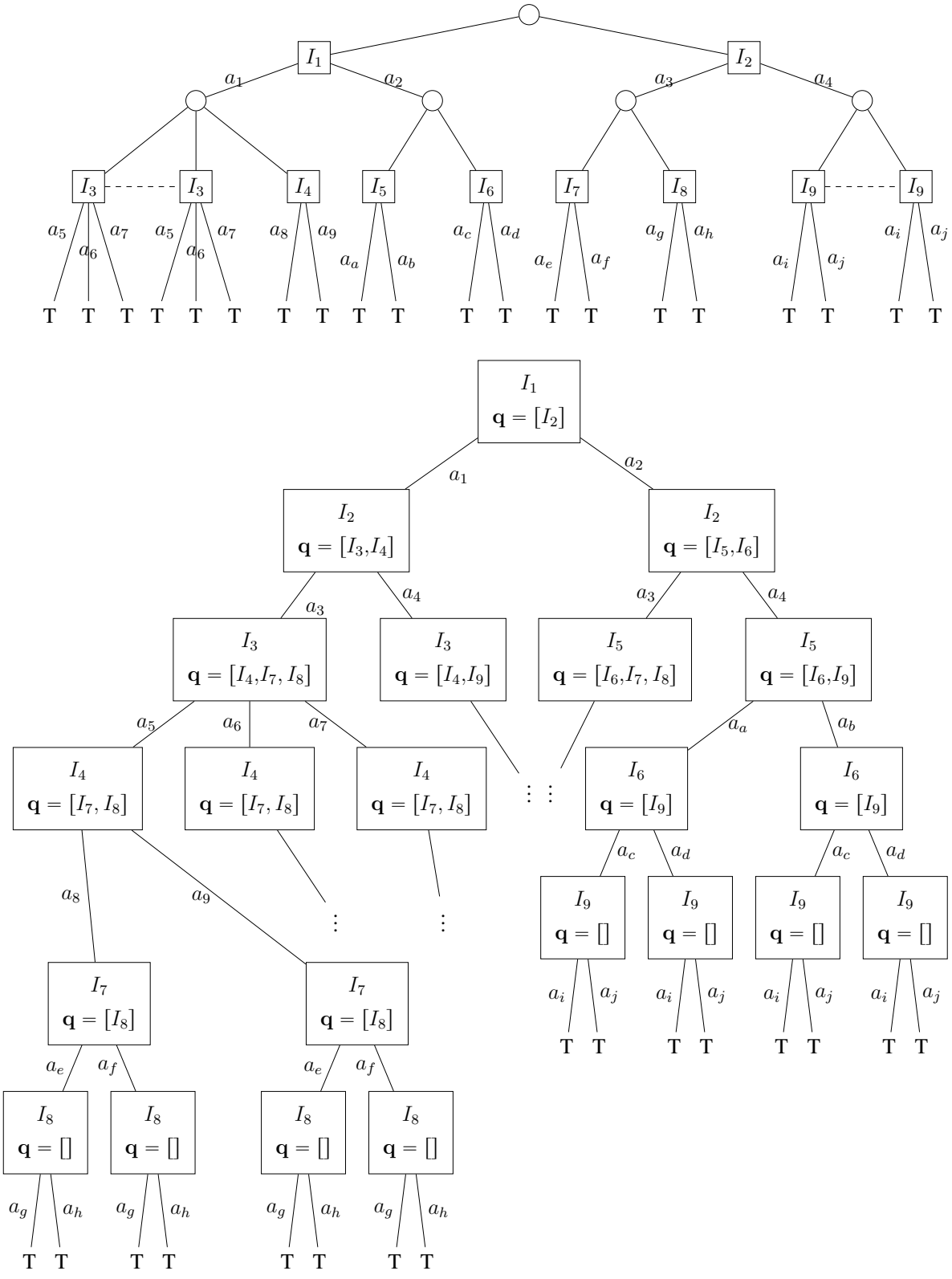
Algorytm 4.7: Konstrukcja następnika w pomocniczej grze

wejście: $(I_p, \mathbf{s}_p, \mathbf{q})$ — etykieta stanu poprzedzającego: zbiór informacyjny, należące do niego stany oraz kolejka zbiorów informacyjnych do przetworzenia
wejście: a — etykieta akcji, która prowadzi ze stanu poprzedzającego do konstruowanego stanu

```
1  v ← ∅;
2  for s ∈ s_p do
3  |  s' ← playActionAllPossibleSuccessors(s,a);
4  |  v ← v ∪ s';
5  end
6  q' ← findFollowerSuccessorsVec(v)// Wywołanie procedury z
       Algorytmu 4.6
7  for e ∈ q' do
8  |  put(q,e);
9  end
10 if empty(q) then
11 |  return stan terminalny
12 else
13 |  (I_new, s_new) ← pop(q);
14 |  return stan nieterminalny (I_new, s_new)
15 end
```

Przykładowa struktura gry oryginalnej i gry pomocniczej przedstawiona jest na Rysunku 4.9. Wybór kolejno ruchów a_1,a_3,a_5,a_9,a_e,a_g oznacza konstrukcję strategii zredukowanej w oryginalnej grze, która przypisuje zbiorom informacyjnym następujące akcje: $I_1 \rightarrow a_1$,$I_2 \rightarrow a_3$, $I_3 \rightarrow a_5$,$I_4 \rightarrow a_9$,$I_7 \rightarrow a_e$,$I_8 \rightarrow a_g$

Tak skonstruowana gra wykorzystywana jest do próbkowania strategii naśladowcy w metodzie *O2-UCT*. Rozpoczynamy od pustego drzewa UCT, metoda UCT będzie poszukiwać ruchów w pomocniczej grze. Pojedyncze próbkowanie strategii naśladowcy do pojedynczy przebieg selekcji, ekspansji, symulacji (symulacja zawsze osiąga węzeł terminalny). Tak wybrana strategia naśladowcy jest przekazywana do kolejnych kroków szkieletu *O2-UCT* prezentowanego na Rysunku 4.8. W kolejnych krokach wyliczana jest strategia lidera i wyliczana wartość wypłaty. Wyliczona wartość wypłaty lidera jest wykorzystywana w metodzie UCT jako wynik



Rysunek 4.9: Rysunek przedstawiający oryginalną grę (u góry) oraz skonstruowaną grę pomocniczą (u dołu). Etykiety węzłów w grze pomocniczej to zbiory informacyjne oraz kolejka zbiorów informacyjnych do przetworzenia. Dla czytelności rysunku pominięto wektory stanów stowarzyszone z poszczególnymi zbiorami informacyjnymi. Część poddrzew została również pominięta, ze względu na ograniczoną przestrzeń. W drzewie oryginalnej gry zostały pominięte wszystkie etykiety związane z zagraniami lidera. Etykieta T oznacza stan terminalny.

gry do propagacji. Takie podejście, z tak skonstruowaną grą powoduje, że szacowana jest jakość każdego ruchu wchodzącego w skład strategii prostej naśladowcy z osobna, co pozwala wybierać te ruchy naśladowcy, które pozwolą wybrać liderowi strategie z dobrą wypłatą, zachowując warunek, że dana strategia naśladowcy ma być najlepszą odpowiedzią na strategię lidera.

Pętla próbkująca strategie naśladowcy jest powtarzana aż do momentu, gdy w drzewie UCT istnieje ścieżka od korzenia do liścia, dla której *wszystkie* ruch na tej ścieżce mają wartość licznika odwiedzin większa lub równą $outerSimulationCount$.

4.3.2 Metoda poszukiwania strategii lidera

Drugą z procedur, wykorzystywaną w głównej pętli metody *O2-UCT*, opisanej na Rysunku 4.8, jest znalezienie strategii mieszanej lidera, dla której uzyskana z poprzedniej procedury strategia naśladowcy jest najlepszą odpowiedzią.

Idea procedury polega na iteracyjnym poprawianiu strategii lidera. Strategia lidera przechowywana jest w zmiennej globalnej i jest inicjowana od razu po wykonaniu próbkowania strategii naśladowcy (ozn. π^r_f). Strategia lidera początkowo jest inicjowana strategią prostą. Następnie uruchamiana jest iteracyjna procedura, która poprawia wynik tej strategii lidera tak, aby jednocześnie π^r_f była najlepszą odpowiedzią naśladowcy. W każdej iteracji ta zmienna jest modyfikowana — zmieniane są prawdopodobieństwa ruchów w strategii. Kierunek zmian strategii zależy od tego, jakie warunki spełnia strategia lidera. W każdej iteracji następuje sprawdzenie, czy strategia naśladowcy uzyskana z poprzedniej części metody jest najlepszą odpowiedzią na tę strategię lidera, czy też istnieje jakaś strategia prosta naśladowcy π^b_f, która da naśladowcy wyższą wypłatę. W pierwszej sytuacji krok poprawy strategii lidera dobierany jest w taki sposób, aby zwiększyć wypłatę lidera, gdy strategia jest grana przeciwko strategii π^r_f, w przeciwnym przypadku poprawka wybierana jest tak, aby zmniejszyć wypłatę naśladowcy, gdy gra on strategię π^b_f, a zwiększyć gdy gra π^r_f, innymi słowy zmiana jest wykonywana w takim kierunku, żeby π^b_f nie było dłużej odpowiedzią naśladowcy lepszą niż π^r_f.

W opisie działania metody sytuacja, gdy π^r_f jest najlepszą odpowiedzią naśladowcy na bieżącą strategię lidera, będzie nazywana strategią lidera w obszarze dopuszczalnym (patrz Definicja 3.1), a przypadek, gdy istnieje π^b_f, taka, że wypłata oczekiwana naśladowcy w grze przeciwko strategii lidera jest większa niż dla π^r_f będzie nazywana strategią lidera w obszarze niedopuszczalnym. Ta nomenklatura jest analogią do programów do rozwiązywania gier Stackelberga opisanych w Rozdziale 3. We wspomnianych programach występuje grupa ograniczeń liniowych, które gwarantują, że pewna strategia naśladowcy jest najlepszą odpowiedzią naśladowcy. Pozostałe układy zmiennych decyzyjnych, niespełniające tych ograniczeń, czyli obszar niedopuszczalny, to te strategie lidera, dla których jest inna najlepsza odpowiedź naśladowcy (lub układ zmiennych nie definiuje poprawnej strategii).

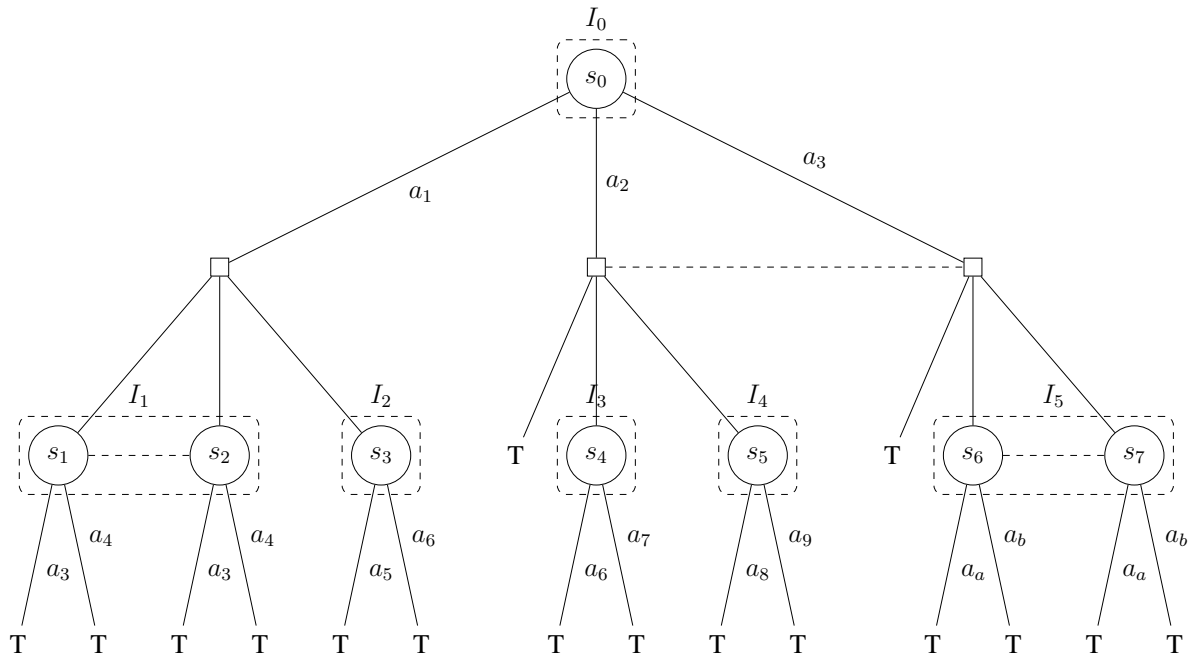
Reprezentacja strategii mieszanej

Przedmiotem tej rozprawy jest poszukiwanie strategii w grach wielokrokowych z niepełną informacją o sumie niezerowej. Dodatkowym warunkiem, który mają spełniać gry jest doskonała pamięć. W grach z doskonałą pamięcią strategię mieszaną można utożsamiać ze strategią behawioralną (Definicja 2.19, równoważność strategii behawioralnej i mieszanej jest przytoczona w Rozdziale 2.4). Strategię behawioralną da się reprezentować jako drzewo. Taki sposób reprezentacji jest pożądany, bo pozwolił uzyskanie mniejszego zużycia pamięci operacyjnej.

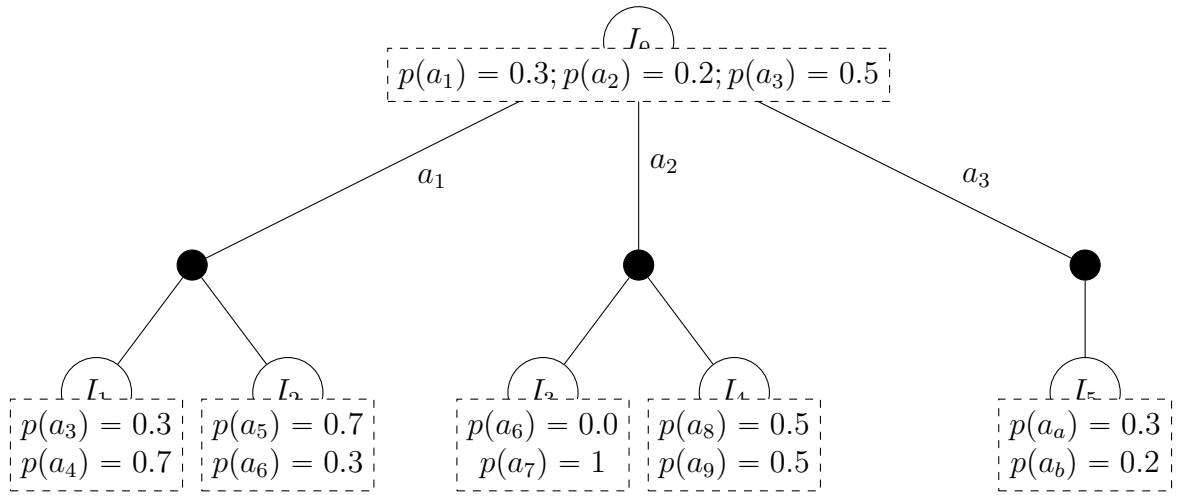
Podsumowując definicję strategii behawioralnej z wcześniejszej części pracy: strategia ta polega na przypisaniu rozkładu prawdopodobieństwa akcji w każdym zbiorze informacyjnym, będącym punktem decyzyjnym gracza, z osobna. Ponadto, można zaobserwować, że w grze z doskonałą pamięcią, można zdefiniować relację poprzednik-następnik pomiędzy stanami gry. Każdy zbiór informacyjny niebędący korzeniem ma jednoznacznie zdefiniowany zbiór informacyjny, który został zaobserwowany przez gracza przed zaobserwowaniem tego zbioru informacyjnego. Jednoznaczność wynika z faktu, że w gra z doskonałą pamięcią jest definiowana jako taka, gdzie gracz odróżnia wszystkie stany, do których zaprowadziła go różna sekwencja wykonanych akcji i zaobserwowanych zbiorów informacyjnych. Taka relacja zatem definiuje drzewo zbiorów informacyjnych gracza, gdzie poprzednikiem danego zbioru informacyjnego jest właśnie ostatni zaobserwowany wcześniej zbiór informacyjny. Należy zauważyć, że krawędzie w tym drzewie możemy (w sposób nieunikatowy) etykietować ruchami, które zostały zagrane w poszczególnych zbiorach informacyjnych, żeby zaobserwować kolejne. Podobnie, ruch poprzedzający zbiór informacyjny jest zdefiniowany jednoznacznie, ze względu na założenie o doskonałej pamięci. Tak zbudowane drzewo można potem wzbogacić o rozkład prawdopodobieństwa przypisany do każdego z węzłów. W efekcie możemy reprezentować strategię behawioralną jako drzewo. Przykład reprezentacji strategii dla gry przedstawionej na Rysunku 4.10 znajduje się na Rysunku 4.11. Warto zwrócić uwagę na czarne węzły, których istnienie wynika z faktu, że w zależności od niedeterminizmu gry oraz zagrań przeciwnika ten sam ruch możne doprowadzić do zaobserwowania przez gracza różnych zbiorów informacyjnych.

Strategia behawioralna może przypisywać niektórym akcjom prawdopodobieństwo równe 0. To oznacza, że rozkłady prawdopodobieństwa w węzłach następujących po krawędzi etykietowanej akcją z prawdopodobieństwem 0 nigdy nie będą wykorzystane, bo te zbiory informacyjne nigdy nie zostana napotkane. W związku z tym możemy zbudować drzewo, które nie zawiera wszystkich zbiorów informacyjnych z gry. W ten sposób, z punktu widzenia implementacji, można oszczędzić pamięć konieczną do przechowania nieużywanych fragmentów drzewa. Przykładowe drzewo przechowujące zredukowaną strategię zostało przedstawione na Rysunku 4.12. Prawdopodobieństwo akcji a_2 wynosi 0, w związku z tym całe poddrzewo następujące po tej akcji mogło być usunięte ze strategii.

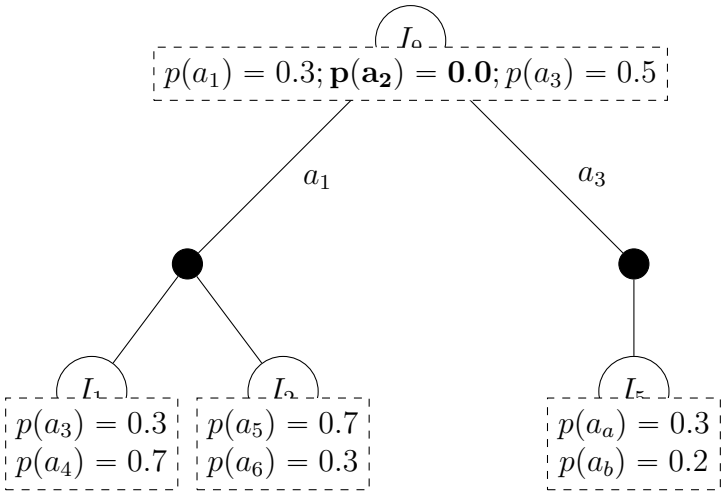
W implementacji stosowanej w metodzie *O2-UCT* redukcja drzewa reprezentującego strategię behawioralną poszła jeszcze dalej. Użyto pojęcia domyślnej strategii w zbiorze informacyj-



Rysunek 4.10: Gra w postaci ekstensywnej, dla której konstruowana jest strategia behawioralna. Na rysunku pominięte są etykiety stanów i ruchów naśladowcy. T oznacza węzeł terminalny. Etykiety nad prostokątami rysowanymi przerywaną linią to zbiory informacyjne.



