POLITECHNIKA WARSZAWSKA

Wydział Mechatroniki

Praca przejściowa

Ireneusz Szulc

Przegląd metod planowania bezkolizyjnych tras dla zespołu robotów mobilnych

Opiekun pracy:
prof. nzw. dr hab. Barbara Siemiątkowska

Warszawa, 2018

Spis treści

Spis treści **2**

1 Wstęp **4**

1.1 Cel i zakres pracy 4

1.2 Założenia 5

1.3 Koordynacja ruchu robotów 6

1.4 Podobieństwo do gier RTS 6

1.5 Podstawowe pojęcia 7

2 Kooperacyjne planowanie tras **9**

2.1 Metody planowania tras 9

2.2 Metoda pól potencjałowych 10

2.3 Rozproszone planowanie tras 10

2.4 Planowanie uwzględniające priorytety 11

2.5 Metoda Path coordination 12

3 Algorytmy oparte o A* **13**

3.1 Algorytm A* 14

3.1.1 Zasada działania 14

3.1.2 Funkcja heurystyczna 14

3.2 Metody ponownego planowania 16

3.2.1 Local Repair A* 16

3.2.2 Algorytm D* 17

3.2.3 D* Extra Lite 17

3.3 Cooperative A* 17

3.3.1 Trzeci wymiar - czas 18

3.3.2 Tablica rezerwacji 18

3.4 Hierarchical Cooperative A* 20

3.5 Windowed Hierarchical Cooperative A* 20

4 Podsumowanie **22**

Bibliografia **23**

Wykaz skrótów **25**

Spis rysunków **26**

Rozdział 1

Wstęp

1.1 Cel i zakres pracy

Przedmiotem niniejszej pracy jest przegląd metod wykorzystywanych do planowania bezkolizyjnych tras dla wielu robotów mobilnych. Stanowi to również wstęp teoretyczny do zaprojektowania algorytmu i implementacji oprogramowania pozwalającego na symulację działania skutecznego planowania tras dla systemu wielorobotowego.

Praca skupia się na przypadkach, w których mamy do czynienia ze środowiskiem z dużą liczbą przeszkód (np. zamknięty budynek z licznymi ciasnymi korytarzami), aby uwypuklić problem blokowania sią agentów często prowadzący do zakleszczenia. Często okazuje się, że należy wtedy zastosować nieco inne podejścia niż te, które sprawdzają się w przypadku otwartych środowisk, a które zostały opisane np. w pracach [8], [10]. W otwartych środowiskach z małą liczbą przeszkód wystarczające może się okazać np. proste replanowanie wykorzystujące algorytm D* (por. 3.2.2) lub LRA* (por. 3.2.1).

W niniejszej pracy starano się znaleźć metody rozwiązujące zagadnienie, w którym znane są:

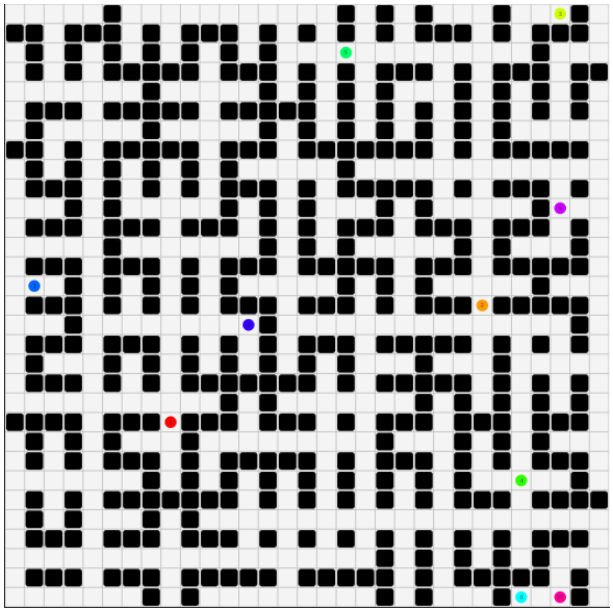
- pełna informacja o mapie otoczenia (położenie statycznych przeszkód),
- aktualne położenie i położenie celu każdego z robotów.

Szukany jest natomiast przebieg tras do punktów docelowych dla agentów. Zadaniem algorytmu będzie wyznaczenie możliwie najkrótszej bezkolizyjnej trasy dla wszystkich robotów. Należy jednak zaznaczyć, że priorytetem jest dotarcie każdego z robotów do celu bez kolizji z innymi robotami. Drugorzędne zaś jest, aby wyznaczone drogi były możliwie jak najkrótsze.

1.2 Założenia

Założenia i ograniczenia rozważanego problemu:

1. Każdy z robotów ma wyznaczony inny punkt docelowy, do którego zmierza.
2. Planowanie tras dotyczy mobilnych robotów holonomicznych.
3. Czas trwania zmiany kierunku robota jest pomijalnie mały.
4. Srodowisko, w którym poruszają się roboty, jest dwuwymiarową przestrzenią zawierającą dużą liczbę przeszkód oraz wąskie korytarze (por. rys. 1.1).
5. Roboty "wiedzą" o sobie i mogą komunikować się ze sobą podczas planowania tras.
6. Każdy robot zajmuje w przestrzeni jedno pole. Na jednym polu może znajdować się maksymalnie jeden robot (por. rys. 1.1).
7. Planowanie tras powinno odbywać się w czasie rzeczywistym.



Rysunek 1.1: Przykładowe środowisko z dużą liczbą przeszkód i rozmieszczonymi robotami. Źródło: własna implementacja oprogramowania symulacyjnego

1.3 Koordynacja ruchu robotów

Koordynacja ruchu robotów jest jednym z fundamentalnych problemów w systemach wielorobotowych [1].

Kooperacyjne znajdowanie tras (ang. *Cooperative Pathfinding*) jest zagadnieniem planowania w układzie wieloagentowym, w którym to agenci mają za zadanie znaleźć bezkolizyjne drogi do swoich, osobnych celów. Planowanie to odbywa się w oparciu o pełną informację o środowisku oraz o trasach pozostałych agentów [11].

Algorytmy do wyznaczania bezkolizyjnych tras dla wielu agentów (robotów) mogą znaleźć zastosowanie w szpitalach (np. roboty TUG i HOMER do dostarczania sprzętu na wyposażeniu szpitala [18]) oraz magazynach (np. roboty transportowe w magazynach firmy Amazon - por. rys. 1.2).