Rysunek 4.11: Drzewo reprezentujące strategię behawioralną dla gry z Rysunku 4.10.



Rysunek 4.12: Drzewo reprezentujące zredukowaną strategię behawioralną w grze z Rysunku 4.10.

nym, tak jak w przytaczanej w Rozdziale 3.6 metodzie podwójnej wyroczni. Domyślna strategia stosowana jest, gdy dany zbiór informacyjny nie znajduje się w drzewie strategii, a wyliczenie wartości gry wymaga podjęcia decyzji w tym zbiorze informacyjnym. W implementacji metody przyjęto, że strategia domyślna, to rozkład jednostajny nad wszystkimi ruchami dostępnymi w danym zbiorze informacyjnym. Dzięki temu przechowywane w pamięci drzewo reprezentujące strategię może nie zawierać niektórych zbiorów informacyjnych, a mimo to, po uwzględnieniu strategii domyślnej, można ją traktować jako pełną strategię behawioralną. W metodzie *O2-UCT* początkowe drzewo reprezentujące strategię lidera składa się z pojedynczej ścieżki i dopiero w trakcie działania metody jest ono rozszerzane o gałęzie potrzebne do wykonania kroków poprawy strategii.

Szczegóły metody poszukiwania strategii lidera

Schemat poszukiwania strategii lidera w metodzie *O2-UCT* przedstawiony jest na Rysunku 4.8. Poszukiwanie to polega na zainicjowaniu strategii lidera, a następnie, w pętli powtarzaniu kroków poprawy strategii. Kierunek poprawy może być dwojaki. Najpierw wykonywane jest przeszukiwanie wszystkich strategii prostych naśladowcy, z użyciem pełnego przeglądu. Dla każdej ze przeglądanych strategii wyliczana jest wartość oczekiwana wypłaty naśladowcy, gdy ta strategia naśladowcy jest grana przeciwko strategii lidera. Jeśli natrafiono na strategię, która ma wyższą wartość wypłaty niż π^r_f, to przegląd jest przerywany, a znaleziona strategia w dalszej części iteracji będzie oznaczona jako π^b_f. Znalezienie takiej strategii oznacza, że w chwili obecnej jesteśmy w obszarze niedopuszczalnym i krok poprawy powinien zostać wykonany tak, aby zbliżyć się do obszaru dopuszczalnego, czyli zmniejszyć $u_f(\delta_l, \pi^b_f)$, a zwiększyć $u_f(\delta_l, \pi^r_b)$. Po wykonaniu tego kroku, rozpoczynamy kolejną iterację pętli. Jeśli nie znaleziono żadnej strategii lepszej od π^r_f, to znaczy, że bieżące rozwiązanie jest w obszarze dopuszczalnym. W takiej sytuacji, jeśli nasze rozwiązanie jest lepsze od poprzednich, to aktualizujemy najlepsze znane

rozwiązanie, a następnie wykonujemy krok poprawy, który zwiększy wypłatę lidera, gdy gra on przeciwko π_f^r. Po tym kroku wykonywana jest kolejna iteracja pętli . Uwaga: nie można wykonać sprawdzenia najlepszego rozwiązania po aktualizacji strategii lidera, gdyż nie ma pewności, czy poprawiona strategia nie jest poza obszarem dopuszczalnym. Pętla poprawiająca wykonywana jest od osiągnięcia ustalonego warunku stopu.

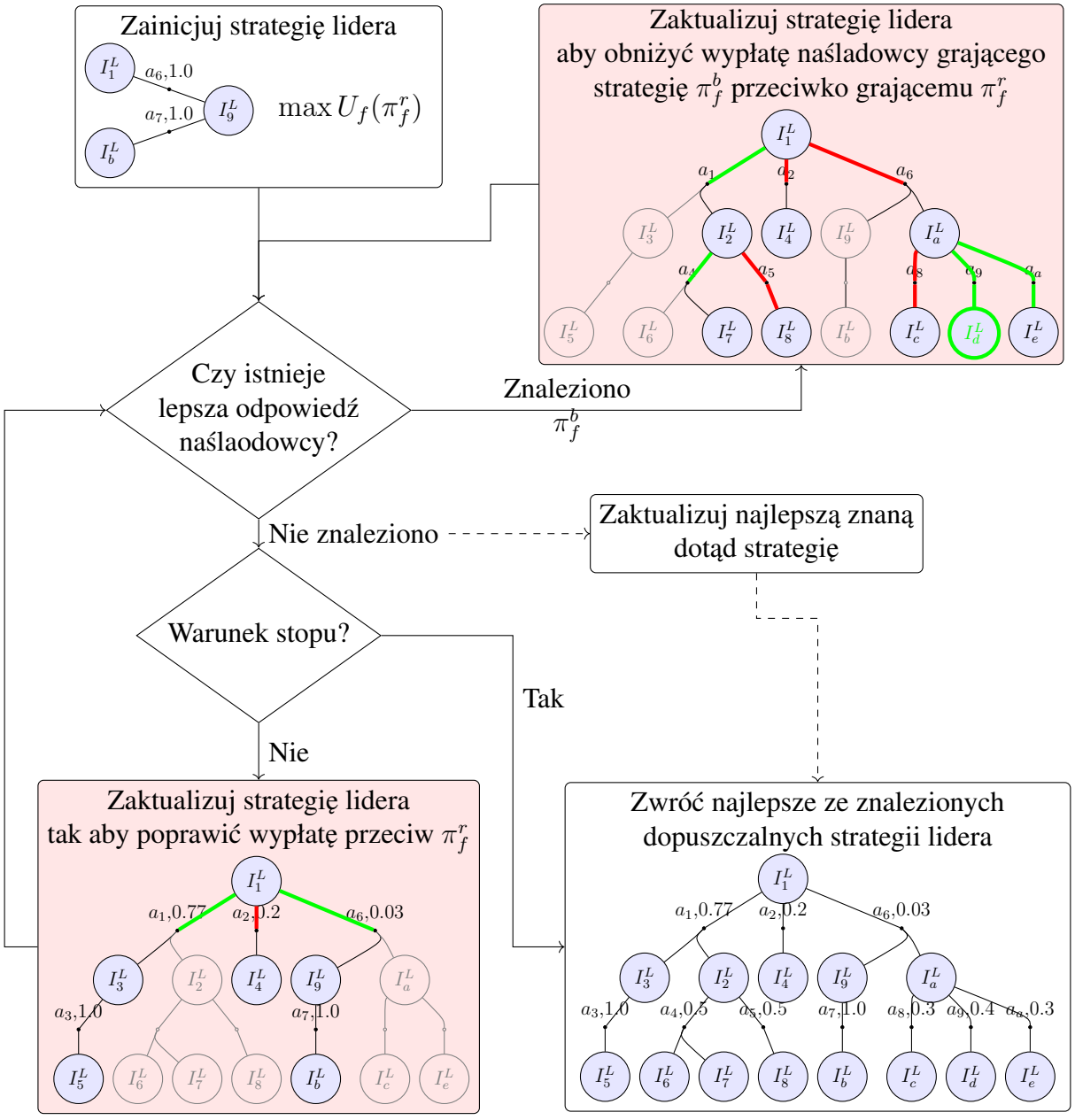
W dalszej części tego rozdziały opisane są poszczególne operacje wykonywane w ramach pętli. W pierwszej kolejności prezentowana jest metoda inicjowania strategii lidera. Następnie prezentowany jest schemat aktualizacji drzewa strategii. Schemat aktualizacji jest ten sam, dla obu kierunków poprawy, różni się tylko parametrami. Na koniec przedyskutowany jest warunek stopu.

Inicjowanie strategii lidera. Procedura inicjująca strategię buduje strategię, która będzie maksymalizować wypłatę naśladowcy gdy ta strategia lidera jest grana przeciwko π_f^r — strategii naśladowcy uzyskanej w zewnętrznej pętli *O2-UCT*. Podejście to może z pozoru wydawać się nieintuicyjne, alternatywą byłoby inicjowanie strategii tak, aby zmaksymalizować wypłatę lidera, gdy naśladowca zagra π_f^r. Należy jednak pamiętać, że poszukujemy strategii lidera, która za możliwie dużą wypłatę, ale tylko pośród strategii, dla których najlepsza odpowiedzią naśladowcy jest wspomniana π_f^r. Zainicjowanie strategii tak, aby maksymalizowała wypłatę lidera spowoduje, ze strategia ta, prawie na pewno, będzie poza obszarem dopuszczalnym. W związku z tym lepszą decyzją jest wybór strategii, która maksymalizuje wypłatę naśladowcy, gdy grana jest strategia π_f^r, a co za tym idzie jest duża szansa, że startowa strategia będzie w obszarze dopuszczalnym.

Teoretycznie możliwe było by znalezienie strategii lidera poprzez wykonanie pełnego przeglądu gry i wyliczenie poszczególnych wartości oczekiwanych każdego ruchu. Postępowanie to jest jednak bardzo kosztowne obliczeniowo dla dużych gier. W związku z tym zostanie zastosowana procedura znajdowania najlepszych ruchów bazująca na metodzie UCT. Procedura będzie startować w korzeniu i dodawać najlepiej oszacowany ruch do strategii. Następnie będzie schodzić do potomka nowododanego ruchu i powtarzać całość operację aż do osiągnięcia węzła, który nie ma następników.

Procedura wybierająca pojedynczy ruch do dodania do drzewa strategii jest opisana w Algorytmie 4.8. Znajdowanie ruchu polega na uruchomieniu metody UCT (linia 2) i zwróceniu najlepszej znalezionej akcji w grze rozważanej przez UCT. Jedyną modyfikacją względem podstawowej wersji metody jest zastosowanie funkcji Q, która definiuje wypłatę dla osiągniętego stanu terminalnego, zamiast przypisywać wprost liczby z wypłatą. Możliwość wymiany funkcji Q pozwoli na zastosowanie tej procedury do dodawania gałęzi do drzewa strategii w innych fazach poszukiwania strategii lidera. Nietrywialnym elementem algorytmu jest konstrukcja gry dla jednego gracza w linii 1. Tworzona jest niedeterministyczna gra, która ma następujące cechy:

- Widoczny dla gracza (metody UCT) stan gry to zbiór informacyjny lidera do którego



Rysunek 4.13: Schemat poszukiwania strategii lidera w metodzie *O2-UCT*. Prostokąty z cieniowanym tłem przedstawiają procedurę aktualizacji strategii.

należy stan z poprzedniego punktu.

- Wewnętrznie gra jako stan przechowuje stan oryginalnej gry będący elementem pewnego zbioru informacyjnego lidera.
- Dopuszczalne akcje to akcje to akcje, które lider może wykonać ze zbioru informacyjnego będącego stanem gry, z zastrzeżeniem dla pierwszego stanu w grze, o czym dalej.

W algorytmie, na początku rozgrywki w pomocniczą grę losowany jest stan z wektora $\mathbf{s}$. Prawdopodobieństwa wylosowania poszczególnych stanów są zdefiniowane w wektorze $\mathbf{p}$. Pierwszy stan, zgodnie z założeniami co do danych wejściowych, należy do zbioru informacyjnego I, który jest zbiorem informacyjnym lidera. W pierwszym stanie gry, którą rozwiązuje algorytm UCT do wyboru są wszystkie akcje ze zbioru akcji dopuszczalnych w zbiorze informacyjnym I oprócz akcji ze zbioru $\mathbf{a}$ (w trakcie początkowej budowy drzewa zbiór zakazanych akcji jest pusty, w dalszych etapach będzie on wykorzystany). Wykonanie ruchu w grze dla jednego gracza i przejście do kolejnego stanu jest wykonywane w następujący sposób:

1. Wykonaj ruch z bieżącego stanu w grze dla dwóch graczy.
2. Jeśli wynik ruchu jest niedeterministyczny, wykonaj losowe próbkowanie stanu (zgodnie z rozkładem prawdopodobieństwa w grze), w przeciwnym wypadku weź stan będący skutkiem tego ruchu.
3. Uzyskany stan może:
   - należeć do zbioru informacyjnego lidera, wtedy nowy stan gry będący skutkiem zagrania ruchu to osiągnięty właśnie stan,
   - należeć do zbioru informacyjnego naśladowcy, wtedy zagraj ruch ze strategii π_f i przejdź do punktu 2,
   - być węzłem terminalnym, wtedy zwróć stan terminalny odpowiadający temu węzłowi.

Po osiągnięciu stanu terminalnego do algorytmu UCT propagowana jest wartość końcowa gry obliczona funkcją Q. W przypadku, gdy maksymalizujemy wypłatę naśladowcy, wybieramy funkcję zwracającą wypłatę naśladowcy, $Q(z) = u_f(z)$.

Aby zainicjować strategię lidera konieczne jest wielokrotne dodawanie ruchów od korzenia, aż do osiągnięcia stanu, po którym nie ma już następników (następują węzły terminalne). Procedura inicjowania całej strategii jest przedstawiona w Algorytmie 4.9. Koncepcja procedury jest następująca: wybierz najlepiej oceniany ruch i dodaj go do strategii, następnie, dla każdego zbioru informacyjnego, który lider może zaobserwować bezpośrednio po tym stanie, przy założeniu, że naśladowca gra strategię π_f^r (ograniczenie do jednej strategii naśladowcy może znacząco zmniejszyć liczbę zbiorów informacyjnych do rozważenia), dodaj węzeł odpowiadający temu zbiorowi do drzewa strategii behawioralnej. Uruchom procedurę dodawania ruchu

Algorytm 4.8: Procedura wyboru ruchu do dodania strategii behawioralnej

wejście: $(I, (\mathbf{s},\mathbf{p}))$ — Zbiór informacyjny i pary stan, prawdopodobieństwo trafienia na stan. Wszystkie stany należą do tego zbioru informacyjnego. Na odpowiadających sobie pozycjach w wektorach $\mathbf{s}$ i $\mathbf{p}$ jest prawdopodobieństwo trafienia na dany stan.
wejście: $\mathbf{a}$ — Wektor akcji ze zbioru informacyjnego I, których nie rozważać, bo są już w strategii
wejście: π_f — Strategia prosta naśladowcy
wejście: $Q : z \to \mathbb{R}$ — funkcja definiująca wypłatę stanu terminalnego dla metody UCT

1 $G \leftarrow singlePlayerGame(I, \mathbf{s}, \mathbf{p}, \mathbf{a}, \pi_f)$;
2 $uctTree \leftarrow UCT(G, Q)$;
3 **return** $bestMove(uctTree)$;

rekurencyjnie dla każdego z dodanych węzłów. Rekurencyjne uruchamianie procedury gwarantuje, że strategia będzie rozbudowywana aż do napotkania zbiorów informacyjnych, po których następują stany terminalne.

Wyjaśnienia wymaga procedura poszukiwania wszystkich zbiorów informacyjnych osiąganych w efekcie zagrania danego ruchu, która jest wywołana w linii 4. Koncepcyjnie ta procedura jest bardzo podobna do procedury z Algorytmu 4.5, choć zachodzą pewne różnice:

- Tym razem szukamy zbiorów informacyjnych lidera przy ustalonej strategii naśladowcy (we wspomnianym algorytmie nie robiliśmy żadnych założeń na temat strategii przeciwnika, wszystkie ruchu były możliwe).
- Schodząc do kolejnych stanów, tym razem, jeśli natrafimy na niedeterminizm, to nie bierzemy tylko wszystkich możliwych kontynuacji, ale również zbieramy, w wektorze $\mathbf{p}$ prawdopodobieństwa osiągnięcia tych stanów.
- Budowa wektora prawdopodobieństw natrafienia na kolejne stany $\mathbf{p}$ jest niezbędna, żeby potem móc zbudować grę pomocniczą wykorzystywaną przez algorytm UCT.

Wszystkie nowododane akcje mają przypisane prawdopodobieństwo 1 w rozkładzie w danym zbiorze informacyjnym. Strategia zbudowana poprzez aplikację Algorytmu 4.9 na korzeniu strategii behawioralnej jest strategią początkową dla pętli poszukującej strategii lidera.

Znalezienie strategii π_f^b. Strategia lidera budowana w kroku inicjującym strategię jest konstruowana tak, aby wszystkie zbiory informacyjne osiągane, gdy gra się przeciwko π_f^r, były w tej strategii. Przegląd wszystkich strategii naśladowcy oznacza, że wyliczając wypłaty przeciwko części z nich będzie trzeba użyć strategii domyślnej, czyli rozkładu jednostajnego. Jeśli znaleziono jakąś lepszą strategię naśladowcy π_f^b, to przed rozpoczęciem procedury poprawy w kierunku obszaru dopuszczalnego do strategii lidera dodawane są wszystkie węzły, związane ze zbiorami informacyjnymi, które nie były osiągalne podczas gry przeciwko π_f^r, a są osiągalne

Algorytm 4.9: Wielokrotne rozszerzanie drzewa reprezentującego strategię o kolejne ruchy aż do osiągnięcia stanu bez następników.

wejście: N — węzeł drzewa reprezentującego strategię behawioralną
wejście: $(I, (\mathbf{s},\mathbf{p}))$ — Zbiór informacyjny i pary stan, prawdopodobieństwo trafienia na stan. Wszystkie stany należą do tego zbioru informacyjnego. Na odpowiadających sobie pozycjach w wektorach $\mathbf{s}$ i $\mathbf{p}$ jest prawdopodobieństwo trafienia na dany stan.
wejście: π_f — Strategia prosta naśladowcy

```
1 def ExpandStrategy(N, (I, (s, p)), π_f):
2     moveToAdd ← calculateMoveToAdd(I, (s, p), π_f, ∅, Q)// Wywołanie
          procedury z Algorytmu 4.8
3     addNewMove(N, moveToAdd);
4     succ ←
          findLeadersSuccessors((I, (s, p)), moveToAdd, π_f)// Znalezienie
          wszystkich zbiorów informacyjnych lidera, które mogą
          zostać napotkane po zbiorze I, gdy naśladowca gra
          ustaloną strategię
5     for (I', (s', p')) ∈ succ do
6         N' ← subtree(N, moveToAdd, I');
7         ExpandStrategy(N', (I', (s', p')), π_f);
8     end
```

w grze przeciw π_f^b. W tym celu, przechodzone jest całe drzewo strategii behawioralnej od korzenia w dół. Przy pierwszym napotkanym zbiorze informacyjnym, którego brakuje w drzewie uruchamiany jest Algorytm 4.9, tym razem ze strategią $\pi_f = \pi_f^b$. Procedura ta jest powtarzana aż do momentu, gdy w drzewie nie ma już brakujących zbiorów informacyjnych.

Procedura poprawy strategii lidera. Na Rysunku 4.13 znajdują się dwa bloki, w których następuje krok poprawy strategii lidera, oba są wyróżnione poprzez cieniowane tło. W ramach tych bloków stosowana jest procedura aktualizacji strategii, która jest wspólna dla obu tych kroków. Parametrem procedury jest funkcja oceny akcji, która różni się w tych dwu blokach, oznaczana dalej jako Q. Procedura wykonuje aktualizację dla każdego ze zbiorów informacyjnych z osobna. Najpierw zostanie zaprezentowany sposób poprawy rozkładu prawdopodobieństwa w pojedynczym zbiorze informacyjnym, a następnie procedura poprawy całości, która przechodzi całe drzewo strategii i wykonuje aktualizację rozkładu prawdopodobieństwa w każdym z węzłów.