Rysunek 1.2: Roboty Kiva pracujące w magazynie firmy Amazon. Źródło: [16]

1.4 Podobieństwo do gier RTS

Problem kooperacyjnego znajdowania tras pojawia się nie tylko w robotyce, ale jest również popularny m.in. w grach komputerowych (strategiach czasu rzeczywistego), gdzie konieczne jest wyznaczanie bezkolizyjnych dróg dla wielu jednostek, unikając wzajemnego blokowania się. Niestety brak wydajnych i skutecznych algorytmów planowania dróg można zauważyć w wielu grach typu RTS (ang. *Real-Time Strategy*), gdzie czasami obserwuje się zjawisko zakleszczenia jednostek w wąskich gardłach (np. Age of Empires II, Warcraft III lub nawet we współczesnych produkcjach) [4] (por. rys. 1.3). Ponadto, zauważalny brak ogólnie dostępnych bibliotek open-

source do rozwiązania problemu typu *Cooperative Pathfinding* świadczy o potrzebie rozwoju tych metod.

Często algorytmy wykorzystywane w grach typu RTS (ang. *Real-Time Strategy*) zajmują się planowaniem bezkolizyjnych dróg dla układu wielu agentów w czasie rzeczywistym (będącego przedmiotem niniejszej pracy), dlatego nic nie stoi na przeszkodzie, aby stosować je zamiennie również do koordynacji ruchu zespołu robotów mobilnych.

Rysunek 1.3: Popularny problem zakleszczania się jednostek w wąskich gardłach występujący w grach typu RTS. Źródło: gra komputerowa Age of Empires II Forgotten Empires

1.5 Podstawowe pojęcia

Robot holonomiczny

Robot holonomiczny to taki robot mobilny, który może zmienić swoją orientację, stojąc w miejscu.

Przestrzeń konfiguracyjna

Przestrzeń konfiguracyjna to N-wymiarowa przestrzeń będąca zbiorem możliwych stanów danego układu fizycznego. Wymiar przestrzeni zależy od rodzaju i liczby wyróżnionych parametrów stanu. W odróżnieniu od przestrzeni roboczej, gdzie robot ma postać bryły, w przestrzeni konfiguracyjnej robot jest reprezentowany jako punkt.

Zupełność algorytmu (ang. *Completeness*)

W kontekście algorytmu przeszukiwania grafu algorytm zupełny to taki, który gwarantuje znalezienie rozwiązania, jeśli takie istnieje. Warto zaznaczyć, że nie gwarantuje to wcale, że znalezione rozwiązanie będzie rozwiązaniem optymalnym.

Rozdział 2

Kooperacyjne planowanie tras

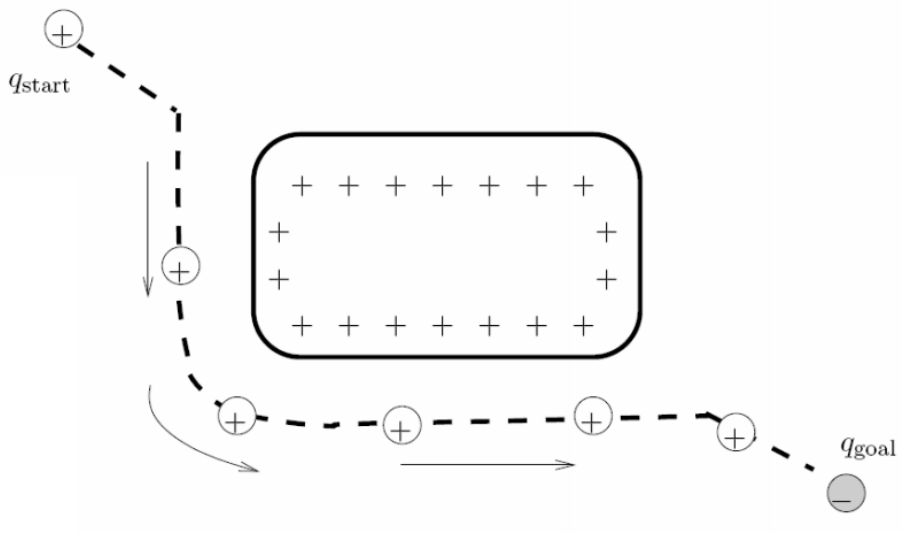
2.1 Metody planowania tras

Spośród metod wykorzystywanych do kooperacyjnego planowania tras dla wielu robotów można wyróżnić dwie zasadnicze grupy [7]:

- **Zcentralizowane** - Drogi wyznaczane są dla wszystkich agentów na raz (jednocześnie). Metody tego typu są często trudne do zrealizowania (gdyż do rozwiązania jest złożony problem optymalizacyjny) oraz mają bardzo dużą złożoność obliczeniową ze względu na ogromną przestrzeń przeszukiwania. Struktura organizacyjna jest scentralizowana - decyzje podejmowane są na podstawie centralnego systemu.

- **Rozproszone** (ang. *decoupled* lub *distributed*) - Podejście to dekomponuje zadanie na niezależne lub zależne w niewielkim stopniu problemy dla każdego agenta. Dla każdego robota droga wyznaczana jest osobno, w określonej kolejności, następnie rozwiązywane są konflikty (kolizje dróg). Zastosowanie metod rozproszonych wiąże się najczęściej z koniecznością przydzielania priorytetów robotom, co stanowi istotny problem, gdyż od ich wyboru może zależeć zupełność algorytmu. Nie należy mylić tej metody z zagadnieniem typu *Non-Cooperative Pathfinding*, w którym agenci nie mają wiedzy na temat pozostałych planów i muszą przewidywać przyszłe ruchy pozostałych robotów [11]. W podejściu rozproszonym agenci mają pełną informację na temat stanu pozostałych robotów, lecz wyznaczanie dróg odbywa się w określonej kolejności.

W systemach czasu rzeczywistego istotne jest, aby rozwiązanie problemu planowania tras uzyskać w krótkim, deterministycznym czasie, dlatego w tego typu sytemach chętniej używane są techniki rozproszone.



Rysunek 2.1: Zasada działania metody pól potencjałowych. Źródło: [17]

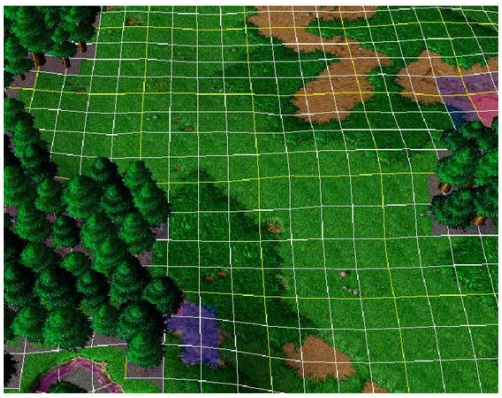
2.2 Metoda pól potencjałowych

Metoda pól potencjałowych (ang. *Artificial Potential Field* lub *Potential Field Technique*) polega na zastosowaniu zasad oddziaływania między ładunkami znanych z elektrostatyki. Roboty i przeszkody traktowane są jako ładunki jednoimienne, przez co "odpychają się" siłą odwrotnie proporcjonalną do kwadratu odległości (dzięki temu unikają kolizji między sobą). Natomiast punkt docelowy robota jest odwzorowany jako ładunek o przeciwnym biegunie, przez co robot jest "przyciągany" do celu. Główną zasadę działania metody przedstawiono na rysunku 2.1.

Technika ta jest bardzo prosta i nie wymaga wykonywania złożonych obliczeń (w odróżnieniu do pozostałych metod zcentralizowanych). Niestety bardzo powszechny jest problem osiągania minimum lokalnego, w którym suma wektorów daje zerową siłę wypadkową. Robot zostaje "uwięziony" w takim minimum lokalnym, przez co nie jest w stanie dotrzeć do wyznaczonego celu. Do omijania tego problemu muszą być stosowane inne dodatkowe metody [14]. Metoda pól potencjałowych nie daje gwarancji ani optymalności, ani nawet zupełności.

2.3 Rozproszone planowanie tras

Najczęściej stosowanymi podejściami są metody oparte o algorytm A* lub jego pochodne. W celu wykonywania wydajnych obliczeń w algorytmach przeszukujących grafy, nawet w przypadku ciągłej przestrzeni mapy, stosuje się podział na dyskretną siatkę pól (por. rys. 2.2) [3].



Rysunek 2.2: Ciągła przestrzeń mapy zdyskretyzowana do siatki pól, Źródło: edytor map z gry Warcraft III.

Popularne podejścia unikające planowania w wysoko wymiarowej zbiorowej przestrzeni konfiguracyjnej to techniki rozproszone i uwzględniające priorytety [1]. Pomimo, że metody te są bardzo efektywne, mają dwie główne wady:

- Nie są zupełne - nie dają gwarancji znalezienia rozwiązania, nawet gdy takie istnieje.
- Wynikowe rozwiązania mogą być nieoptymalne.

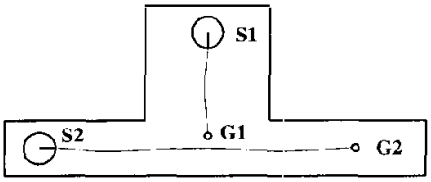
2.4 Planowanie uwzględniające priorytety

Często używaną w praktyce metodą jest planowanie z uwzględnianiem priorytetów. W tej technice agenci otrzymują unikalne priorytety. Algorytm wykonuje indywidualne planowanie sekwencyjnie dla każdego agenta w kolejności od najwyższego priorytetu. Trajektorie agentów o wyższych priorytetach są ograniczeniami (ruchomymi przeszkodami) dla pozostałych agentów [2].

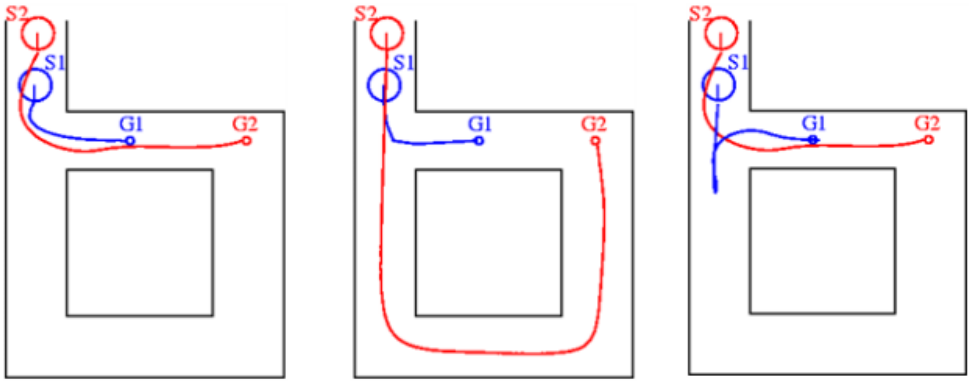
Złożoność ogólnego algorytmu rośnie liniowo wraz z liczbą agentów, dzięki temu to podejście ma zastosowanie w problemach z dużą liczbą agentów. Algorytm ten jest zachłanny i niezupełny w takim znaczeniu, że agentów zadowala pierwsza znaleziona trajektoria niekolidująca z agentami wyższych priorytetów.

Istotną rolę doboru priorytetów robotów w procesie planowania tras ukazuje prosty przykład przedstawiony na rysunku 2.3. Jeśli robot 1 (zmierzający z punktu S1 do G1) otrzyma wyższy priorytet niż robot 2 (zmierzający z S2 do G2), spowoduje to zablokowanie przejazdu dla robota 2 i w efekcie prawidłowe, istniejące rozwiązanie nie zostanie znalezione.

Układ priorytetów może mieć także wpływ na długość uzyskanych tras. Potwierdzający to przykład został przedstawiony na rysunku 2.4. W zależności od wyboru priorytetów, wpływających na kolejność planowania tras, otrzymujemy różne rozwiązania.



Rysunek 2.3: Sytuacja, w której żadne rozwiązanie nie zostanie znalezione, stosując planowanie uwzględniające priorytety, jeśli robot 1 ma wyższy priorytet niż robot 2. Źródło: [1]



Rysunek 2.4: a) Niezależne planowanie optymalnych tras dla 2 robotów; b) suboptymalne rozwiązanie, gdy robot 1 ma wyższy priorytet; c) rozwiązanie, gdy robot 2 ma wyższy priorytet. Źródło: [1]

2.5 Metoda Path coordination

Jedną z metod rozproszonego planowania tras z uwzględnianiem priorytetów jest Path Coordination, której idea przedstawia się w następujących krokach [1]:

1. Wyznaczenie ścieżki dla każdego robota **niezależnie** (np. za pomocą algorytmu A*)
2. Przydział priorytetów
3. Próba rozwiązania możliwych konfliktów między ścieżkami. Roboty utrzymywane są na ich indywidualnych ścieżkach (wyznaczonych na początku), wprowadzane modyfikacje pozwalają na zatrzymanie się, ruch naprzód, a nawet cofanie się, ale tylko **wzdłuż trajektorii** w celu uniknięcia kolizji z robotem o wyższym priorytecie.