Aktualizacja prawdopodobieństw w pojedynczym węźle jest opisana w Algorytmie 4.10. Procedura aktualizacji przechowuje dla każdego z węzłów strategii behawioralnej dwa elementy, które są przenoszone pomiędzy wywołaniami Algorytmu 4.10. Wartości te są usuwane dopiero przy próbkowaniu kolejnej strategii naśladowcy π_f^r w zewnętrznej pętli *O2-UCT*. Te dwa elementy to wektor momentu (bezwładności) $momentum_N$ oraz współczynnik normalizacji $normalizer_N$. Obie te wartości są początkowo zainicjowane zerami (wektor ma zera

na wszystkich współrzędnych, a współczynnik normalizacji jest zerowy). N jest węzłem drzewa strategii behawioralnej, wartości są inicjowane dla każdego węzła z osobna. Algorytm aktualizacji prawdopodobieństw działa w trzech etapach:

ekspansja Na etapie inicjacji strategii opisanej powyżej, strategia składa się z jednego ruchu wybranego w każdym węźle, więc efektywnie jest strategią prostą. W trakcie działania metody, przy każdej aktualizacji, z prawdopodobieństwem $expProb$ dodawany jest do drzewa strategii behawioralnej nowy ruch. Poddrzewo poniżej nowododanego ruchu jest rozwianej w taki sam sposób, jak w przypadku inicjowania pierwszej strategii. Ekspansja opisana jest w liniach 3—9 i jest analogiczna do procedury inicjowania strategii w Algorytmie 4.9. W przypadku, gdy w węźle N są już wszystkie ruchy możliwe w grze, ekspansja nie jest wykonywana w ogóle.

ocena ruchów W linii 15 dla każdego ruchu znajdującego się w drzewie strategii behawioralnej dokonywana jest ocena ruchu. Wyliczane są wszystkie stany terminalne, które mogą być osiągnięte po zagraniu danego ruchu, przy uwzględnieniu strategii naśladowcy i niedeterminizmu w grze. Dla każdego ze stanów terminalnych wyliczana jest wartość funkcji $Q(z)$, a na koniec wyliczana jest wartość oczekiwana $\mathbb{E}Q(z)$ po wszystkich osiągalnych węzłach terminalnych z, według rozkładu prawdopodobieństwa wynikającego z niedeterminizmu gry. Te oceny są dopisywane do wektora zbierającego oceny ruchu. Ocena zapisywana do wektora jest różnicą oceny bieżącego węzła i oceny, gdy zawsze byłby grany zadany ruch[2].

aktualizacja strategii Wyliczone oceny ruchów nie są bezpośrednio używane do poprawy wartości prawdopodobieństw poszczególnych ruchów w rozkładzie. Zamiast tego stosowany jest wektor bezwładności (momentu) znany z wielu metod optymalizacji. Wektory ocen ze wszystkich historycznych przebiegów są zsumowane w wektorze momentu w linii 18. To daje wypadkowy kierunek poprawy strategii. To podejście jest istotne, ze względu na fakt, że są naprzemiennie wykonywane operacje poprawy wypłaty lidera i operacje mające na celu powrót do obszaru dopuszczalnego, które mogą być działaniami w przeciwnych kierunkach. Dzięki zastosowaniu momentu, przeciwstawne działanie jest wygładzone, co daje szybszą poprawę strategii. Aby zapobiec rośnięciu wartości poprawek w nieskończoność, utrzymywany jest również współczynnik normalizacji, w którym sumowane są normy L_1 wszystkich dodawanych poprawek (linia 19). Następnie wyliczane są nowe współczynniki poprzez dodanie do prawdopodobieństwa wartości momentu podzielonego przez normalizator. Tak wyliczone współczynniki mogą nie dawać w efekcie rozkładu prawdopodobieństwa, bo część z nich może być ujemna, a całość nie musi sumować się do 1. W związku z tym w linii 22 wywoływana jest procedura, która: zamienia wszystkie ujemne współczynniki na 0, potem wykonuje normalizację wektora

[2]Miara ta bywa określana jako *Counterfactual Regret* [64].

tak, aby elementy sumowały się do 1. Jeśli normalizacja nie jest możliwa, bo wszystkie pozycje są zerowe, to wektor zastępowany jest wektorem reprezentującym rozkład jednostajny. Tak uzyskany wektor współczynników jest użyty jako nowy rozkład prawdopodobieństwa ruchów w węźle N.

Algorytm 4.10: Aktualizacja rozkładu prawdopodobieństwa w pojedynczym węźle drzewa strategii behawioralnej.

wejście: N — Węzeł drzewa reprezentującego strategię behawioralną
wejście: Q — Funkcja oceniająca węzeł terminalny gry

```
1  rand U(0,1);
2  if rand < expansionProb ∧ expansionPossible(N) then
3  |  moveToAdd ← calculateMoveToAdd(I, (s, p), π_f, getMovesInTree(N), Q);
4  |  addNewMove(N, moveToAdd);
5  |  setMoveProbability(N, moveToAdd, 0);
6  |  succ ← findLeadersSuccessors((I, (s, p)), moveToAdd, π_f);
7  |  for (I', (s', p')) ∈ succ do
8  |  |  N' ← subtree(N, moveToAdd, I');
9  |  |  ExpandStrategy(N', (I', (s', p')), π_f);
10 |  end
11 end
12 nodeAssessment ← expectedSubtreeValue(N, π_f, Q)// Wartość
       oczekiwana wypłaty poddrzewa gdy zostanie zagrana
       aktualna strategia w tym węźle
13 assessments ← [];
14 for move ∈ getMovesInTree(N) do
15 |  assmt ← expectedSubtreeValue(N, move, π_f, Q)// Wartość oczekiwana
   |     wypłaty poddrzewa gdy zostanie zagrany ruch move w
   |     tym węźle
16 |  append(assessments, assmt − nodeAssessment);
17 end
18 momentum_N ← momentum_N + assessments;
19 normalizer_N ← normalizer_N + L_1(assessments);
20 probabilities ← getProbabilitiesVector(N);
21 probabilities ← probabilites + momentum_N/normalizer_N;
22 probabilities ← makeProbabilityDistribution(probabilities);
23 setProbabilitiesVector(N, probabilities);
```

Cały krok poprawy strategii lidera polega na przejściu całego drzewa strategii lidera w kolejności od liści do korzenia (rodzic zawsze jest przetwarzany po wszystkich potomkach) i zaaplikowaniu powyższej procedury w każdym przetwarzanym węźle.

Aktualizacja strategii w celu poprawy wypłaty lidera gdy gra przeciwko π_f^r. W sytuacji, gdy nie znaleziono żadnej strategii naśladowcy, która jest lepszą odpowiedzią niż π_f^r (blok poprawy strategii z lewej strony Rysunku 4.13) powyższa procedura jest aplikowana z następującymi parametrami:

- jako strategia π_f używana do wyznaczenia osiągalnych zbiorów informacyjnych jest użyta strategia π_f^r,
- jako funkcja Q użyta jest funkcja, która wykorzystuje wartość wypłaty lidera przypisaną do danego stanu terminalnego.

Aktualizacja strategii w celu przybliżenia do obszaru dopuszczalnego. W sytuacji, gdy znaleziono lepszą odpowiedź naśladowcy, oznaczaną w tym opisie π_f^b, celem jest zmniejszenie wypłaty naśladowcy, gdy grana jest strategia π_f^b, a zwiększenie wypłaty, gdy grana jest π_f^r. W tym celu zaprezentowana wyżej procedura jest nieznacznie modyfikowana. Zamiast pojedynczego wektora stanów $\mathbf{s}$ i prawdopodobieństw $\mathbf{p}$ przechowywane są pary wektorów $(\mathbf{s}_r, \mathbf{s}_b)$ oraz $(\mathbf{p}_r, \mathbf{p}_b)$, odpowiednio pierwsze elementy każdej z par są wyliczane przy założeniu, że naśladowca gra π_f^r, a drugie elementy są wyliczane przy założeniu, że grana jest strategia π_f^b. W związku z tym cała procedura wyliczania stanów następujących po ruchu i w efekcie docierania do stanów terminalnych jest wykonywana osobno dla każdej ze strategii. Dla obu strategii naśladowcy obliczany jest również osobny zestaw węzłów terminalnych. Funkcja Q, jest tym razem również definiowana w kontekście dwóch strategii naśladowcy i jest definiowana jako różnica pomiędzy wypłatą naśladowcy, gdy jest grana strategia π_f^r i wypłatą naśladowcy, gdy grana jest strategia π_f^b.

Warunek stopu. Warunkiem stopu użytym w eksperymentach z *O2-UCT* jest alternatywa logiczna trzech sytuacji:

- wykonano $positiveStepLimit$ kroków poprawy wypłaty lidera, gdy strategia jest grana przeciwko π_f^r,
- wykonano $feasibilityStepLimit$ kroków poprawy w kierunku obszaru dopuszczalnego i cały czas istnieje jakaś strategia π_f^b,
- wykonano $poorImprovementLimit$ kroków poprawy wypłaty lidera, a wypłata najlepszej znanej strategii poprawiła się w tym czasie o nie więcej niż $poorImprovement\varepsilon$.

Wszystkie powyższe wartości są parametrami metody.

4.3.3 Sposoby przyspieszenia obliczeń

Zaprezentowana powyżej metoda poszukiwania strategii lidera posiada dodatkowe usprawnienia, które zostały pominięte w głównym opisie, aby zwiększyć jego czytelność. Metoda w formie prezentowanej powyżej jest jednak zbyt wolna do stosowania w praktyce i poniższe optymalizacje są jej integralną częścią (choć należy je traktować jako szczegóły implementacyjne, a nie część koncepcji). Te elementy to:

repertuar ostatnio znalezionych strategii naśladowcy Kosztowną operacją jest stwierdzenie, czy istnieje strategia naśladowcy π_f^b, taka, że $u_f(\delta_l, \pi_f^r) < u_f(\delta_l, \pi_f^b)$. Wymagany jest tu pełny przegląd wszystkich strategii prostych naśladowcy, których liczba rośnie wykładniczo wraz z długością gry. Aby przyspieszyć ten mechanizm, stosowany jest wektor, w którym pamiętane jest $histLimit$ ostatnio znajdowanych strategii naśladowcy. Metoda poszukiwania strategii naśladowcy w pierwszej kolejności przegląda strategie z wektora i po natrafieniu na pierwszą, która daje wypłatę wyższą niż π_f^r, zwraca ją jako wynik. Dopiero jeśli w wektorze nie znaleziono strategii, to wykonywany jest pełny przegląd. Początkowo ten wektor jest pusty, w momencie znalezienia strategii spełniającej warunki jest ona dodawana do wektora. Jeśli w wektorze jest już $histLimit$ strategii, to najstarsza strategia jest usuwana.

próg różnicy wypłat strategii naśladowcy Drugim problemem związanym z poszukiwaniem strategii π_f^b takiej, ze $u_f(\delta_l, \pi_f^r) < u_f(\delta_l, \pi_f^b)$ jest to, że wybierana jest pierwsza strategia, spełniająca warunek. W wielu przypadkach będzie tak, że zostanie wybrana strategia, gdzie różnica wypłat naśladowcy pomiędzy π_f^b i π_f^r jest bardzo mała, podczas, gdy istnieją kandydaci na π_f^b, gdzie różnica jest dużo wyższa. Takie podejście powoduje bardzo dużą liczbę kroków poprawy strategii lidera, które mają wykonać zmianę w kierunku obszaru dopuszczalnego, jednak jest to minimalna wartość kroku. Dużo lepsza jest sytuacja, gdzie różnica wypłat jest duża. W tego punktu widzenia, najlepszym rozwiązaniem byłby wybór najlepszej odpowiedzi naśladowcy. To podejście uniemożliwiłoby jednak korzystanie z wektora przechowującego historyczne strategie naśladowcy. W związku z tym przyjęto rozwiązanie pośrednie, w którym przegląd jest wykonywany do natrafienia na pierwszą strategię, dla której $u_f(\delta_l, \pi_f^r) - u_f(\delta_l, \pi_f^r) > minEarlyStopDiff$. Jeśli nie natrafiono na żadną strategię spełniającą ten warunek, to zwracana jest strategia, która maksymalizuje tę różnicę.

zapamiętywanie wyników pośrednich Procedura poprawy rozkładu prawdopodobieństwa opisana w Algorytmie 4.10 wykorzystuje wartości oczekiwane wypłat graczy w poddrzewach gry. Każdorazowe wyliczanie tych wartości dla dużych gier może być kosztowne, a wartości wyliczone w niższych węzłach mogą być wykorzystane do obliczenia wartości w węzłach powyżej. W implementacji metody te wartości oczekiwane są zapamiętywane i wykorzystywane ponownie przy wyliczaniu wartości dla wyższych części drzewa.

odcinanie w drzewie strategii behawioralnej elementów o zerowym prawdopodobieństwie We wstępnych testach metody *O2-UCT* często obserwowanym zjawiskiem było to, że do strategii lidera były dodawane pewne ruchy, a następnie, po pewnej liczbie iteracji, gdy zostało dodane więcej ruchów, prawdopodobieństwo zagrania tych ruchów dodanych wcześniej było zmniejszane do 0 i pozostawało takie już do końca. W związku z tym, żeby oszczędzić pamięć oraz przyspieszyć przeglądanie drzewa, zaimplementowany został mechanizm, który usuwa z drzewa strategii ruchy i związane z nimi poddrzewa, jeśli

te przez $zeroProbIterationsLimit$ iteracji z rzędu prawdopodobieństwo danego ruchu było 0.

4.3.4 Wyniki eksperymentalne

Skuteczność działania metody *O2-UCT* została sprawdzona eksperymentalnie. Zbadano, podobnie jak w przypadku *Mixed-UCT* czas potrzebny do znalezienia strategii mieszanej lidera oraz wartość oczekiwaną wypłaty, gdy znaleziona strategia lidera jest grana przeciwko najlepszej odpowiedzi naśladowcy na tę strategię. Metoda *O2-UCT* została porównana z metodami, które można stosować do ogólnej klasy wielokrokowych gier Stackelberga z niepełna informacją o sumie niezerowej, znanymi z literatury. Eksperymenty zostały przeprowadzone na trzech różnych zbiorach gier. Wyniki przedstawiono w formie średnich zagregowanych po rozmiarze gier testowych.

W testach zostały użyte następujące zbiory gier:

- Warehouse Games (WHG), opisany w Rozdziale 4.1.1, z liczbą rund $T = 3,4,5,6,7,8$
- WHG non-zero sum (WNZ), również opisany w Rozdziale 4.1.1, z liczbą rund $T = 3,4,5,6,7$
- Search Games (SEG), opisany w Rozdziale 3.9.1, z liczbą rund $T = 4,5$.

Repertuar metod, z którymi porównywana była metoda *O2-UCT* jest szerszy niż w przypadku metody *Mixed-UCT*. Wynika to z faktu, że te eksperymenty były przeprowadzone w roku 2019, 3 lata później, niż w przypadku *Mixed-UCT* i w literaturze pojawiły się metody nieznane wcześniej. Metoda została porównana z następującymi metodami z literatury, w tym dwiema, które znajdują dokładne rozwiązanie:

- *BC2015* [17], opisana w Rozdziale 3.7.1, metoda ta wykorzystuje postać sekwencyjną gry i rozwiązuje jeden program mieszany program liniowy. Ta metoda znajduje dokładne rozwiązanie.
- *C2016* [87], opisana w Rozdziale 3.7.2, metoda ta wykorzystuje postać sekwencyjną gry oraz rozwiązuje wiele iteracyjnie poprawianych programów liniowych. Ta metoda znajduje dokładne rozwiązanie.
- *CBK2018* [95], opisana w Seckji 3.7.3, metoda ta opiera się o zwijanie dużych poddrzew gry do pojedynczych węzłów, rozwiązywanie gry uproszczonej metodą *C2016*, a następnie rozwijania poddrzew związanych z węzłami, które mają największy wpływ na wynik uproszczonej gry.

Wszystkie eksperymenty zostały przeprowadzone na komputerze z procesorem Intel Xeon Silver 4116 @ 2.10GHz i 256GB RAM działającym pod kontrolą systemu Debian GNU/Linux.

W przypadku metod wykorzystujących programowanie liniowe został użyty solwer Cplex. Metoda *O2-UCT* została zaimplementowana w języku Scala i uruchomiona na maszynie wirtualnej Java w wersji 8.

Każdy pojedynczy przebieg poszukiwania strategii, dla każdej z metod był uruchomiony z limitem czasowym 200h, po upływie tego czasu proces był przerywany. W przypadku metod korzystających z solwera MILP (wszystkich metod pochodzących z literatury), uruchamiano jeden proces na raz, w związku z tym proces miał dostęp do całości pamięci operacyjnej (z dokładnością do części zajmowanej przez system). W przypadku metody *O2-UCT* eksperymenty były uruchamiane równolegle, 32 na raz. Wszystkie metody były jednowątkowe. Wspomniana konfiguracja sprzętowa ma w sumie 48 wątków procesora, w związku z czym w przypadku równoległych uruchomień nie było problemu z brakiem niezajętego rdzenia CPU. W przypadku metody *O2-UCT*, której wyniki zależą od działania generatora liczb pseudolosowych, wykonano 10 niezależnych przebiegów dla każdej z gier ze zbioru testowego. W przypadku pozostałych metod uruchamiany był tylko jeden przebieg. Początkowo było planowane uruchomienie eksperymentów dla pozostałych metod również w sposób równoległy. Okazało się jednak, że dla dużych gier programy liniowe konstruowane przez te metody wymagają ponad 128GB pamięci operacyjnej do rozwiązania. Stąd ostatecznie uruchomiono te eksperymenty sekwencyjnie, jeden po drugim.

Parametry metod. Metody *BC2015* i *C2016* nie mają żadnych parametrów. Metoda *CBK2018* ma dwa parametry związane z tym, które poddrzewa będą rozwijane, ϵ i δ, im mniejszy parametr ϵ, tym więcej poddrzew będzie rozwiniętych (co daje dokładniejszy wynik, ale i większy czas obliczeń). Zgodnie z sugestiami w pracy autorów metody *CBK2018* [95], użyto zestawu $\epsilon = 0.4, \delta = 0.4$. Dodatkowo sprawdzono wariant dokładniejszy, z wartościami $\epsilon = 0.0, \delta = 0.4$. Pierwszy zestaw parametrów jest oznaczony w wynikach jako *CBK2018(a)*, a drugi odpowiednio *CBK2018(b)*. Parametry metody *O2-UCT* zostały wymienione w opisie tej metody. Tabela 4.11 prezentuje parametry użyte w trakcie eksperymentów. Wartości tych parametrów zostały wybrane poprzez przeprowadzenie testów na zbiorze WHG, na małą skalę, dla wielu różnych zestawów.

Prezentacja wyników. Wyniki prezentowane są osobno dla każdego z trzech zbiorów testowych. Na oddzielnych wykresach prezentowany jest czas obliczeń w zależności od rozmiaru gry i średnia wypłata lidera w zależności od rozmiaru gry. Wykresy prezentują wyniki dla zagregowanych rozmiarów gry (na osi X). Agregacja przebiegała w następujący sposób: dla każdej z gier wyliczono liczbę węzłów z jakiej składa się drzewo tej gry w postaci ekstensywnej, ozn. s, następnie zostało wykonane zaokrąglenie do najbliższej potęgi liczby 10 na skali logarytmicznej:

$$10^{round(\log_{10} s)},$$

Tabela 4.11: Zestaw parametrów metody *O2-UCT* użyty we wstępnych eksperymentach.