Rozdział 3

Algorytmy oparte o A*

Kiedy pojedynczy agent staje przed zadaniem znalezienia drogi do wyznaczonego celu, prosty algorytm A* sprawdza się znakomicie. Jednak w przypadku, gdy wiele agentów porusza się w tym samym czasie, podejście to może się już nie sprawdzić, powodując wzajemne blokowanie się i zakleszczenie jednostek. Rozwiązaniem tego problemu jest kooperacyjne znajdowanie tras. Dzięki tej technice roboty będą mogły skutecznie przemieszczać się przez mapę, omijając trasy wyznaczone przez inne jednostki oraz schodząc innym jednostkom z drogi, jeśli to konieczne [11].

Zagadnienie znajdowania drogi jest również ważnym elementem sztucznej inteligencji zaimplementowanej w wielu grach komputerowych. Chociaż klasyczny algorytm A* potrafi doprowadzić pojedynczego agenta do celu, to jednak dla wielu agentów wymagane jest zastosowanie innego podejścia w celu unikania kolizji. Istniejące rozwiązania są jednak wciąż oparte o Algorytm A*, choć nieco zmodyfikowany. Chociaż A* może zostać zaadaptowany do replanowania trasy na żądanie, w przypadku wykrycia kolizji tras, to jednak takie podejście nie jest zadowalające pod wieloma względami. Na trudnych mapach z wieloma wąskimi korytarzami i wieloma agentami może to prowadzić do zakleszczenia agentów w wąskich gardłach lub do cyklicznego zapętlenia akcji agentów [11].

Rozdział ten zmierza do zaprezentowania zasady działania oraz cech algorytmu WHCA* (por. 3.5), zaproponowanego przez Davida Silvera [11]. Zaczynamy jednak od przedstawienia samego algorytmu A* i wprowadzając kolejne modyfikacje (CA*, HCA*), przekształcimy go stopniowo w WHCA*. W rozdziale zaprezentowano także rozwiązania alternatywne takie, jak: D* (por. 3.2.2) i LRA* (por. 3.2.1).

3.1 Algorytm A*

3.1.1 Zasada działania

A* jest algorytmem heurystycznym służącym do przeszukiwania grafu w celu znalezienia najkrótszej ścieżki między węzłem początkowym a węzłem docelowym. Algorytm ten jest powszechnie stosowany w zagadnieniach sztucznej inteligencji oraz w grach komputerowych [19]. Opiera się na zapisywaniu węzłów w dwóch listach: zamkniętych (odwiedzonych) i otwartych (do odwiedzenia). Jest modyfikacją algorytmu Dijkstry poprzez wprowadzenie pojęcia funkcji heurystycznej $h(n)$. Wartość funkcji heurystycznej powinna określać przewidywaną drogę do węzła docelowego z bieżącego punktu. Pełny koszt $f(n)$ stanowi sumę dotychczasowego kosztu $g(n)$ oraz przewidywanego pozostałego kosztu.

$$f(n) = g(n) + h(n) \tag{3.1}$$

gdzie:

$g(n)$ - dotychczasowy koszt dotarcia do węzła n, dokładna odległość miedzy węzłem n a węzłęm początkowym

$h(n)$ - heurystyka, przewidywana pozostała droga od węzła bieżącego do węzła docelowego

$f(n)$ - oszacowanie pełnego kosztu ścieżki od węzła startowego do węzła docelowego prowadzącej przez węzeł n

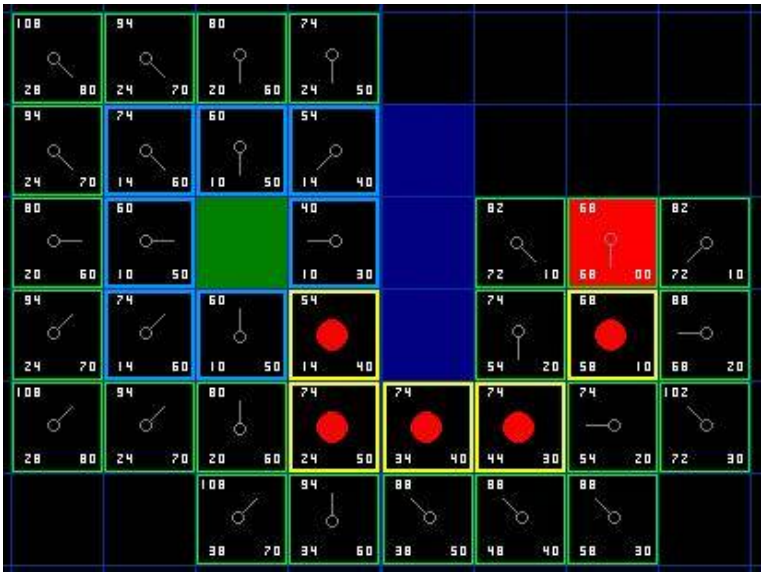
n - bieżący węzeł, wierzchołek przeszukiwanego grafu

W każdym kroku przeszukiwany jest węzeł o najmniejszej wartości funkcji $f(n)$. Dzięki takiemu podejściu najpierw sprawdzane są najbardziej "obiecujące" rozwiązania, co pozwala szybciej otrzymać wynik (w porównaniu do algorytmu Dijkstry). Algorytm kończy działanie w momencie, gdy napotka węzeł będący węzłem docelowym. Dla każdego odwiedzonego węzła zapamiętywane są wartości $g(n)$, $h(n)$ oraz węzeł będący rodzicem w celu późniejszego odnalezienia drogi powrotnej do węzła startowego po napotkaniu węzła docelowego (por. rys. 3.1).

Algorytm zwraca optymalny wynik (najkrótszą możliwą ścieżkę), ale w pewnych warunkach (por. 3.1.2).

3.1.2 Funkcja heurystyczna

Od wyboru sposobu obliczania heurystyki zależy czas wykonywania algorytmu oraz optymalność wyznaczonego rozwiązania.



Rysunek 3.1: Ilustracja wyznaczania działania przez A. Każdy odwiedzony węzeł wskazuje na swojego rodzica, co umożliwia późniejszą rekonstrukcję drogi. Źródło: [15]*

Funkcja heurystyczna $h(n)$ jest dopuszczalna, jeśli dla dowolnego węzła n spełniony jest warunek

$$h(n) \leqslant h^*(n) \tag{3.2}$$

gdzie:

$h^*(n)$ - rzeczywisty koszt ścieżki od węzła n do celu

Innymi słowy, heurystyka dopuszczalna to taka, która nigdy nie przeszacowuje pozostałej do przebycia drogi. Wtedy algorytm A* zwraca optymalną (najkrótszą) ścieżkę [11].

A* opiera się na heurystyce, która "kieruje" przeszukiwaniem. To od niej zależy, w kierunku których węzłów będzie podążało przeszukiwanie. Źle wybrana heurystyka może prowadzić do zbędnego odwiedzania dodatkowych węzłów.

Poniżej przedstawiono najczęściej wykorzystywane heurystyki będące oszacowaniem odległości między przeszukiwanym węzłem (x_n, y_n) a węzłem docelowym (x_g, y_g) na dwuwymiarowej mapie.

Heurystyka euklidesowa

Heurystyka wykorzystująca metrykę euklidesową wyraża dokładną odległość po linii prostej. Wymaga to jednak częstego obliczania pierwiastków (lub stosowania *look-up table*).

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \tag{3.3}$$

Heurystyka Manhattan

W przypadku, gdy agent może poruszać się po mapie jedynie poziomo lub pionowo (nie na ukos) wystarczająca okazuje się metryka Manhattan (metryka miejska):

$$h(n) = |x_n - x_g| + |y_n - y_g| \tag{3.4}$$

Heurystyka metryki maksimum

Zastosowanie metryki maksimum (metryki Czebyszewa) może sprawdzić się np. dla niektórych figur szachowych:

$$h(n) = max(|x_n - x_g|, |y_n - y_g|) \tag{3.5}$$

Heurystyka zerowa

Przyjęcie heurystyki równej $h(n) = 0$ sprawia, że algorytm A* sprowadza się do algorytmu Dijkstry.

3.2 Metody ponownego planowania

3.2.1 Local Repair A*

W algorytmie Local Repair A* (LRA*) każdy z agentów znajduje drogę do celu, używając algorytmu A*, ignorując pozostałe roboty oprócz ich obecnych sąsiadów. Roboty zaczynają podążać wyznaczonymi ścieżkami do momentu, aż kolizja z innym robotem jest nieuchronna (w lokalnym otoczeniu). Wtedy następuje ponowne przeliczenie drogi pozostałej do przebycia, z uwzględnieniem nowo napotkanej przeszkody.

Możliwe (i całkiem powszechne [11]) jest uzyskanie cykli (tych samych sekwencji ruchów powtarzających się w nieskończoność), dlatego zazwyczaj wprowadzane są pewne modyfikacje, aby rozwiązać ten problem. Jedną z możliwości jest zwiększanie wpływu losowego szumu na wartość heurystyki. Kiedy agenci zachowują się bardziej losowo, prawdopodobne jest, że wydostaną się z problematycznego położenia i spróbują podążać innymi ścieżkami.

Algorytm ten ma jednak sporo poważnych wad, które szczególnie ujawniają się w trudnych środowiskach z dużą liczbą przeszkód. Wydostanie się z zatłoczonego wąskiego gardła może trwać bardzo długo. Prowadzi to także do ponownego przeliczania trasy w prawie każdym kroku. Wciąż możliwe jest również odwiedzanie tych samych lokalizacji w wyniku zapętleń.

3.2.2 Algorytm D*

D* (*Dynamic A* Search*) jest przyrostowym algorytmem przeszukiwania. Jest modyfikacją algorytmu A* pozwalającą na szybsze replanowanie trasy w wyniku zmiany otoczenia (np. zajmowania wolnego pola przez innego robota). Wykorzystywany jest m.in. w nawigacji robota do określonego celu w nieznanym terenie. Początkowo robot planuje drogę na podstawie pewnych założeń (np. nieznany teren nie zawiera przeszkód). Podążając wyznaczoną ścieżką, robot odkrywa rzeczywisty stan mapy i jeśli to konieczne, wykonuje ponowne planowanie trasy na podstawie nowych informacji. Często wykorzystywaną implementacją (z uwagi na zoptymalizowaną złożoność obliczeniową) jest wariant algorytmu *D* Lite* [6].

3.2.3 D* Extra Lite

Wartym uwagi jest także algorytm *D* Extra Lite* charakteryzujący się jeszcze korzystniejszą wydajnością niż *D* Lite*, co potwierdziły przeprowadzone obszerne testy w różnego rodzaju środowiskach (m.in. na mapach z gier komputerowych oraz w zawiłych labiryntach) [9].
D Extra Lite* służy do przyrostowego planowania najkrótszej ścieżki w dwuwymiarowej przestrzeni bez dokładnej wiedzy o środowisku. Wykorzystanie wyników z poprzednich iteracji oraz wczesne odrzucanie pewnych węzłów z drzewa przeszukiwania znacznie skraca czas potrzebny do wykonania ponownego planowania. *D* Extra Lite* jest nowatorskim algorytmem ogólnego przeznaczenia. Naturalnym jego zastosowaniem jest nawigacja robotów mobilnych [9].

3.3 Cooperative A*

Cooperative A* jest algorytmem do rozwiązywania problemu kooperacyjnego znajdowania tras. Metoda może być również nazywana czasoprzestrzennym algorytmem A* (*time-space A* search*) Zadanie planowania jest rozdzielone na serię pojedynczych poszukiwań dla poszczególnych agentów. Pojedyncze poszukiwania są wykonywane w trójwymiarowej czasoprzestrzeni i biorą pod uwagę zaplanowane ścieżki przez pozostałych agentów. Akcja wykonania postoju (pozostania w tym samym miejscu) jest uwzględniona w zbiorze akcji możliwych do wykonania. Po przeliczeniu dróg dla każdego agenta, stany zajętości pól są zaznaczane w tablicy rezerwazji (ang. Reservation table). Pozycje w tej tablicy są uważane jako pola nieprzejezdne i w efekcie są omijane podczas przeszukiwania przez późniejszych agentów [11].