Parametr	Opis	Wartość
$outerSimulationCount$	liczba odwiedzin ruchów w próbkowaniu w zewnętrznej pętli UCT po której algorytm się kończy	30
$feasibilityStepLimit$	liczba kroków poprawy strategii lidera mających za zadanie osiągnąć obszar dopuszczalny, po której procedura jest przerywana, jeśli nie osiągnięto obszaru dopuszczalnego	10000
$positiveStepLimit$	limit kroków poprawy strategii lidera tak, aby zwiększyć wypłatę lidera, po której procedura się kończy	5000
$poorImprovementLimit$	liczba kroków poprawy lidera, przy której nastąpiła za mała poprawa wypłaty, po której nastepuje wczesne zatrzymanie metody	500
$poorImprovement\varepsilon$	minimalna wartość poprawy wypłaty naśladowcy, przy której nie następuje wczesne zarzymanie	10^{-5}
C	współczynnik eksploracji z wzoru (4.1) wykorzystywany w algorytmie UCT	1.4
$expansionProb$	prawdopodobieństwo dodania nowego ruchu do strategii behawioralnej w trakcie poprawy węzła strategii	0.3
$addMoveUctSimulations$	liczba symulacji UCT używana w Algorytmie 4.8	300

gdzie $round$ oznacza zaokrąglenie do najbliższej liczby całkowitej. Tak zaokrąglone liczby węzłów w drzewie gry posłużyły do agregacji wyników. Wartości wyników, zarówno czasu, jak i wypłat lidera, dla wszystkich gier, dla których zaokrąglona liczba węzłów była taka sama, zostały uśrednione. Na wykresie średniemu wynikowi dla wszystkich gier z takiego kubełka są prezentowane jako jeden punkt. Ponadto wszystkie przebiegi były uruchamiane z limitami, 200 godzin czasu działania i przy pamięci 256GB. Dla części metod obliczenia dla większych gier były przerwane. W takiej sytuacji:

- w przypadku prezentacji wyników dotyczących czasu obliczeń w miejsce wartości było wstawiane 200 godzin (limit czasu), w przypadku, gdy w dla danej uśrednionej wartości liczby węzłów dla danej metody nie udało się obliczyć ani jednego przypadku, punkt na wykresie był pomijany.

- w przypadku prezentacji średnich wypłat, najpierw zostały wyznaczone podzbiory gier, które wpadają do danej średniej liczby węzłów. Osobno dla każdego zbioru testowego. Następnie zostały odrzucone wszystkie punkty, dla których udało się policzyć mniej niż 70% strategii dokładnych (metodą *BC2015* lub *C2016*). Na wykresie zostały zaprezentowane wyniki tylko dla tych gier, dla których były dostępne strategie dokładne. W przypadku, gdy którąś z metod nie udało się obliczyć wszystkich strategii, dla których zostały policzone dokładne wyniki, taki punkt nie jest prezentowany na wykresie.

Tabela 4.12: Liczba gier WHG rozwiązanych przez poszczególne metody w podziale na rozmiar gry. Rozmiary gier są podane jako etykiety kolumn.

Metoda	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
O2-UCT	1	25	25	25	25	25	22	2
BC2015	1	25	25	25	24	0	0	0
C2016	1	25	25	25	24	0	0	0
CBK2018(a)	1	25	25	25	25	22	0	0
CBK2018(b)	1	25	25	25	25	22	0	0
Liczba gier	1	25	25	25	25	25	22	2

Tabela 4.13: Liczba gier WNZ rozwiązanych przez poszczególne metody w podziale na rozmiar gry. Rozmiary gier są podane jako etykiety kolumn.

Metoda	10^3	10^4	10^5	10^6	10^7	10^8
O2-UCT	25	25	23	30	34	7
BC2015	25	25	23	23	4	0
C2016	25	25	23	22	15	0
CBK2018(a)	25	25	23	22	15	2
CBK2018(b)	25	25	23	20	0	0
Liczba gier	25	25	23	30	34	7

W tabelach 4.12, 4.13 i 4.14 są prezentowane liczby gier odpowiednio ze zbioru WHG, WNZ, SEG, z poszczególnych przedziałów, dla których udało policzyć się strategie w limicie czasu i pamięci z użyciem poszczególnych metod. Na początku warto zwrócić uwagę na fakt, że agregacja gier według liczby węzłów w drzewie gry nie do końca pokrywa się z liczbą rund w grze, w szczególności nie wszystkie przedziały w grach WHG i WNZ mają po 25 elementów. Wynika to z faktu, że na liczbę węzłów w drzewie gry wpływa zarówno liczba rund, jak i struktura grafu. W przypadku gier SEG, gdzie nie ma różnorodnej struktury grafów, a jedynie zróżnicowanie wypłat, które nie wpływa na liczbę węzłów w drzewie można zaobserwować zupełny brak gier, których liczba węzłów została zaokrąglona do 10^5. Następną obserwacją jest fakt, że jedynie metoda *O2-UCT* była w stanie rozwiązać wszystkie gry testowe. Bardziej szczegółowe obserwacje można poczynić dla poszczególnych zbiorów testowych z osobna. W przypadku liczby rozwiązanych gier ze zbioru WHG, prezentowanej w Tabeli 4.12 wszystkie metody sa w stanie rozwiązać gry, z kubełka dla 10^5 węzłów i wszystkich mniejszych gier. W przypadku kubełka 10^6 metody *BC2015* oraz *C2016* nie były w stanie rozwiązać jednej instancji. W przypadku większych gier te dwie metody nie dały rozwiązać żadnej gry w narzuconych limitach czasu i pamięci. Metoda *CBK2018*, niezależnie od parametrów, była w stanie rozwiązać wszystkie gry z kubełka 10^6. W przypadku kubełka 10^7 nie udało się rozwiązać 3 gier. W przypadku kubełków 10^8 i 10^9 jedynie metoda *O2-UCT* była w stanie obliczyć strategię lidera. W wykresu dotyczącego gier WNZ obserwacje są podobne, mniejsze gry, do kubełka 10^5 włącznie, zo-

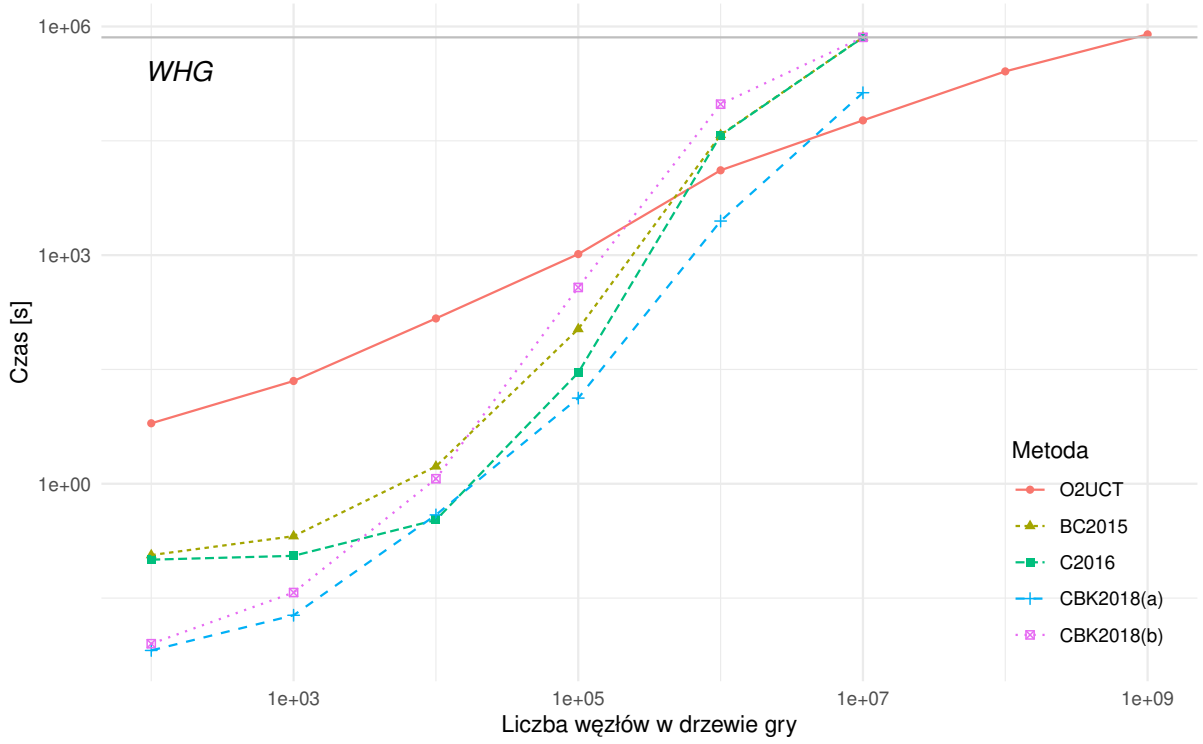
Tabela 4.14: Liczba gier SEG rozwiązanych przez poszczególne metody w podziale na rozmiar gry. Rozmiary gier są podane jako etykiety kolumn.

Metoda	10^4	10^5	10^6	10^7
O2-UCT	10	–	10	10
BC2015	10	–	10	0
C2016	10	–	10	0
CBK2018(a)	10	–	10	5
CBK2018(b)	10	–	10	3
Liczba gier	10	0	10	10

stały obliczone przez wszystkie metody. W przypadku kubełka 10^6 metoda *BC2015* obliczyła 23 gry z 30, a metody *C2016* i *CBK2018* były w stanie obliczyć jeszcze mniejszą liczbę gier. Z drugiej strony, w przypadku kubełka 10^7 więcej, bo aż 15 z 34 gier obliczyły metody *C2016* i *CBK2018(a)*. Pokazuje to, że koszt obliczenia gry zależy nie tylko od liczby węzłów w drzewie gry, ale również od innych cech struktury gry, które mogą przyspieszyć lub spowolnić obliczenia z użyciem różnych metod. Można również zaobserwować, ze wariant *CBK2018(b)*, który powinien znajdować bardziej dokładne rozwiązania, rozwiązuje istotnie mniej dużych gier niż wariant *CBK2018(a)*. W przypadku gier SEG, dla których liczba rozwiązanych gier przedstawiona jest w Tabeli 4.14, ze względu na dużo mniejsze zróżnicowanie wielkości gier, jedyne różnice pomiędzy metodami można zaobserwować tylko w kubełku 10^7. Jedynie metoda *O2-UCT* rozwiązuje tu wszystkie 10 gier. Metody dokładne nie były w stanie rozwiązać ani jednej gry z tego kubełka. W przypadku *CBK2018(a)* udało się rozwiązać 5 gier, a w przypadku wariantu *CBK2018(b)* — 3 gry.

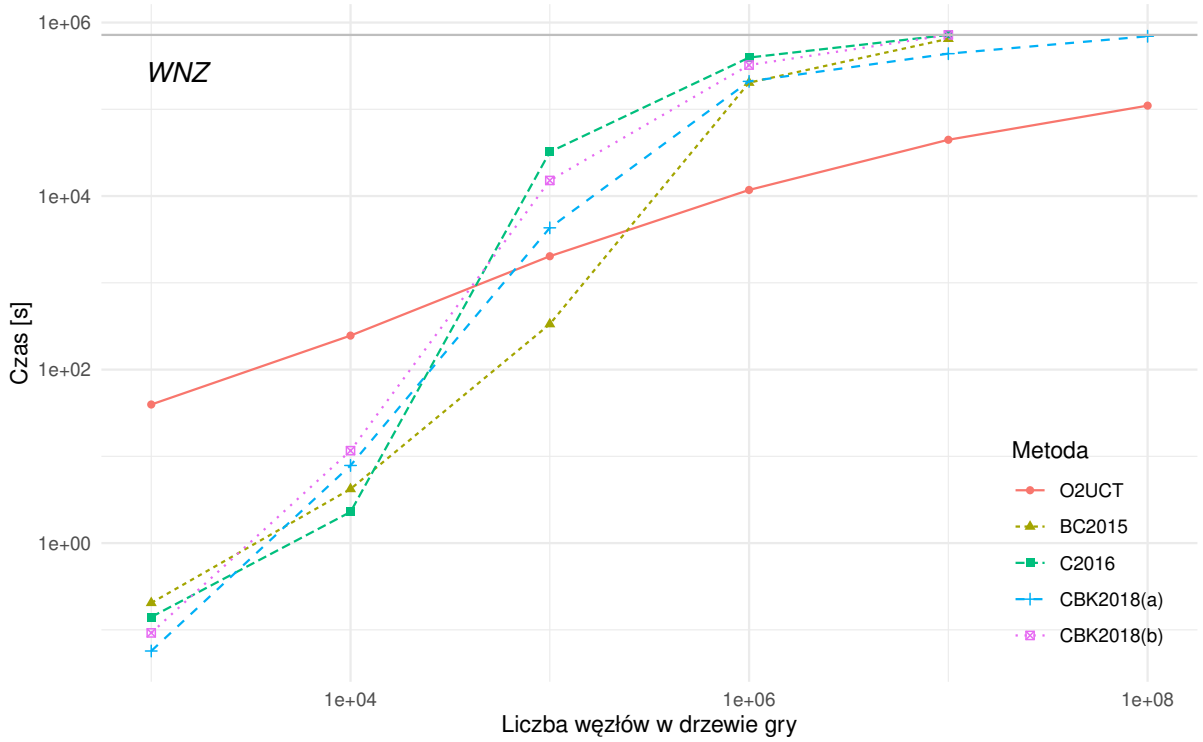
Czas obliczeń

Rysunki 4.14, 4.15 i 4.16 przedstawiają średni czas obliczeń dla poszczególnych rozmiarów gier, dla każdej z metod. Szara pozioma linia oznacza limit czasu, po którym obliczenia były przerywane: 200 godzin. W przypadku gier *WHG*, dla których czasy są prezentowane na Rysunku 4.14, można zaobserwować, że dla gier z kubełka 10^5 i mniejszych *O2-UCT* jest najwolniejszą metodą. Różnica miedzy *O2-UCT*, a pozostałymi metodami jest największa dla najmniejszych gier i maleje wraz ze zbliżaniem się do kubełka 10^5. Dla gier w kubełku 10^6 metoda *O2-UCT* wylicza strategię szybciej, niż pozostałe metody, za wyjątkiem *CBK2018(b)*, która jest tylko nieco szybsza. W przypadku kubełka 10^7 metoda *O2UCT* jest najszybszą ze wszystkich, a dla większych kubełków *O2-UCT* była jedyną, która była w stanie policzyć wynik. Warto też zwrócić uwagę na fakt, że czas obliczeń dla wszystkich metod wykorzystujących programowanie liniowe czas obliczeń rośnie dużo szybciej, niż w przypadku metody *O2-UCT*. W oczy rzuca się też mniejszy przyrost czasu pomiędzy kubełkami 10^6 i 10^7 w porównaniu z przyrostem pomiędzy 10^5 i 10^6 dla metod *BC2015*, *C2016* i *CBK2018*. Jest to spowodowane, tym, że dla części przebiegów proces został przerwany ze względu na limit czasu i wynik został

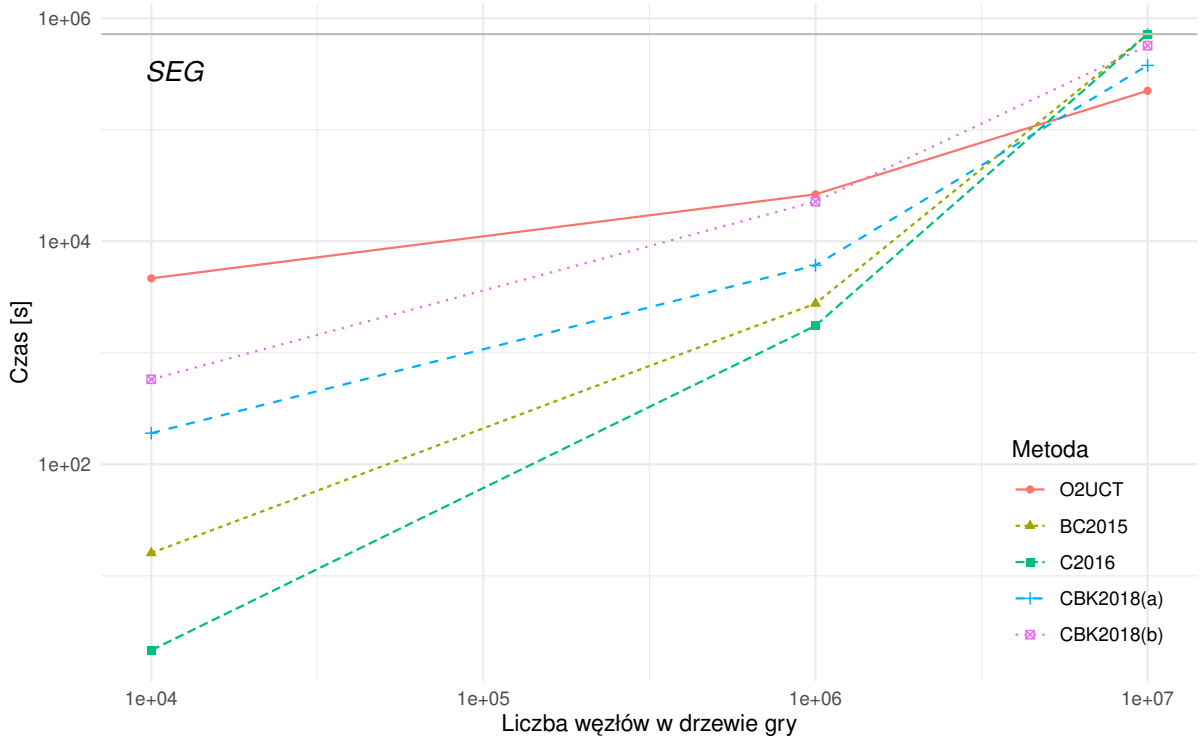


Rysunek 4.14: Średni czas obliczeń dla gier ze zbioru WHG dla metody *O2-UCT* i wszystkich metod z nią porównywanych.