Należy zaznaczyć, że planowanie dla każdego agenta odbywa się sekwencyjnie według przydzielonych priorytetów. Algorytm ten jest podatny na zmianę kolejności agentów. Odpowiedni dobór priorytetów może wpłynąć na wydajność algorytmu oraz jakość uzyskanego wyniku.

3.3.1 Trzeci wymiar - czas

Do rozwiązania problemu kooperacyjnego znajdowania dróg algorytm przeszukiwania potrzebuje mieć pełną wiedzę na temat przeszkód oraz jednostek na mapie. Aby zapisać tą informację, potrzeba rozszerzyć mapę o trzeci wymiar - czas. Pierwotną mapę będziemy nazywac mapą przestrzenną, natomiast nową - czasoprzestrzenną mapą [11].

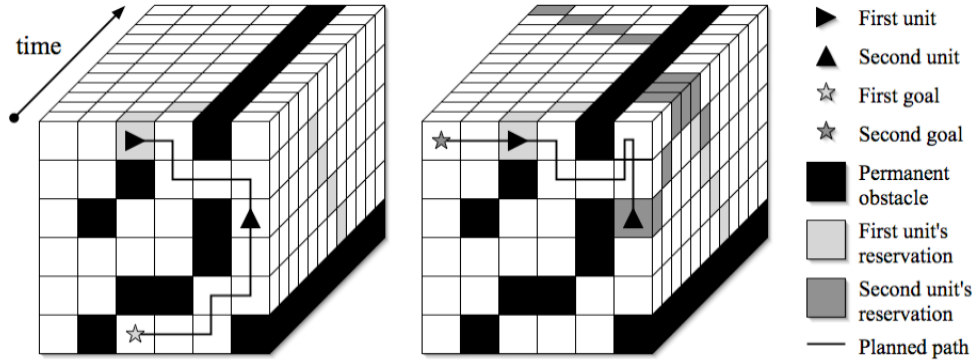
Zagadnienie sprowadza się do przeszukiwania grafu, w którym każdy węzeł ma przypisane 3 wielkości: położenie x, położenie y oraz czas. Podczas gdy zakres wielkości x i y jest znany i wynika z rozmiarów mapy oraz podziału jej wymiarów na dyskretne pola, to jednak określenie wymiaru czasu i jego granicy nie jest trywialnym zagadnieniem. Wymiar czasu możemy również zdyskretyzować i przyjać, że krok czasu jest okresem, jaki zajmuje robotowi przejście z jednego pola na sąsiednie (poziomo lub pionowo). Natomiast górną granicą czasu powinna być maksymalna liczba ruchów potrzebna do dotarcia do celu przez ostatniego robota. Wybór za małej liczby może spowodować, że algorytm nie zdąży znaleźć drogi dla niektórych agentów, z kolei zbyt duża granica kroków czasu znacząco wydłuża obliczenia. Rozwiązanie tego problemu zostało opisane w późniejszym podrozdziale 3.5.

Wprowadzenie trzeciego wymiaru wprowadza także konieczność zmian w doborze odpowiedniej heurystyki odpowiedzialnej za oszacowanie drogi pozostałej do celu.

3.3.2 Tablica rezerwacji

Tablica rezerwacji (ang. *Reservation Table*) reprezentuje współdzieloną wiedzę o zaplanowanych ścieżkach przez wszystkich agentów. Jest to informacja o zajętości każdej z komórki na mapie w danym miejscu i określonym czasie [11]. Jak tylko agent zaplanuje trasę, każda komórka odpowiadająca ścieżce zaznaczana jest jako zajęta w tablicy rezerwacji.

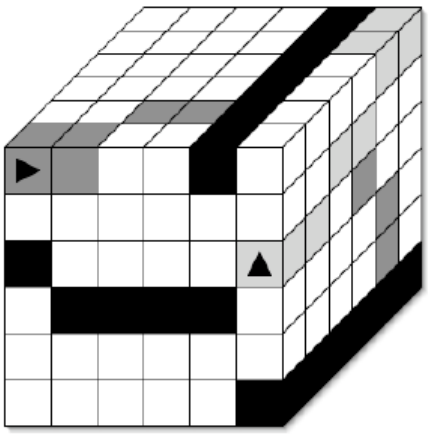
W prostej implementacji tablica rezerwacji jest trójwymiarową kostką (dwa wymiary przestrzenne i jeden wymiar czasu). Każda komórka kostki, która jest przecinana przez zaplanowaną przez agenta ścieżkę, jest zaznaczana jako nieprzejezdna przez określony czas trwania. W ten sposób zapobiega to planowania kolizyjnych tras przez pozostałych agentów. (por. rys. 3.2)



Rysunek 3.2: Dwie jednostki kooperacyjnie poszukujące tras. (A) Pierwsza jednostka znajduje ścieżkę i zaznacza ją w tablicy rezerwacji. (B) Druga jednostka znajduje ścieżkę, uwzględniając istniejące rezerwacje pól, również zaznaczając ją w tablicy rezerwacji. Źródło: [11]

Jeśli tylko niewielka część z całej tablicy rezerwacji będzie markowana jako zajęta, wydajniej jest zaimplementować ją jako tablicę typu *hash table*. Daje to zaletę oszczędności pamięci poprzez pamiętanie jedynie współrzędnych (x, y, t) zajętych pól.

W ogólności poszczególni agenci mogą mieć różną prędkość lub rozmiary, zatem tablica rezerwacji musi mieć możliwość zaznaczenia dowolnego zajętego obszaru. Zostało to przedstawione na rysunku 3.3.



Rysunek 3.3: Tablica rezerwacji jest współdzielona między wszystkimi agentami. Jej rozmiar powinien być odpowiednio dopasowany do agentów o różnych prędkościach. Źródło: [11]

Niestety powyższy sposób wykorzystania tablicy rezerwacji w pewnych sytuacjach nie zapobiega zderzeniom czołowym jednostek zmierzających w przeciwnych kierunkach. Jeśli jedna jednostka zarezerwowała komórki (x, y, t) i $(x + 1, y, t + 1)$, nic nie stoi na przeszkodzie, aby kolejna jednostka mogła zarezerwować komórki $(x + 1, y, t)$ i $(x, y, t + 1)$. Ten problem może być rozwiązany poprzez zajmowanie (rezerwowanie) dwóch sąsiednich komórek w tym samym czasie t podczas ruchu robota, a nie tylko jednej komórki.