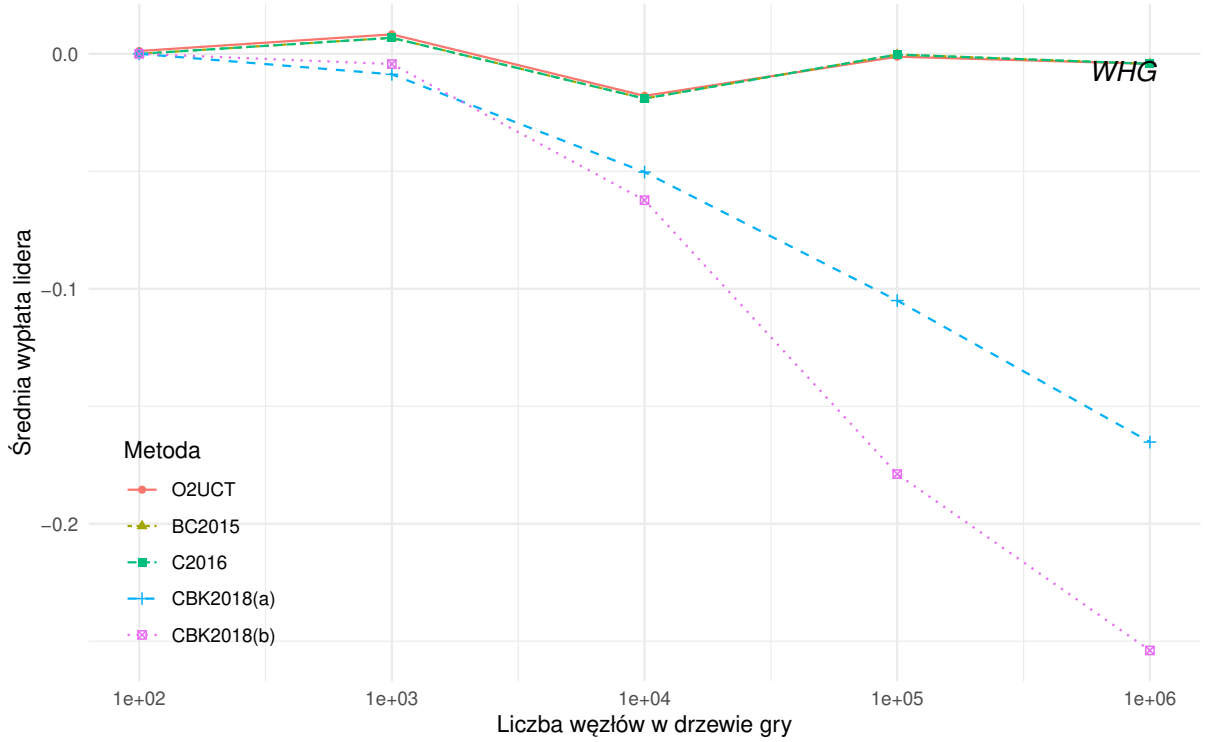
zastąpiony przez 200 godzin, podczas gdy faktyczny cas były większy. Można tez zaobserwować, że jeśli pominiemy *O2-UCT*, to najszybszą z metod jest *CBK2018(a)*, co jest zgodne z intuicją, jako, że *CBK2018* jest metodą przybliżoną, a zestaw parametrów oznaczony jako (a) wymaga mniej rozwinięć węzłów w drzewie gry. W przypadku czasów obliczeń dla gier WNZ, zaprezentowanych na Rysunku 4.15, obserwacje dotyczące metody *O2-UCT* są podobne, jak w przypadku WHG. Dla małych gier *O2-UCT* jest metodą istotnie wolniejszą od pozostałych. Tym razem kubełek 10^5 jest granicznym punktem, gdzie metoda *O2-UCT* jest szybsza od większości z konkurencyjnych metod. Tym razem w kubełku 10^5 szybsza od *O2-UCT* jest jedynie metoda *BC2015*. Dla kubełka 10^6 i większych *O2-UCT* jest najszybszą z metod. Fakt, że dla tego zbioru gier Punkt w którym *O2-UCT* jest najszybsza ze wszystkich następuje wcześniej jest prawdopodobnie związany z tym, że zbiór WNZ ba bardziej skomplikowaną strukturę wypłat niż WHG. Podobnie, jak poprzednio można zaobserwować efekt „wypłaszczenia" czasów metod innych niż *O2-UCT* powyżej kubełka 10^5. Ponownie wynika to z faktu osiągnięcia limitu czasu obliczeń. Również podobnie, jak w poprzednim przypadku, przyrost czasu obliczeń wraz ze wzrostem rozmiaru gry w przypadku *O2-UCT* jest dużo mniejszy, niż w przypadku pozostałych metod. Ostatnim ze zbiorów testowych jest SEG. Wyniki dla tego zbioru są przedstawione na Rysunku 4.16. Ze względu na mniejszą liczbę punktów na wykresie, przewaga *O2-UCT* nie jest aż tak wyraźna, jak w poprzednich przypadkach, jednak cały czas dla kubełka 10^7, czyli największego, *O2-UCT* jest najszybszą z metod. W przypadku pozostałych metod można zaobserwować fakt, że metoda *CBK2018(a)* jest najwolniejsza ze wszystkich metod innych niż



Rysunek 4.15: Średni czas obliczeń dla gier ze zbioru WNZ dla metody *O2-UCT* i wszystkich metod z nią porównywanych.



Rysunek 4.16: Średni czas obliczeń dla gier ze zbioru SEG dla metody *O2-UCT* i wszystkich metod z nią porównywanych.

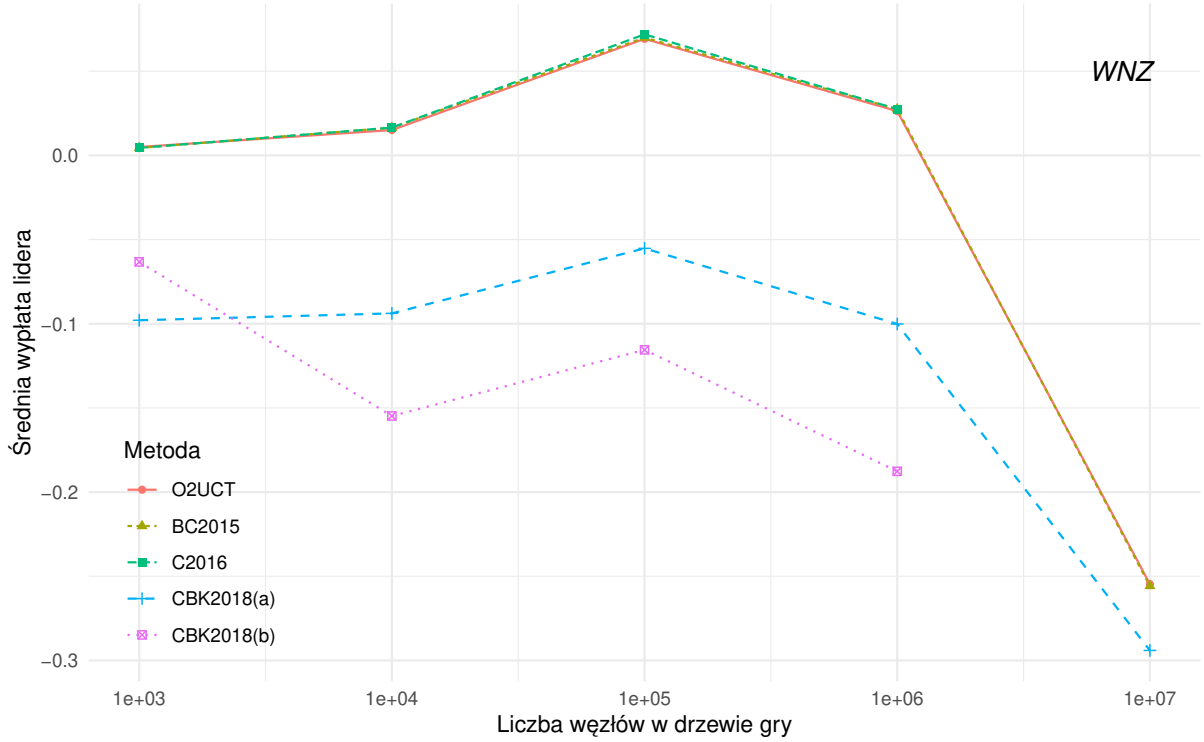


Rysunek 4.17: Średnia wypłaya lidera dla gier ze zbioru WHG dla metody *O2-UCT* i wszystkich metod z nią porównywanych.

O2-UCT, co nie miało miejsca w dwóch poprzednich zbiorach testowych.

Wartość oczekiwana wypłaty lidera

Na Rysunkach 4.17, 4.18 i 4.19 prezentowane są średnie wartości wypłaty lidera dla strategii znajdowanych przez poszczególne metody. Na wstępie należy zauważyć, ze w przeciwieństwie do czasów obliczeń, gdzie naturalne jest, że dla większych gier czas obliczeń tez będzie większy, w tym przypadku wartości oczekiwane wypłaty nie muszą mieć związku z rozmiarem gry. Wykresy prezentują wartości dla dokładnego rozwiązania (metody *BC2015* i *C2016* jako wynik dają dokładnie stan Równowagi Stackelberga). Punkty dla obu tych metod, w związku z powyższym, zawsze się pokrywają. Wyniki pozostałych metod nie mogą być większe niż wynik dokładny. Na Rysunku 4.17 prezentowane są wartości średnie wypłaty lidera dla zbioru WHG. Można zaobserwować, że wartości *O2-UCT* praktycznie pokrywają się z wartościami dla metod wyliczających dokładną strategię. W przypadku metody przybliżonej zaczerpniętej z literatury — *CBK2018* — w obu wariantach konfiguracji wypłaty lidera są słabsze. Co więcej różnica między wypłata optymalną, a zwracaną przez metodę, rośnie wraz ze wzrostem rozmiaru gry. W przypadku wyników dla gier WNZ, prezentowanych na Rysunku 4.18, ponownie można zaobserwować, ze wyniki metody *O2-UCT* są bardzo bliskie wynikom metod dokładnych. W przypadku kubełka 10^5 wartość wypłaty jest nieznacznie niższa, ale mimo to, w porównaniu z wartościami uzyskiwanymi przez strategie obliczone przez metodę *CBK2018*



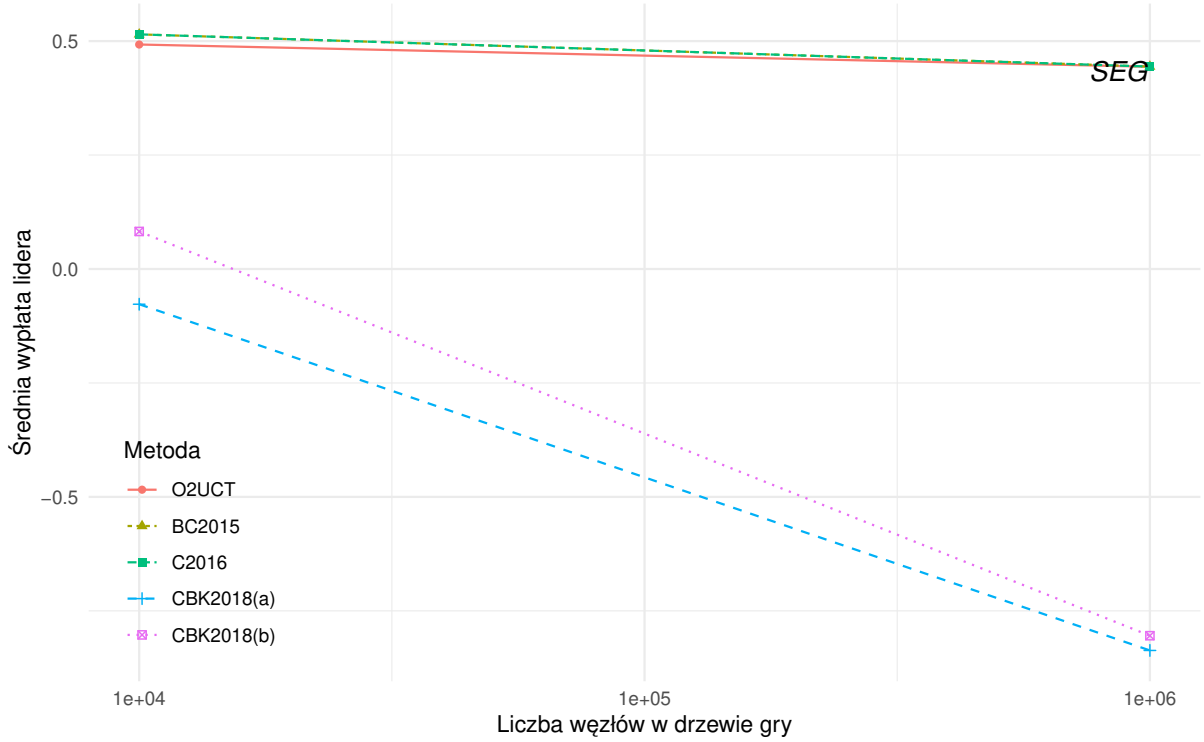
Rysunek 4.18: Średnia wypłaya lidera dla gier ze zbioru WNZ dla metody *O2-UCT* i wszystkich metod z nią porównywanych.

wynik jest bardzo dobry Ponownie metoda *CBK2018* w obu wariantach zrwaca wyniki istotnie gorsze niż pozostałe metody. Wyniki dla gier SEG, prezentowane na Rysunku 4.19, co do głównych trendów, nie odbiegają od tych zaobserwowanych dla wcześniejszych zbiorów gier. Ponownie metoda *O2-UCT* daje nieznacznie słabsze wyniki niż metody dokładne, a druga metoda przybliżona – *CBK2018*, w obu wariantach daje wyniki dużo słabsze.

Podsumowanie eksperymentów

Przeprowadzone eksperymenty pokazują, że metoda *O2-UCT* dla odpowiednio dużych gier pozwala znajdować strategie lidera dużo szybciej niż konkurencyjne metody, zarówno dokładne, jak i przybliżone. Skalowalność czasowa metody *O2-UCT* jest dużo lepsza niż w przypadku metod rozwiązujących programy liniowe. Metoda *O2-UCT* generuje przy tym strategie lidera, które dają wartości wypłat porównywalne z tymi, które są zwracane przez metody dokładne. Należy jednak zauważyć, ze, ze względu na to, ze nie są znane rozwiązania dokładne dla większych gier ze zbirów testowych, nie było możliwości zbadania jakości uzyskiwanych strategii dla wszystkich elementów zbiorów testowych, porównanie zostało wykonane tylko dla mniejszych gier. Drugim aspektem, na który należy zwrócić uwagę, jest to, że metody *BC2015* i *C2016*, coś są dużo wolniejsze i wymagają dużo więcej pamięci operacyjnej, gwarantują znalezienie dokładnego rozwiązania. Metoda *O2-UCT* nie daje takiej gwarancji.

Elementem, który nie był wprost mierzony w eksperymentach, a który można było zaobser-



Rysunek 4.19: Średnia wypłaya lidera dla gier ze zbioru SEG dla metody *O2-UCT* i wszystkich metod z nią porównywanych.

wować przy równoległym uruchamianiu wielu przypadków testowych na raz, było zapotrzebowanie na pamięć. W przypadku metod opartych o programowanie liniowe konieczne było sekwencyjne uruchamianie metod, ze względu na bardzo duże zużycie pamięci. Z drugiej strony metoda *O2-UCT* z powodzeniem rozwiązała wszystkie gry ze zbiorów testowych, gdy była uruchamiana równolegle i na jeden proces przypadało średnio 8GB pamięci RAM. Warto też zauważyć, ze *O2-UCT* w porównaniu z *CBK2018*, która też jest metodą przybliżoną, zwraca strategie lidera dające dużo większą wypłatę.

4.4 Ewaluacja proponowanych metod

W tym rozdziale zostały zaprezentowane dwie metody poszukiwania Równowagi Stackelberga wykorzystujące podejścia metaheurystyczne. Zarówno w przypadku *Mixed-UCT*, jak i w przypadku *O2-UCT* metody metaheurystyczne są w stanie rozwiązywać dużo większe gry niż podejścia oparte o programowanie liniowe. Ograniczenie metody klasycznych ma dwojakie źródło, po pierwsze jest to duże zapotrzebowanie na pamięć operacyjną. Metody oparte o programowanie liniowe muszą zbudować program liniowy reprezentujący całą grę. Ten program musi w całości zmieścić się w dostępnej pamięci operacyjnej. W związku z tym, że rozmiar gry rośnie wykładniczo wraz ze wzrostem liczby rund w grze. To oznacza, że rozmiary programów liniowych dla większych gier przekraczają dostępną pamięć, a co więcej, biorąc pod uwagę wykładniczy wzrost zapotrzebowania na pamięć, rozbudowa komputera o dodatkową

pamięć operacyjną nie jest pomocna. Niezależnie od tego, gdyby dało się ten program przechować w pamięci, czas rozwiązywania programu również rośnie wraz ze wzrostem rozmiaru, co oznacza, że czasy wykonania dla dużych gier są zaporowo duże. Widać to szczególnie na Rysunku 4.7, gdzie wykonano ekstrapolację czasów wykonania poza zakres gier, gdzie udało się przeprowadzić obliczenia. Bardzo szybko osiągane są czasy rzędu miesiąca i dłuższe, co uniemożliwia praktyczne zastosowanie tych metod. Metody *Mixed-UCT* i *O2-UCT*, dzięki temu, że próbkują tylko fragment drzewa gry, nie wymagają, aby przechować całą reprezentację gry na raz w pamięci. To pozwala na wielokrotnie mniejsze zużycie pamięci niż klasyczne metody.

Technicznie, obie metody zaimplementowane są z wykorzystaniem funkcji symulujących grę. Implementacja każdej z metod oczekuje gry opisanej w postaci funkcji, która dla danego stanu gry wylicza stany, które są efektem zagrania danego ruchu. Ten sposób implementacji gry zużywa niewielką ilość pamięci operacyjnej i pozwala metodom działać w sposób efektywny.

Nie należy też zapominać, że metody metaheurystyczne nie dają gwarancji otrzymania optymalnego rozwiązania. W przypadku *Mixed-UCT* wartości oczekiwane wypłat dla znalezionych strategii są w części przypadków znacząco mniejsze niż wypłat dla strategii ze stanu Równowagi Stackelberga. Strategie obliczane przez metody *O2-UCT* są bardzo bliskie wartościom zwracanym przez metody dokładne. Biorąc pod uwagę fakt, że dla dużych gier metoda *O2-UCT* jest dużo szybsza, należy traktować to podejście jako ważną i przydatną alternatywę dla metod dokładnych.

Porównując metody *Mixed-UCT* i *O2-UCT* między sobą należy przede wszystkim zwrócić uwagę na różne zbiory gier, które te metody mogą rozwiązywać. Metoda *O2-UCT* jest w stanie obliczać strategie lidera w dowolnych grach wielokrokowych o sumie niezerowej z niepełną informacją, dla których zachodzi własność doskonałej pamięci. (Tu warto przypomnieć, że metody *BC2105* i *C2016* również działają tylko dla gier z tą własnością). W przypadku metody *Mixed-UCT* wymagane jest, żeby gra spełniała dodatkowe założenie, (†) ze strony 115. To założenie nie jest spełnione w przypadku wielu gier, na przykład w przypadku zbioru *Search Games*, na której była testowana druga z metod — (*O2-UCT*). Ponadto metoda *Mixed-UCT*, o czym zostało wspomniane w Rozdziale 4.2.5, nie osiągnęła zadowalających wyników w trakcie wstępnych testów na zbiorze WHG non-zero sum. W związku z tym należy traktować metodę *O2-UCT* jako metodę bardziej skuteczną i bardziej ogólną niż *Mixed-UCT*.