3.4 Hierarchical Cooperative A*

Metoda *Hierarchical Cooperative A** (HCA*) wprowadza pewną modyfikację do algorytmu Cooperative A*. Modyfikacja ta dotyczy heurystyki opartej na abstrakcjach przestrzeni stanu [11]. HCA* jest także jednym z przykładów rozproszonego podejścia do planowania tras.

W tym podejściu abstrakcja przestrzeni stanu oznacza zignorowanie wymiaru czasu, jak również tablicy rezerwacji. Innymi słowy, abstrakcja jest prostą dwuwymiarową mapą z usuniętymi agentami. Abstrakcyjne odległości mogą być rozumiane jako dokładne oszacowania odległości do celu, ignorując potencjalne interakcje z innymi agentami. Jest to oczywiście dopuszczalna heurystyka (por. 3.1.2). Niedokładność heurystyki wynika jedynie z trudności związanych z interakcją z innymi agentami (jak bardzo agent musi zboczyć z pierwotnie zaplanowanej ścieżki w celu ominięcia innych agentów).

Do wyznaczenia heurystyki dla abstrakcyjnej przestrzeni stanu opisywane podejście wykorzystuje algorytm przeszukiwania *Reverse Resumable A** (RRA*). Algorytm ten wykonuje zmodyfikowane przeszukiwanie A* w odwrotnym kierunku. Przeszukiwanie zaczyna się w węźle docelowym agenta i kieruje się do początkowego położenia. Jednak zamiast kończyć w tym punkcie, przeszukiwanie jest kontynuowane do natrafienia na węzeł N, w którym znajduje się agent.

Algorytm HCA* jest więc taki, jak algorytm CA*, ale z bardziej wyszukaną heurystyką, która używa RRA* do obliczania abstrakcyjnych odległości na żądanie.

3.5 Windowed Hierarchical Cooperative A*

Problematyczne zagadnienie związane z wyżej wspomnianymi algorytmami jest takie, że kończą one działanie w momencie, gdy agent osiąga swój cel. Jeśli agent znajduje się już w miejscu docelowym, np. w wąskim korytarzu, to może on blokować części mapy dla innych agentów. W takiej sytuacji agenci powinni kontynuować kooperację z pozostałymi jednostkami, nawet po osiągnięciu swoich celów. Może to zostać zrealizowane np. poprzez usunięcie się z wąskiego gardła w celu przepuszczenia pozostałcyh agentów, a następnie powrót do docelowego punktu [11].

Kolejny problem związany jest z wrażliwością na kolejność agentów (przydzielone priorytety). Chociaż czasem możliwy jest skuteczny, globalny przydział priorytetów [7], to jednak dobrym rozwiązaniem może być dynamiczne modyfikowanie kolejności agentów. Wtedy rozwiązania mogą zostać znalezione w tych przypadkach, w których zawiodło przydzielanie niezmiennych priorytetów [11].

Rozwiązaniem powyższych kwestii jest zamknięcie algorytmu przeszukiwania w oknie cza-

sowym. Kooperacyjne planowanie jest ograniczone do ustalonej głębokości. Każdy agent szuka częściowej ścieżki do celu i zaczyna nią podążać. W regularnych okresach (np. gdy agent jest w połowie drogi) okno jest przesuwane dalej i wyznaczana jest nowa ścieżka.

Aby zapewnić, że agenci podążają do prawidłowych punktów docelowych, ograniczana jest tylko głębokość przeszukiwania kooperacyjnego (związanego z wieloma agentami), podczas gdy przeszukiwanie abstrakcyjnych odległości (heurystyki opisanej w podrozdziale 3.4) odbywa się bez ograniczeń głębokości. Okno o rozmiarze w może być rozumiane jako pośrednia abstrakcja, która jest równoważna wykonaniu w kroków w rzeczywistym środowisku (z uwzględnieniem pozostałych agentów) a następnie wykonaniu pozostałych kroków zgodnie z abstrakcją (bez uwzględnienia innych agentów). Innymi słowy, pozostali agenci są jedynie rozważani dla w pierwszych kroków (poprzez tablicę rezerwacji) a dla pozostałych kroków są ignorowani [11].

Rozmiar okna jest wielkością ustalaną arbitralnie. Rozmiar okna powinien być przyjęty jako czas trwania najdłuższego przewidywanego wąskiego gardła (zatoru). Dobrą praktyką jest przyjęcie wartości równej liczbie agentów na mapie, gdyż to właśnie z ich powodów mogą wystąpić ewentualne zmiany w zaplanowanej trasie.

Porównanie HCA* i WHCA*

Algorytm HCA* wybiera ustaloną kolejność agentów i planuje trasy dla każdego agenta po kolei, unikając kolizji z poprzednio wyznaczonymi ścieżkami. Natomiast użycie przeszukiwania z przesuwanym oknem w WHCA* poprawia skuteczność algorytmu oraz przyspiesza proces wyznaczania rozwiązania [12].

Fakt wykonywania przeszukiwania w oknie oznacza, że planowanie algorytmem WHCA* wykonywane jest zawsze z ustaloną liczbą kroków w przyszłości i wybierany jest najbardziej obiecujący węzeł na granicy tego okna [13]. W metodach Hierarchical Cooperative A* oraz Cooperative A* wybór granicy wymiaru czasu (głębokości przeszukiwania w liczbie kroków) stanowi balans pomiędzy wydajnością a zupełnością algorytmu.

W obu podejściach HCA* i WHCA* zastosowano dodatkowy algorytm przeszukiwania wstecz (RRA*) wspomagający heurystykę. Służy on wyznaczeniu dokładnej odległości z węzła do celu, pomijając wpływ innych agentów. Jest to często heurystyka wysokiej jakości (prawie idealne oszacowanie), gorzej sprawdza się jedynie w środowiskach z dużą ilością wąskich gardeł i zatorów [13].

Chociaż takie przeszukiwanie wstecz prowadzi do początkowego wykonania większej ilości obliczeń (wykonanie pełnego przeszukania A* z punktu docelowego do punktu startowego, jak również do innych węzłów), to jednak koszt obliczeniowy w kolejnych krokach jest już zdecydowanie niższy [13].

Rozdział 4

Podsumowanie

Większość popularnych algorytmów wykorzystywanych do planowania tras dla wielu robotów mobilnych (agentów) opiera się o A*.

Kooperacyjne planowanie tras jest ogólną techniką koordynacji dróg wielu jednostek. Znajduje zastosowanie, gdzie wiele jednostek może komunikować się ze sobą, przekazując informację o ich ścieżkach. Poprzez planowanie wprzód w czasie, jak również i w przestrzeni, jednostki potrafią schodzić sobie z drogi nawzajem w celu uniknięcia kolizji. Metody kooperacyjnego planowania są bardziej skuteczne i znajdują trasy wyższej jakości niż te uzyskane przez A* z metodą *Local Repair*.

Wiele z udoskonaleń przestrzennego algorytmu A* może być również zaadaptowane do czasprzestrzennego A*. Ponadto, wprowadzenie wymiaru czasu otwiera nowe możliwości do rozowju algorytmów znajdowania dróg.

Najbardziej obiecującym pod względem skuteczności algorytmem wydaje się być metoda WHCA*.

Aby wydajnie prowadzić obliczenia, zakłada się, że każdy ruch robota trwa tyle samo. Wprowadza to upraszczające, błędne założenie, że ruch robota na pole w kierunku poziomym lub pionowym trwa tyle samo, co na ukos.

W wielu przypadkach metody do planowania bezkolizyjnych tras w systemach wieloagentowych mogą być wykorzystywane zamiennie zarówno do wyznaczania trajektorii robotów mobilnych, jak i w grach komputerowych, np. strategiach czasu rzeczywistego do planowania tras wielu jednostek.

Zaprezentowane algorymy mogą znaleźć zostosowanie również w środowiskach z ciągłą przestrzenią oraz w dynamicznych środowiskach, w których to ścieżki muszą być przeliczane po wykryciu zmiany na mapie.

Bibliografia

[1] Bennewitz M.; Burgard W.; Thrun S. *Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems*. 2001.

[2] Cap M.; Novak P.; Vokrinek J.; Pechoucek M. *Asynchronous Decentralized Algorithm for Space-Time Cooperative Pathfinding*. Workshop Proceedings of the European Conference on Articial Intelligence (ECAI 2012), 2012.

[3] Duc L. M.; Sidhu A. S.; Chaudhari N. S. *Hierarchical Pathfinding and AI-Based Learning Approach in Strategy Game Design*. International Journal of Computer Games Technology, 2008.

[4] Geramifard A.; Chubak P. *Efficient Cooperative Path-Planning*. Computing Science Department, University of Alberta, 2005.

[5] Hart P. E.; Nilsson N. J.; Raphael B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4, 1968.

[6] Koenig S.; Likhachev M. *D* Lite*. Proceedings of the AAAI Conference of Artificial Intelligence, 2002.

[7] Latombe J. *Robot Motion Planning*. Boston, MA: Kluwer Academic, 1991.

[8] Mówiński K.; Roszkowska E. *Sterowanie hybrydowe ruchem robotów mobilnych w systemach wielorobotycznych*. Postępy Robotyki, 2016.

[9] Przybylski M.; Putz B. *D* Extra Lite: A Dynamic A* With Search-Tree Cutting and Frontier-Gap Repairing*. International Journal of Applied Mathematics and Computer Science, 2017.

[10] Siemiątkowska B. *Uniwersalna metoda modelowania zachowań robota mobilnego wykorzystująca architekturę uogólnionych sieci komórkowych*. 2009.

[11] Silver D. *Cooperative Pathfinding.* Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, 2005.

[12] Standley T.; Korf R. *Complete Algorithms for Cooperative Pathfinding Problems.* Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[13] Toresson A. *Real-time Cooperative Pathfinding.* 2010.

[14] Zhu Q.; Yan Y.; Xing Z. *Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing.* Intelligent Systems Design and Applications, 2006.

[15] A* pathfinding for beginners. `http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm`. Dostęp: 2018-01-02.

[16] Amazon warehouse demand devours robots and workers. `https://www.roboticsbusinessreview.com/supply-chain/amazon_warehouse_demand_devours_robots_and_workers`. Dostęp: 2018-01-05.

[17] Choset H. Robotic motion planning: Potential functions. `https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf`. Dostęp: 2018-01-02.

[18] Roboty TUG i HOMER firmy Aethon. `http://www.aethon.com/tug/tughealthcare/`. Dostęp: 2018-01-02.

[19] Searching using A*. `http://web.mit.edu/eranki/www/tutorials/search/`. Dostęp: 2018-01-02.

Wykaz skrótów

CA*	Cooperative A*
HCA*	Hierarchical Cooperative A*
LRA*	Local Repair A*
MAS	Multi-Agent System
RRA*	Reverse Resumable A*
RTS	Real Time Strategy
WHCA*	Windowed Hierarchical Cooperative A*

Spis rysunków

1.1 Przykładowe środowisko z dużą liczbą przeszkód i rozmieszczonymi robotami. Źródło: własna implementacja oprogramowania symulacyjnego 5
1.2 Roboty Kiva pracujące w magazynie firmy Amazon. Źródło: [16] 6
1.3 Popularny problem zakleszczania się jednostek w wąskich gardłach występujący w grach typu RTS. Źródło: gra komputerowa Age of Empires II Forgotten Empires 7

2.1 Zasada działania metody pól potencjałowych. Źródło: [17] 10
2.2 Ciągła przestrzeń mapy zdyskretyzowana do siatki pól, Źródło: edytor map z gry Warcraft III. 11
2.3 Sytuacja, w której żadne rozwiązanie nie zostanie znalezione, stosując planowanie uwzględniające priorytety, jeśli robot 1 ma wyższy priorytet niż robot 2. Źródło: [1] 12
2.4 a) Niezależne planowanie optymalnych tras dla 2 robotów; b) suboptymalne rozwiązanie, gdy robot 1 ma wyższy priorytet; c) rozwiązanie, gdy robot 2 ma wyższy priorytet. Źródło: [1] 12

3.1 Ilustracja wyznaczania działania przez A*. Każdy odwiedzony węzeł wskazuje na swojego rodzica, co umożliwia późniejszą rekonstrukcję drogi. Źródło: [15] . . 15
3.2 Dwie jednostki kooperacyjnie poszukujące tras. (A) Pierwsza jednostka znajduje ścieżkę i zaznacza ją w tablicy rezerwacji. (B) Druga jednostka znajduje ścieżkę, uwzględniając istniejące rezerwacje pól, również zaznaczając ją w tablicy rezerwacji. Źródło: [11] 19
3.3 Tablica rezerwacji jest współdzielona między wszystkimi agentami. Jej rozmiar powinien być odpowiednio dopasowany do agentów o różnych prędkościach. Źródło: [11] 19