Rozdział 5

Podsumowanie

W rozprawie zostały zaprezentowanie dwie metody metaheurystyczne, które służą do znalezienia strategii lidera w sekwencyjnych grach Stackelberga o sumie niezerowej z niepełną informacją z własnością doskonałej pamięci. Metoda późniejsza, *O2-UCT* jest w stanie operować na całej klasie wspomnianych gier, metoda wcześniejsza, *Mixed-UCT* jest w stanie operować tylko na podklasie gier spełniających dodatkowo warunek (†) ze strony 115: *zbiory informacyjne są jednoznacznie zdefiniowane przez sekwencję ruchów wykonanych przez gracza, dla którego ten zbiór informacyjny jest punktem decyzyjnym*. Obie metody bazują na algorytmie *UCT*, popularnej metaheurystyce do przeszukiwania gier z pełną informacją, na przykład gier planszowych. Metoda *Mixed-UCT* jest próbą bezpośredniego zastosowania podejścia *UCT* do otrzymywania strategii mieszanej lidera, natomiast szkielet metody *O2-UCT* jest silnie zainspirowany istniejącymi w literaturze podejściami do rozwiązywania gier Stackelberga, a podejście *UCT* zostało tam wykorzystane jako element realizujący mniejsze zadania, konkretnie ukierunkowane próbkowanie strategii naśladowcy oraz wybór ruchów do dodania do strategii lidera. Skuteczność obu metod została sprawdzona eksperymentalnie. Metody zostały uruchomione na różnych rodzinach gier testowych i porównane z istniejącymi podejściami z literatury. Eksperymenty pokazały, że obie proponowane metody dla małych gier testowych są dużo wolniejsze od podejść z literatury, jednak wraz ze wzrostem rozmiaru gier czas obliczeń, w przypadku obu metod proponowanych w rozprawie, rośnie wolniej, niż w przypadku metod z literatury. W przypadku obu metod, dla odpowiednio dużych gier testowych są one rzędy wielkości szybsze niż metody z literatury. Co więcej, ekstrapolacja wyników czasowych dla poszczególnych metod sugeruje, że metody z literatury mogą potrzebować czasu obliczeń liczonego w miesiącach lub nawet w latach, podczas gdy metodom metaheurytycznym wystarczy mniej niż 200 godzin. Podczas uruchamiania eksperymentów zaobserwowano również, że zapotrzebowanie metod metaheurystycznych na pamięć jest wielokrotnie mniejsze niż w przypadku metod zaczerpniętych z literatury. Oprócz zapotrzebowania na zasoby obliczeniowe, porównano również jakość znajdowanych strategii. W przypadku metody *Mixed-UCT* wypłaty uzyskiwane przez znalezione strategie są trochę gorsze niż wypłaty uzyskiwane przez strategie optymalne, co więcej, zaobserwowano, ze dla gier dalszych od sumy zerowej, wyniki *Mixed-UCT* są gor-

sze. W przypadku *O2-UCT* znalezione strategie dają wypłaty bardzo bliskie wypłatom, które można uzyskać grając strategie optymalne. Z racji tego, że metoda *O2-UCT* była zbudowana i testowana później, niż *Mixed-UCT*, istniała możliwość porównania metody *O2-UCT* z metodą oznaczaną w tej pracy jako *CBK2018*, która jest zupełnie innym podejściem do przybliżenia Równowagi Stackelberga. W porównaniu,*O2-UCT* znajdowała dużo lepsze strategie niż wspomniana metoda przybliżona, a dla dużych potrzebowała również mniej czasu na wyliczenie strategii. Pozwala to twierdzić, że podejście *O2-UCT* przy niewielkim koszcie czasowym pozwala wyznaczyć dobre przybliżenie strategii lidera ze stanu Równowagi Stackelberga. Porównanie metod *Mixed-UCT* i *O2-UCT* ze sobą pokazuje, że metoda *O2-UCT* może działać na dużo szerszej klasie gier niż *Mixed-UCT*, a wyliczone przez nią strategie dają lepsze wypłaty. W związku z tym można uznać, że *O2-UCT* jest podejściem uniwersalnie lepszym od *Mixed-UCT*.

5.1 Możliwości modyfikacji i dalszego rozwoju

Metody *Mixed-UCT* i *O2-UCT* zostały zaimplementowane i przetestowane. Wyniki eksperymentów potwierdzają ich użyteczność. Nie znaczy to jednak, że są to rozwiązania, których w żaden sposób nie można udoskonalić. W tym rozdziale zaprezentowane będą elementy, które warto zbadać pod kątem poprawy szybkości działania i jakości uzyskiwanych strategii w metodzie *O2-UCT*. Ponadto ten rozdział zawiera propozycje dalszego rozwoju tych metod, które nie są usprawnieniami działania w obszarze gier, które już zostały zbadane.

Usprawnienie przeglądu strategii naśladowcy. Elementem, który wstępuje w obu metodach, jest pełny przegląd strategii prostych naśladowcy. W przypadku metody *Mixed-UCT* procedura aktualizacji strategii naśladowcy w linii 14 Algorytmu 4.2 wymaga znalezienia najlepszej odpowiedzi naśladowcy na zadaną strategię lidera. Podobnie w metodzie *O2-UCT* stwierdzenie czy istnieje strategia spełniająca warunki do bycia strategią π_f^b będąca częścią procedury poszukiwania strategii lidera opisanej w Rozdziale 4.3.2, mimo usprawnienia polegającego na stosowaniu repertuaru ostatnio znalezionych strategii naśladowcy, wymaga zastosowania pełnego przeglądu strategii tych strategii. Wszystkie użyte zbiory testowe mają własność, że liczba strategii prostych naśladowcy rośnie wykładniczo wraz ze zwiększaniem liczby rund w grze. Złagodzenie tego kosztu przyniosłoby duży zysk w czasie wykonania w przypadku dużych gier.

Naturalnym, w kontekście tej rozprawy, byłoby zastosowanie algorytmu *UCT*, również do przeglądu strategii naśladowcy. Proste zastosowanie algorytmu *UCT* nie jest jednak wystarczające w tym miejscu. O ile, wraz ze wzrostem liczby symulacji *UCT* rośnie szansa, że najlepiej oceniony ruch to ruch faktycznie najlepszy, to jednak nigdy nie ma takiej gwarancji. W momencie, kiedy liczba symulacji jest mała, istnieje ryzyko, że istnieje lepszy ruch, niż wskazany przez metodę jako najlepszy. W przypadku, gdy decydujemy, czy zadana strategia lidera

jest w obszarze dopuszczalnym, czy nie, taka pomyłka może prowadzić do błędnego przekonania, że aktualna strategia lidera jest w obszarze dopuszczalnym, mimo, że w rzeczywistości tak nie jest. W związku z tym, przed zastosowaniem heurystycznego przeszukiwania przestrzeni strategii prostych naśladowcy konieczne jest opracowanie rozwiązania, które łagodzi błędnej oceny bycia w obszarze dopuszczalnym. Jest to kierunek, w którym warto poprowadzić badania nam metodą *O2-UCT*.

Modyfikacja metody poprawy strategii lidera w metodzie *O2-UCT*. Szczegółowa analiza przebiegów metody *O2-UCT* pokazuje, że aplikowanie kroku aktualizacji strategii lidera w celu poprawy wypłaty przeciw strategii naśladowcy π_f^r często powoduje wyjście poza obszar dopuszczalny. Implikuje to potrzebę wielokrotnego wykonywania kroków poprawy w celu powrotu do obszaru dopuszczalnego. Mimo zastosowania momentu, który łagodzi ten problem, liczba dodatkowych kroków poprawy, które służą tylko powrotowi do obszaru dopuszczalnego jest duża i znacząco wpływa na czas wykonania algorytmu. Planowane jest wprowadzenie modyfikacji do sposobu wyznaczania kierunku aktualizacji strategii w celu poprawy wypłaty lidera tak, aby uwzględniał on ostatnio znalezione lepsze odpowiedzi naśladowcy i żeby aktualizacja nie powodowała wyjścia poza obszar dopuszczalny.

Rozważanie niepełnej racjonalności naśladowcy. We wstępie do rozprawy, w Rozdziale 2.7 zasygnalizowano, że w wielu praktycznych problemach nie zakłada się systematyczne zaburzenie odpowiedzi naśladowcy, które modeluje fakt, że naśladowca w rzeczywistej sytuacji może nie podejmować w pełni racjonalnych decyzji. W chwili pisania tej rozprawy trwają prace nad dodaniem do metody *O2-UCT* możliwości modelowania niepełnej racjonalności naśladowcy. Jak dotąd został opublikowany rozszerzony abstrakt, który zawiera wstępne wyniki dotyczące zastosowania różnych podejść metaheurystycznych, w tym *O2-UCT* do problemu niepełnej racjonalności [46].

5.2 Weryfikacja hipotezy badawczej

Hipoteza postawiona na początku tej rozprawy brzmi:

> *Możliwe jest wykorzystanie metod Monte Carlo do efektywnego aproksymowania strategii lidera w wielokrokowych Grach Stackelberga o sumie niezerowej z niepełną informacją, które dają niewiele gorszą wartość oczekiwana wypłaty lidera, przy optymalnej odpowiedzi naśladowcy, w porównaniu z wartością oczekiwaną wypłaty lidera w stanie równowagi.*

Przeprowadzone badania pokazały, że metoda *Mixed-UCT*, która bezpośrednio wykorzystuje Monte Carlo Tree Seach/Upper Confidence Bound applied to Trees (UCT) potrafi efektywnie przybliżać strategie lidera w Grach Stackelberga o sumie *bliskiej sumie zerowej* z niepełną informacją oraz dodatkowo ze specyficzną strukturą informacji definiowaną warunkiem

(†) na stronie 115. Metoda nie działa w sposób zadowalający dla całej klasy gier z niepełną informacją i tylko w częściowy sposób realizuje założenie postawione w hipotezie. Należy jednak podkreślić, że założeniem w trakcie konstrukcji tej metody było zastosowanie metody UCT w sposób bezpośredni do poszukiwania strategii lidera. Druga z metod, *O2-UCT* korzysta z metod Monte Carlo, konkretnie metody UCT pośrednio, jako mechanizmu próbkowania strategii naśladowcy oraz jako mechanizmu poszukiwania ruchu, który będzie dodany do strategii lidera. W tym przypadku wyniki eksperymentalne potwierdzają, że wartości oczekiwane wypłaty dla znalezionych strategii lidera są tylko niewiele gorsze od wartości oczekiwanych wypłaty dla strategii ze stanu Równowagi Stackelberga. To oznacza, że udało się potwierdzić hipotezę badawczą postawioną na początku rozprawy.

5.3 Konkluzje

Główną osią porównań w części eksperymentalnej związanej z metodami *Mixed-UCT* i *O2-UCT* było porównanie wyników z wynikami metod opartych o programowanie liniowe. Programowanie liniowe jest klasycznym i popularnym podejściem do rozwiązywania różnych klas problemów. Ogromnym plusem stosowania programowania liniowego jest fakt, że na rynku dostępne są solwery, które zapewniają bardzo konkurencyjne czasy obliczeń nawet dla dużych programów liniowych. Mimo to, metody te, dla dużych gier, wykonywały się znacznie wolniej niż bazujące na podejściach metaheurystycznych metody proponowane w tej rozprawie. Zysk czasowy jest tym większy im większe są to gry. Jakość uzyskiwanych wyników pozostaje przy tym na wysokim poziomie. Kosztem związanym ze stosowaniem metaheurystyk jest jednak bardziej czasochłonna i skomplikowana implementacja. Programowanie liniowe wymaga niewielkiego nakładu programistycznego, podczas gdy metody *Mixed-UCT* i *O2-UCT* to kilka tysięcy linii kodu. Po części objętość wynika z faktu, że w przypadku wielu metaheurystyk brakuje gotowych i łatwych w użyciu bibliotek z implementacją. To pokazuje, że warto promować podejścia metaheurystyczne, które często pozwalają rozwiązywać problemy wielokrotnie większe niż te, które można rozwiązać klasycznymi metodami, bo takie działania promocyjne mogą sprowokować powstanie lepszych bibliotek implementujących te rozwiązania.

Mówiąc o rozwiązywaniu większych problemów warto też zwrócić uwagę na fakt, że podejścia metaheurystyczne, które dają nieznacznie tylko gorsze wyniki od dokładnych, pozwalają na rozwiązywanie dużo większych problemów. W praktyce może to oznaczać, że dzięki zastosowaniu takiej metody możliwe będzie zbudowanie dużo dokładniejszego modelu rzeczywistego procesu i mimo tego, że względem wybranego modelu rozwiązanie to nie będzie optymalne, znalezienie rozwiązania lepszego w rzeczywistym zastosowaniu, bo obarczonego dużo mniejszym uproszczeniem na etapie modelowania.

Rozprawa pokazuje również, że obszar Gier Stackelberga, choć mało popularny w świadomości ogółu badaczy w obszarze informatyki jest niezwykle interesujący. Asymetria graczy w Równowadze Stackelberga powoduje, że subtelne zmiany w strategii lidera mogą mieć bar-

dzo duży wpływ na otrzymaną na koniec wartość oczekiwaną wypłaty. Ta cecha powoduje, że metody, które łatwo jest zastosować do gier gdzie gracze są symetryczni trudno jest zaadaptować do Gier Stackelberga. Widać to chociażby w konstrukcji metody *Mixed-UCT*, gdzie adaptacja metody *UCT* wymagała znaczącej liczby dodatkowych elementów aby można ją było wykorzystać do poszukiwania strategii lidera. Należy przy tym pamiętać, że z Grami Stackelberga wiąże się szereg zastosowań wymienionych we wstępie do tej rozprawy. W opinii autora rozprawy wśród badaczy z obszaru informatyki brakuje zainteresowania tematem gier asymetrycznych, w szczególności Gier Stackelberga i należy rozpocząć działania na rzecz popularyzacji prowadzenia badań w tym obszarze.

Jedną z przeszkód, które powodują, że bardzo rzadko autorzy porównują ze sobą różne metody do poszukiwania Równowagi Stackelberga jest fakt, że w tej dziedzinie, w przeciwieństwie do wielu innych dziecin, jak na przykład różne obszary uczenia maszynowego, czy optymalizacji, nie ma ogólnie przyjętego zbioru do testowania skuteczności metod. Praktycznie każda z prac przytoczonych w przeglądzie metod w Rozdziale 3 była testowana na innych zbiorach gier. Czyni to porównywanie skuteczności metod trudnym. Co więcej cechy konkretnego zbioru testowego mogą być (niekoniecznie w świadomy sposób) dobrane tak, aby faworyzować którąś z metod. W związku z tym autor rozprawy chciałby podkreślić konieczność budowania repozytoriów gier testowych, które pozwolą na bardziej ujednolicony sposób porównywania metod przybliżających Równowagę Stackelberga.

Bibliografia

[1] Bo An, Fernando Ordóñez, Milind Tambe, Eric Shieh, Rong Yang, Craig Baldwin, Joseph DiRenzo III, Kathryn Moretti, Ben Maule i Garrett Meyer. *A deployed quantal response-based patrol planning system for the US Coast Guard*. W: *Interfaces* 43.5 (2013), s. 400–420. DOI: 10.1287/inte.2013.0700.

[2] Bo An i Milind Tambe. *Stackelberg Security Games (SSG) Basics and Application Overview*. W: *Improving Homeland Security Decisions* (2007), 485–507. DOI: 10.1017/9781316676714.021.

[3] Jaqueline S. Angelo i Helio J. C. Barbosa. *Differential evolution to find Stackelberg-Nash equilibrium in bilevel problems with multiple followers*. W: *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015). DOI: 10.1109/cec.2015.7257088.

[4] Peter Auer, Nicolo Cesa-Bianchi i Paul Fischer. *Finite-time analysis of the multiarmed bandit problem*. W: *Machine learning* 47.2-3 (2002), s. 235–256. DOI: 10.1023/A:1013689704352.

[5] Robert J. Aumann. *Subjectivity and correlation in randomized strategies*. W: *Journal of Mathematical Economics* 1.1 (1974), 67–96. ISSN: 0304-4068. DOI: 10.1016/0304-4068(74)90037-8.

[6] Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi i Viggo Kann. *Complexity and approximation: combinatorial optimization problems and their approximability properties*. Springer, 1999. ISBN: 3540654313. DOI: 10.1007/978-3-642-58412-1.

[7] Rudolf Avenhaus, Akira Okada i Shmuel Zamir. *Inspector Leadership with Incomplete Information*. W: *Game Equilibrium Models IV* (1991), 319–361. DOI: 10.1007/978-3-662-07369-8_14.

[8] Nicola Basilico, Nicola Gatti i Francesco Amigoni. *Leader-follower strategies for robotic patrolling in environments with arbitrary topologies*. W: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*. 2009, s. 57–64.

[9] Nicola Basilico, Nicola Gatti i Francesco Amigoni. *Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder*. W: *Artificial Intelligence* 184-185 (2012), 78–123. ISSN: 0004-3702. DOI: 10.1016/j.artint.2012.03.003.

[10] T. Başar i R. Srikant. *A Stackelberg Network Game with a Large Number of Followers*. W: *Journal of Optimization Theory and Applications* 115.3 (2002), 479–490. ISSN: 1573-2878. DOI: 10.1023/a:1021294828483.

[11] Tamer Başar i Geert Jan Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition*. Society for Industrial i Applied Mathematics, 1998. ISBN: 9781611971132. DOI: 10.1137/1.9781611971132.

[12] Ulrich Berger. *Brown's original fictitious play*. W: *J. Economic Theory* 135.1 (2007), s. 572–578. DOI: 10.1016/j.jet.2005.12.010.

[13] Dimitris Bertsimas i John N. Tsitsiklis. *Introduction to linear optimization*. Athena scientific optimization and computation series. Athena Scientific, 1997. ISBN: 978-1-886529-19-9.

[14] Juan S. Borrero, Oleg A. Prokopyev i Denis Sauré. *Sequential Interdiction with Incomplete Information and Learning*. W: *Operations Research* 67.1 (2019), 72–89. ISSN: 1526-5463. DOI: 10.1287/opre.2018.1773.

[15] Juan Sebastian Borrero, Oleg A. Prokopyev i Denis Sauré. *Sequential Shortest Path Interdiction with Incomplete Information*. W: *Decision Analysis* 13.1 (2016), s. 68–98. DOI: 10.1287/deca.2015.0325.

[16] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý i Michal Pěchouček. *An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information*. W: *J. Artif. Intell. Res.* 51 (2014), s. 829–866. DOI: 10.1613/jair.4477.

[17] Branislav Bošanský i Jiří Čermak. *Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games*. W: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Red. Blai Bonet i Sven Koenig. AAAI Press, 2015, s. 805–811. ISBN: 978-1-57735-698-1.

[18] Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Peter Bro Miltersen i Troels Bjerre Sørensen. *Computation of Stackelberg Equilibria of Finite Sequential Games*. W: *Lecture Notes in Computer Science* (2015), 201–215. ISSN: 1611-3349. DOI: 10.1007/978-3-662-48995-6_15.

[19] Steven J. Brams. *Mathematics and Democracy*. W: (2008). DOI: 10.1515/9781400835591.

[20] M. Breton, A. Alj i A. Haurie. *Sequential Stackelberg equilibria in two-person games*. W: *Journal of Optimization Theory and Applications* 59.1 (1988), 71–97. ISSN: 1573-2878. DOI: 10.1007/bf00939867.

[21] George W Brown. *Iterative solution of games by fictitious play*. W: *Activity analysis of production and allocation* 13.1 (1951), s. 374–376.

[22] Gerald G. Brown, W. Matthew Carlyle, Robert C. Harney, Eric M. Skroch i R. Kevin Wood. *Interdicting a Nuclear-Weapons Project*. W: *Operations Research* 57.4 (2009), 866—877. ISSN: 1526-5463. DOI: 10.1287/opre.1080.0643.

[23] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis i Simon Colton. *A Survey of Monte Carlo Tree Search Methods*. W: *IEEE Trans. Comput. Intellig. and AI in Games* 4.1 (2012), s. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[24] Vincent Conitzer i Tuomas Sandholm. *Computing the optimal strategy to commit to*. W: *Proceedings of the 7th ACM conference on Electronic commerce - EC '06* (2006). DOI: 10.1145/1134707.1134717.

[25] A.A. Cournot. *Researches Into the Mathematical Principles of the Theory of Wealth*. Macmillan, 1897.

[26] Hamzeh Davarikia i Masoud Barati. *A tri-level programming model for attack-resilient control of power grids*. W: *Journal of Modern Power Systems and Clean Energy* 6.5 (2018), 918–929. ISSN: 2196-5420. DOI: 10.1007/s40565-018-0436-y.

[27] Stephan Dempe. *Foundations of Bilevel Programming*. Springer New York, NY, 2002. ISBN: 978-0-306-48045-4. DOI: 10.1007/b101970.

[28] Guy Desaulniers, Jacques Desrosiers i Marius M. Solomon. *Column Generation*. Springer, Boston, MA, 2005. ISBN: 9780387254869. DOI: 10.1007/b135457.

[29] Marten van Dijk, Ari Juels, Alina Oprea i Ronald L. Rivest. *FlipIt: The Game of „Stealthy Takeover"*. W: *Journal of Cryptology* 26.4 (2012), 655–713. ISSN: 1432-1378. DOI: 10.1007/s00145-012-9134-5.

[30] A. Downs. *An Economic Theory of Democracy*. Harper, 1957.

[31] Fei Fang, Thanh H. Nguyen, Rob Pickles, Wai Y. Lam, Gopalasamy R. Clements, Bo An, Amandeep Singh, Brian C. Schwedock, Milin Tambe i Andrew Lemieux. *PAWS — A Deployed Game-Theoretic Application to Combat Poaching*. W: *AI Magazine* 38.1 (2017), s. 23. ISSN: 0738-4602. DOI: 10.1609/aimag.v38i1.2710.

[32] Fei Fang, Peter Stone i Milind Tambe. *When Security Games go green: Designing defender strategies to prevent poaching and illegal fishing*. W: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2015.

[33] Matthew L. Ginsberg. *GIB: Imperfect information in a computationally challenging game*. W: *Journal of Artificial Intelligence Research* 14 (2001), s. 303–358.

[34] Gurobi Optimization, Inc. *Gurobi optimizer reference manual*. `http://www.gurobi.com/documentation/8.0/`. 2018.

[35] John C. Harsanyi. *Games with Incomplete Information Played by "Bayesian" Players, I–III Part I. The Basic Model*. W: *Papers in Game Theory* (1982), 115–138. DOI: `10.1007/978-94-017-2527-9_6`.

[36] Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordóñez i Milind Tambe. *Security Games with Arbitrary Schedules: A Branch and Price Approach*. W: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Red. Maria Fox i David Poole. AAAI Press, 2010.

[37] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe i Fernando Ordóñez. *Software assistants for randomized patrol planning for the LAX airport police and the federal air marshal service*. W: *Interfaces* 40.4 (2010), s. 267–290.

[38] Matthew Paul Johnson, Fei Fang i Milind Tambe. *Patrol Strategies to Maximize Pristine Forest Area*. W: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2012, s. 295–301.

[39] Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum i Frank van Harmelen, red. *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016. ISBN: 978-1-61499-671-2.

[40] Jan Karwowski i Jacek Mańdziuk. *A New Approach to Security Games*. W: *Artificial Intelligence and Soft Computing - 14th International Conference, ICAISC 2015, Zakopane, Poland, June 14-28, 2015, Proceedings, Part II*. Red. Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh i Jacek M. Zurada. T. 9120. Lecture Notes in Computer Science. Springer, 2015, s. 402–411. ISBN: 978-3-319-19368-7. DOI: `10.1007/978-3-319-19369-4\_36`.

[41] Jan Karwowski i Jacek Mańdziuk. *Mixed Strategy Extraction from UCT Tree in Security Games*. W: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. Red. Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum i Frank van Harmelen. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, s. 1746–1747. ISBN: 978-1-61499-671-2. DOI: `10.3233/978-1-61499-672-9-1746`.

[42] Jan Karwowski i Jacek Mańdziuk. *The Impact of the Number of Averaged Attacker's Strategies on the Results Quality in Mixed-UCT*. W: *Artificial Intelligence and Soft Computing - 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11-15, 2017*. Red. Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh i Jacek M. Zurada. T. 10246. Lecture Notes in Computer Science. Springer, 2017, s. 477–488. ISBN: 978-3-319-59059-2. DOI: 10.1007/978-3-319-59060-8_43.

[43] Jan Karwowski i Jacek Mańdziuk. *A Monte Carlo Tree Search approach to finding efficient patrolling schemes on graphs*. W: *European Journal of Operational Research* 277.1 (2019), s. 255–268. DOI: 10.1016/j.ejor.2019.02.017.

[44] Jan Karwowski i Jacek Mańdziuk. *Double-Oracle Sampling Method for Stackelberg Equilibrium Approximation in General-Sum Extensive-Form Games*. W: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.02 (2020), 2054–2061. ISSN: 2159-5399. DOI: 10.1609/aaai.v34i02.5578.

[45] Jan Karwowski i Jacek Mańdziuk. *Stackelberg Equilibrium Approximation in General-Sum Extensive-Form Games with Double-Oracle Sampling Method*. W: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. Red. Edith Elkind, Manuela Veloso, Noa Agmon i Matthew E. Taylor. International Foundation for Autonomous Agents i Multiagent Systems, 2019, s. 2045–2047. ISBN: 978-1-4503-6309-9.

[46] Jan Karwowski, Jacek Mańdziuk i Adam Żychowski. *Anchoring Theory in Sequential Stackelberg Games*. W: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. Red. Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An i Neil Yorke-Smith. International Foundation for Autonomous Agents i Multiagent Systems, 2020, s. 1881–1883. ISBN: 978-1-4503-7518-4.

[47] Jan Karwowski, Jacek Mańdziuk, Adam Żychowski, Filip Grajek i Bo An. *A Memetic Approach for Sequential Security Games on a Plane with Moving Targets*. W: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, s. 970–977. ISBN: 978-1-57735-809-1.

[48] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez i Milind Tambe. *Computing optimal randomized resource allocations for massive security games*. W: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*. 2009, s. 689–696.

[49] Christopher Kiekintveld, Janusz Marecki i Milind Tambe. *Methods and Algorithms for Infinite Bayesian Stackelberg Security Games*. W: *Decision and Game Theory for Security* (2010), 257–265. ISSN: 1611-3349. DOI: 10.1007/978-3-642-17197-0_18.

[50] Levente Kocsis i Csaba Szepesvári. *Bandit based monte-carlo planning*. W: *Machine Learning: ECML 2006*. Springer, 2006, s. 282–293.

[51] Daphne Koller, Nimrod Megiddo i Bernhard von Stengel. *Efficient Computation of Equilibria for Extensive Two-Person Games*. W: *Games and Economic Behavior* 14.2 (1996), 247–259. ISSN: 0899-8256. DOI: 10.1006/game.1996.0051.

[52] H. W. Kuhn. *Extensive Games and the Problem of Information*. W: *Contributions to the Theory of Games (AM-28)* II (1953). Red. Harold William Kuhn i Albert William Tucker, 193–216. DOI: 10.1515/9781400881970-012.

[53] Harold W Kuhn. *Extensive games*. W: *Proceedings of the National Academy of Sciences of the United States of America* 36.10 (1950), s. 570.

[54] G. Leitmann. *On generalized Stackelberg strategies*. W: *Journal of Optimization Theory and Applications* 26.4 (1978), 637–643. ISSN: 1573-2878. DOI: 10.1007/bf00933155.

[55] Joshua Letchford. "Computational Aspects of Stackelberg Games". Prac. dokt. Duke University, 2013.

[56] Kenneth Levenberg. *A method for the solution of certain non-linear problems in least squares*. W: *Quarterly of applied mathematics* 2.2 (1944), s. 164–168.

[57] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin i Hefei China. *Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization*. W: (2013).

[58] Alberto Marchesi, Matteo Castiglioni i Nicola Gatti. *Leadership in Congestion Games: Multiple User Classes and Non-Singleton Actions*. W: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (2019). DOI: 10.24963/ijcai.2019/69.

[59] Alberto Marchesi, Stefano Coniglio i Nicola Gatti. *Leadership in Singleton Congestion Games*. W: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (2018). DOI: 10.24963/ijcai.2018/62.

[60] Jason R. Marden, Gürdal Arslan i Jeff S. Shamma. *Joint Strategy Fictitious Play With Inertia for Potential Games*. W: *IEEE Trans. Automat. Contr.* 54.2 (2009), s. 208–220. DOI: 10.1109/TAC.2008.2010885.

[61] Michael Maschler. *A price leadership method for solving the inspector's non-constant-sum game*. W: *Naval Research Logistics Quarterly* 13.1 (1966), 11–33. ISSN: 1931-9193. DOI: 10.1002/nav.3800130103.

[62] H. Brendan McMahan, Geoffrey J. Gordon i Avrim Blum. *Planning in the Presence of Cost Functions Controlled by an Adversary*. W: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. Red. Tom Fawcett i Nina Mishra. AAAI Press, 2003, s. 536–543. ISBN: 1-57735-189-4.

[63] David P. Morton, Feng Pan i Kevin J. Saeger. *Models for nuclear smuggling interdiction*. W: *IIE Transactions* 39.1 (2007), 3—14. ISSN: 1545-8830. DOI: 10.1080/07408170500488956.

[64] Todd W Neller i Marc Lanctot. *An introduction to counterfactual regret minimization*. W: *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*. T. 11. 2013.

[65] John von Neumann i Oskar Morgenstern. *Theory of games and economic behavior, 2nd rev*. Princeton University Press, 1947.

[66] Giuseppe De Nittis, Alberto Marchesi i Nicola Gatti. *Computing the Strategy to Commit to in Polymatrix Games*. W: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*. 2018, s. 989–996.

[67] Simon Parsons i Michael J. Wooldridge. *Game Theory and Decision Theory in Multi-Agent Systems*. W: *Autonomous Agents and Multi-Agent Systems* 5.3 (2002), s. 243–254. DOI: 10.1023/A:1015575522401.

[68] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez i Sarit Kraus. *Efficient Algorithms to Solve Bayesian Stackelberg Games for Security Applications.* W: *AAAI*. 2008, s. 1559–1562.

[69] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez i Sarit Kraus. *Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games*. W: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. 2008, s. 895–902.

[70] Eghbal Rashidi, Hugh Medal i Aaron Hoskins. *An attacker-defender model for analyzing the vulnerability of initial attack in wildfire suppression*. W: *Naval Research Logistics (NRL)* 65.2 (2018), 120—134. ISSN: 0894-069X. DOI: 10.1002/nav.21792.

[71] Eghbal Rashidi, Hugh Richard Medal i Aaron Hoskins. *Mitigating a pyro-terror attack using fuel treatment*. W: *IISE Transactions* 50.6 (2018), 499–511. ISSN: 2472-5862. DOI: 10.1080/24725854.2017.1415490.

[72] Andrew Romich, Guanghui Lan i J. Cole Smith. *Algorithms for optimizing the placement of stationary monitors*. W: *IIE Transactions* 47.6 (2015), 556—576. ISSN: 1545-8830. DOI: 10.1080/0740817x.2014.953646.

[73] Aaron Schlenker, Matthew Brown, Arunesh Sinha, Milind Tambe i Ruta Mehta. *Get Me to My GATE on Time: Efficiently Solving General-Sum Bayesian Threat Screening Games*. W: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. Red. Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum i Frank van Harmelen. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, s. 1476–1484. ISBN: 978-1-61499-671-2. DOI: 10.3233/978-1-61499-672-9-1476.

[74] Julian Schrittwieser i in. *Mastering Atari, Go, chess and shogi by planning with a learned model*. W: *Nature* 588.7839 (2020), 604–609. ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4.

[75] Yoav Shoham i Kevin Leyton-Brown. *Multiagent Systems. Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008. ISBN: 9780511811654. DOI: 10.1017/cbo9780511811654.

[76] David Silver i Joel Veness. *Monte-Carlo Planning in Large POMDPs*. W: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.* Red. John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel i Aron Culotta. Curran Associates, Inc., 2010, s. 2164–2172.

[77] David Silver i in. *Mastering the game of Go with deep neural networks and tree search*. W: *Nature* 529.7587 (2016), s. 484–489. DOI: 10.1038/nature16961.

[78] David Silver i in. *Mastering the game of Go without human knowledge*. W: *Nature* 550.7676 (2017), 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270.

[79] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld i Milind Tambe. *Stackelberg Security Games: Looking Beyond a Decade of Success*. W: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, lip. 2018, s. 5494–5501. DOI: 10.24963/ijcai.2018/775.

[80] J. Cole Smith, Mike Prince i Joseph Geunes. *Modern Network Interdiction Problems and Algorithms*. W: *Handbook of Combinatorial Optimization* (2013), 1949—1987. DOI: 10.1007/978-1-4419-7997-1_61.

[81] Heinrich von Stackelberg. *Marktform und Gleichgewicht*. Wiedeń: Springer, 1934.

[82] Bernhard von Stengel. *Efficient Computation of Behavior Strategies*. W: *Games and Economic Behavior* 14.2 (1996), 220–246. ISSN: 0899-8256. DOI: 10.1006/game.1996.0050.

[83] Bernhard von Stengel i Françoise Forges. *Extensive-Form Correlated Equilibrium: Definition and Computational Complexity*. W: *Mathematics of Operations Research* 33.4 (2008), 1002–1022. ISSN: 1526-5471. DOI: 10.1287/moor.1080.0340.

[84] Bernhard von Stengel i Shmuel Zamir. *Leadership games with convex strategy sets*. W: *Games and Economic Behavior* 69.2 (2010), 446–457. ISSN: 0899-8256. DOI: 10.1016/j.geb.2009.11.008.

[85] F. Teytaud i O. Teytaud. *Creating an Upper-Confidence-Tree program for Havannah*. W: *Advances in Computer Games* (2010), s. 65–74.

[86] Jason Tsai, Zhengyu Yin, Jun-young Kwak, David Kempe, Christopher Kiekintveld i Milind Tambe. *Urban security: Game-theoretic resource allocation in networked physical domains*. W: *AAAI*. 2010, s. 881–886.

[87] Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý i Christopher Kiekintveld. *Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games*. W: *30th AAAI Conference on Artificial Intelligence*. 2016, s. 439–445.

[88] Bernhard Von Stengel i Shmuel Zamir. *Leadership with commitment to mixed strategies*. Spraw. tech. CDAM, 2004.

[89] Karol Walędzik, Jacek Mańdziuk i Sławomir Zadrożny. *Risk-Aware Project Scheduling for Projects with Varied Risk Levels*. W: *3rd IEEE Symposium on Computational Intelligence for Human-like Intelligence*. IEEE Press, 2015, s. 1642–1649. DOI: 10.1109/SSCI.2015.231.

[90] Xinrun Wang, Bo An, Martin Strobel i Fookwai Kong. *Catching Captain Jack: Efficient Time and Space Dependent Patrols to Combat Oil-Siphoning in International Waters*. W: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018, s. 208–215.

[91] Yue Yin, Bo An i Manish Jain. *Game-theoretic resource allocation for protecting large public events*. W: *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*. 2014, s. 826–834.

[92] Zhengyu Yin, Albert Xin Jiang, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, Tuomas Sandholm i John P. Sullivan. *TRUSTS: Scheduling Randomized Patrols for Fare Inspection in Transit Systems Using Game Theory*. W: *AI Magazine* 33.4 (2012), s. 59. ISSN: 0738-4602. DOI: 10.1609/aimag.v33i4.2432.

[93] Jing Zhang, Jun Zhuang i Brandon Behlendorf. *Stochastic shortest path network interdiction with a case study of Arizona–Mexico border*. W: *Reliability Engineering & System Safety* 179 (2018), 62–73. ISSN: 0951-8320. DOI: 10.1016/j.ress.2017.10.026.

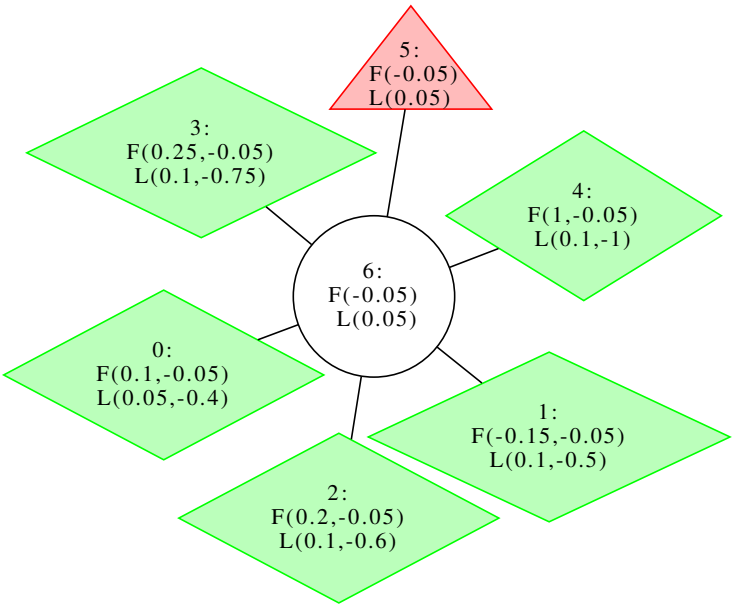
[94] Martin Zinkevich, Michael Johanson, Michael H. Bowling i Carmelo Piccione. *Regret Minimization in Games with Incomplete Information*. W: *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*. Red. John C. Platt, Daphne Koller, Yoram Singer i Sam T. Roweis. Curran Associates, Inc., 2007, s. 1729–1736.

[95] Jakub Černý, Branislav Bošanský i Christopher Kiekintveld. *Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games*. W: *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*. Red. Éva Tardos, Edith Elkind i Rakesh Vohra. ACM, 2018, s. 151–168. DOI: 10.1145/3219166.3219219.

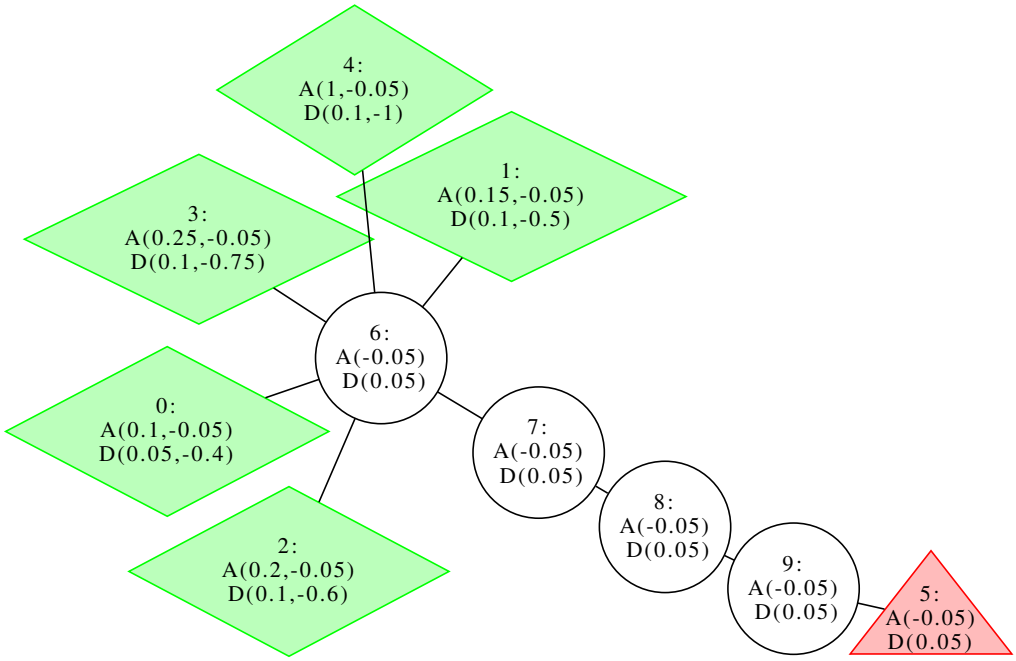
Dodatek A

Gry testowe ze zbioru Basic

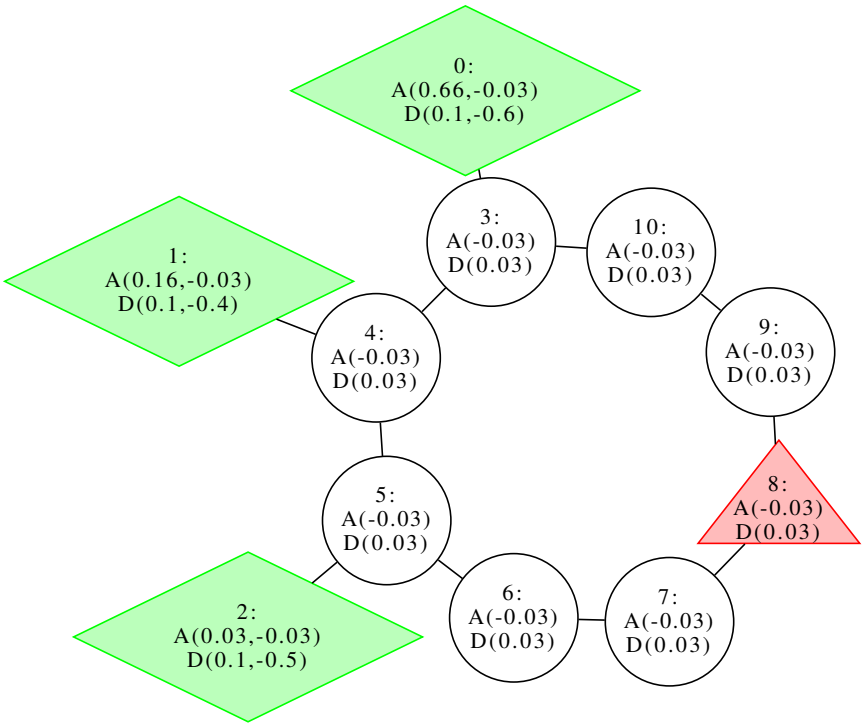
W tym dodatku zaprezentowane są wszystkie gry wchodzące w skład zbioru *Basic* opisanego na stronie 110. Rysunki A.1—A.12 przedstawiają poszczególne gry testowe. W szczególności przedstawione są gry na grafie strukturze grafu *game3* ze zmienionymi wypłatami.



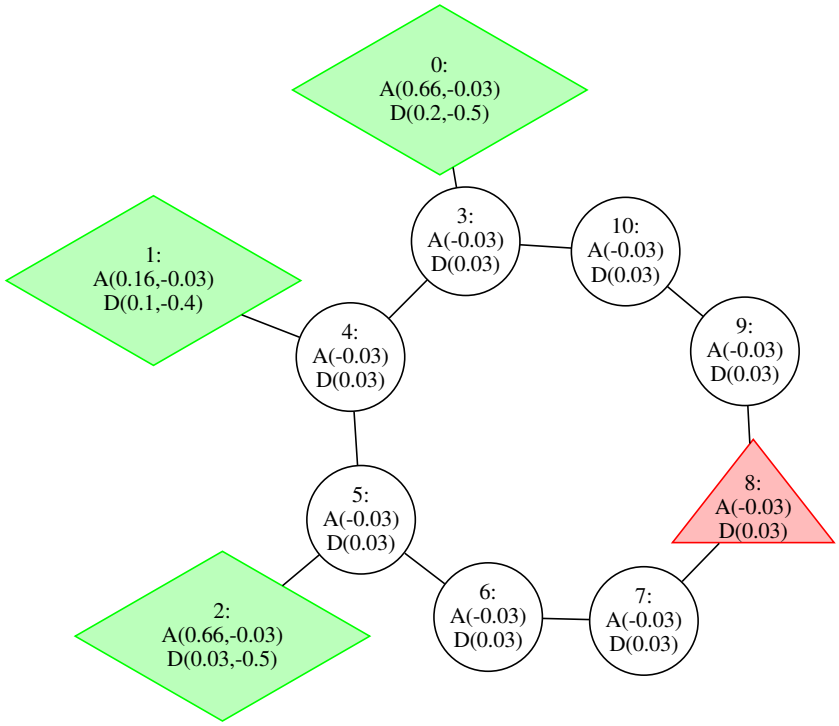
Rysunek A.1: *game1*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 5 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



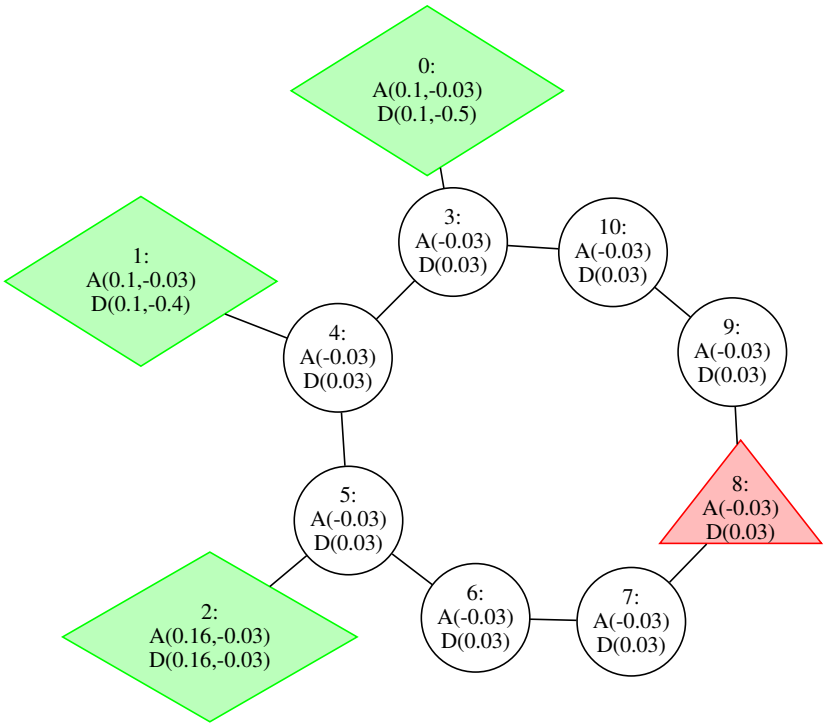
Rysunek A.2: *game1*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 5 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



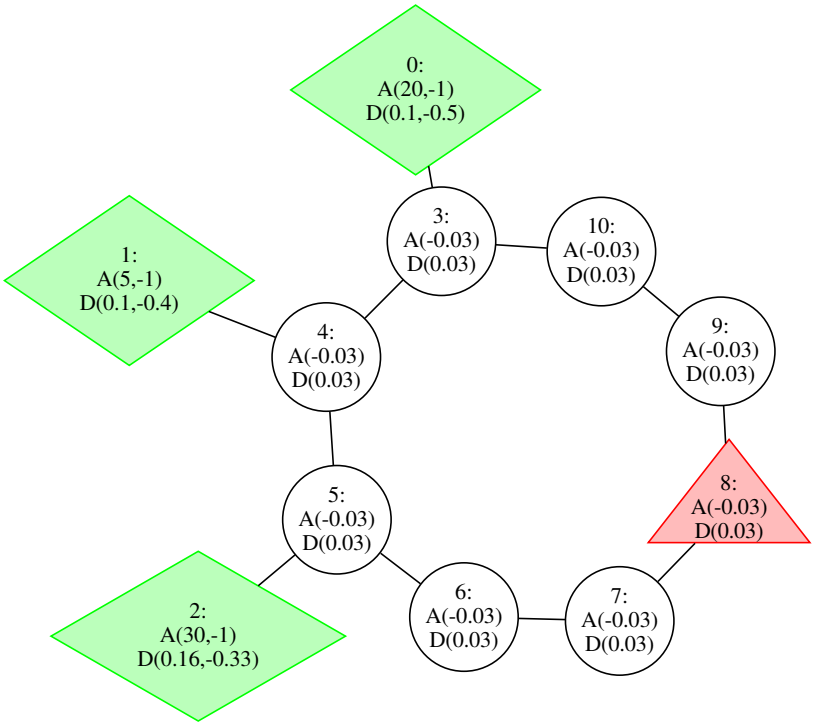
Rysunek A.3: *game3*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



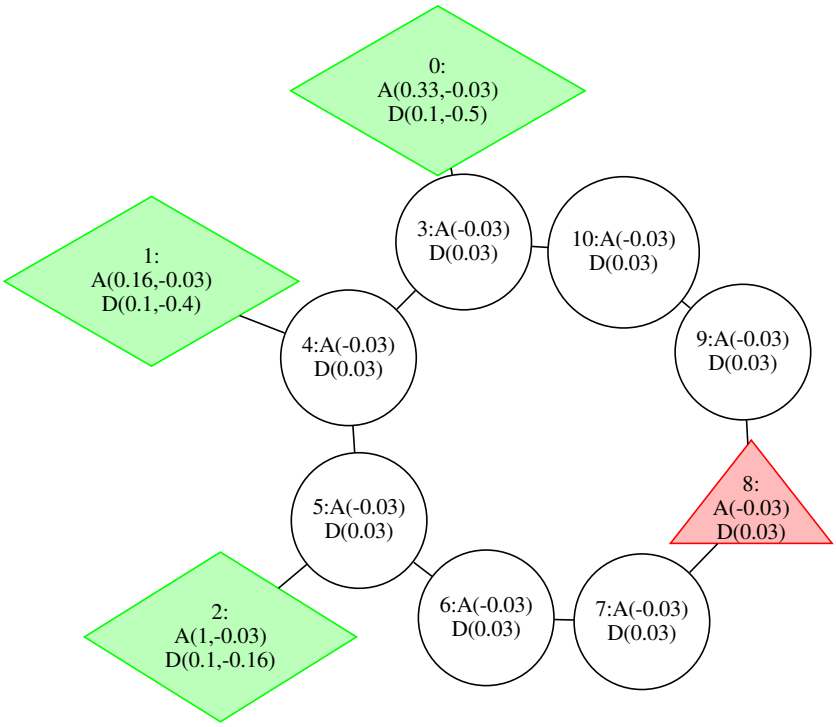
Rysunek A.4: *game3a*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



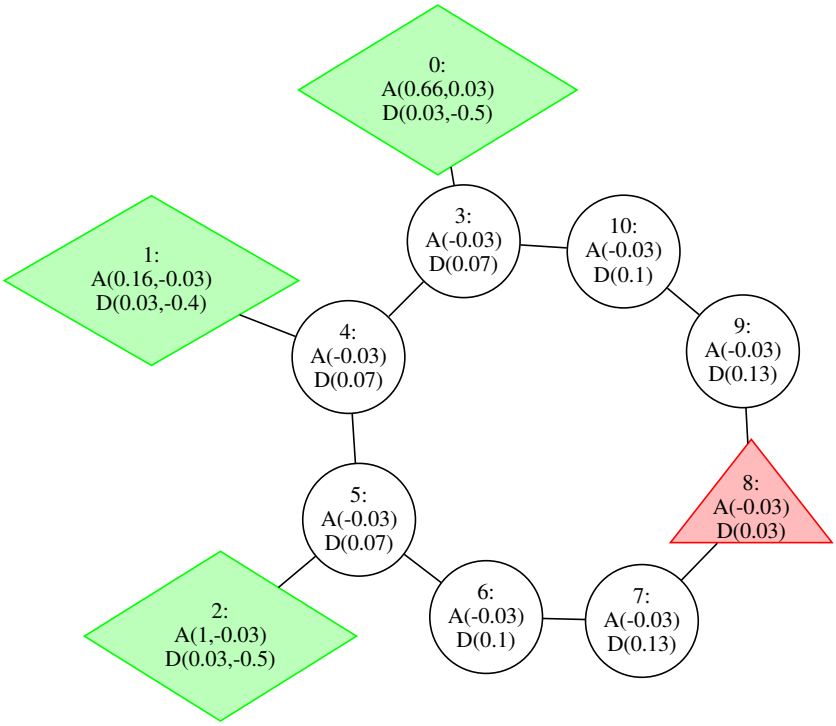
Rysunek A.5: *game3b*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



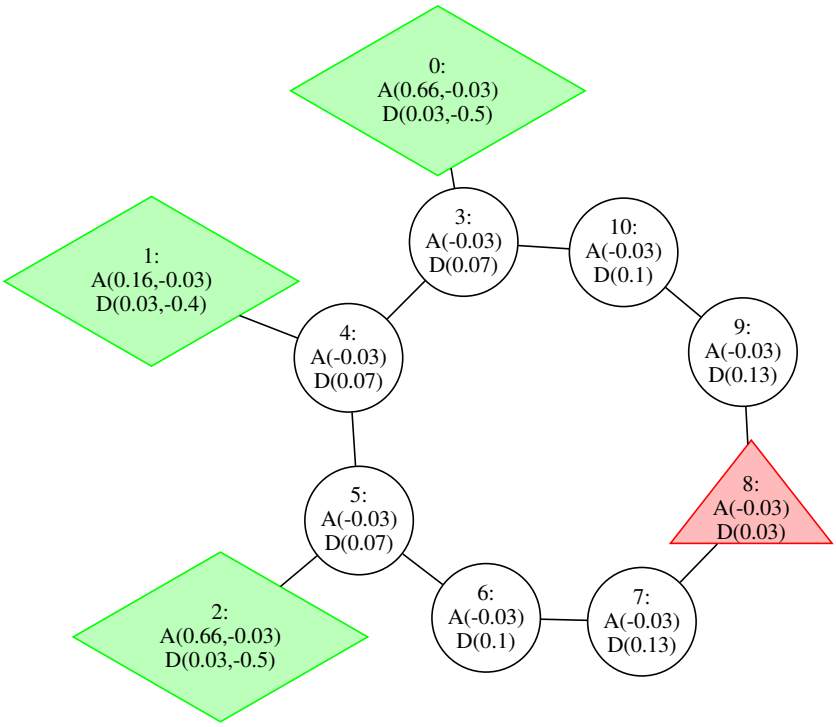
Rysunek A.6: *game3c*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



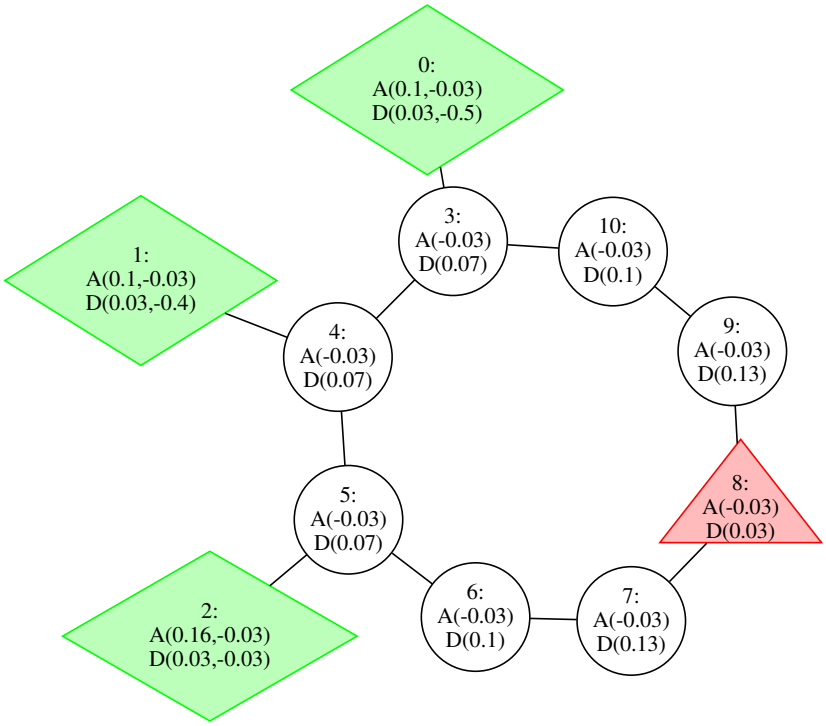
Rysunek A.7: *game3d*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



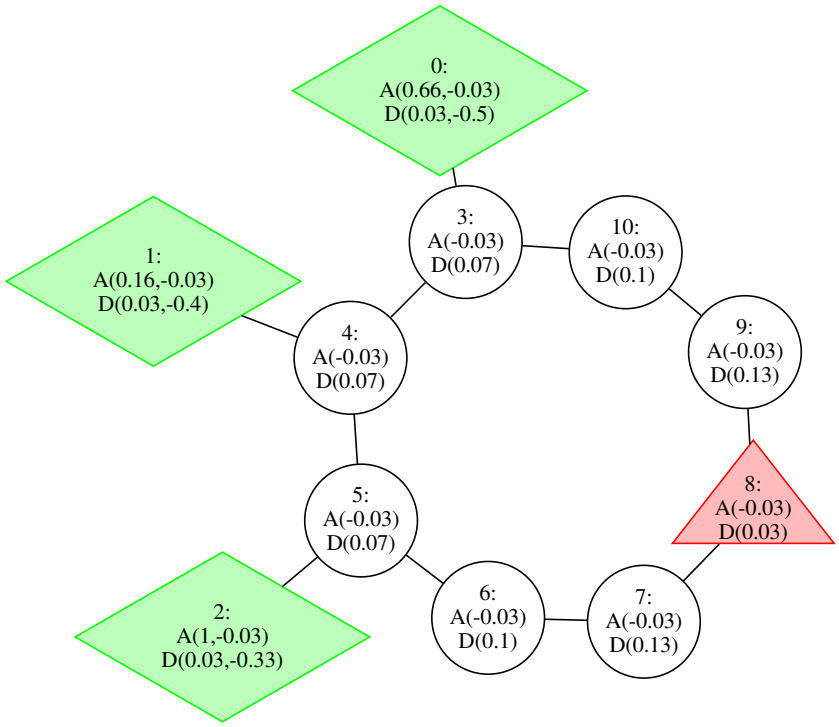
Rysunek A.8: *game4*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



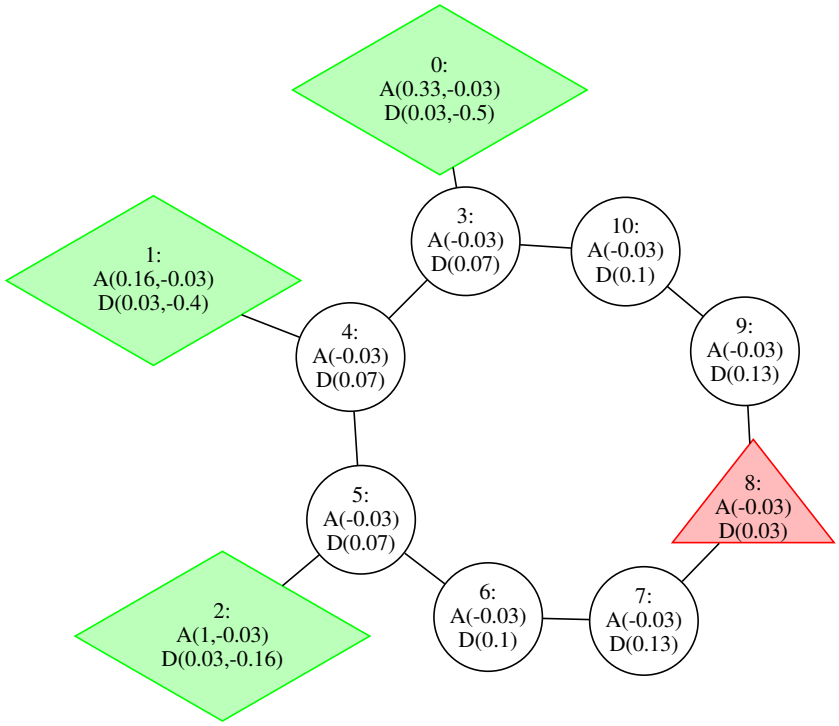
Rysunek A.9: *game4a*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



Rysunek A.10: *game4b*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



Rysunek A.11: *game4c*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.



Rysunek A.12: *game4d*. Wierzchołek z indeksem 0 jest punktem startowym lidera, wierzchołek z indeksem 8 (trójkąt), jest punktem startowym naśladowcy, a na zielono zaznaczone są wierzchołki z cennym zasobem. W nawiasie po literze F są wypłaty naśladowcy (follower), a po L wypłaty lidera.