КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Кваліфікаційна робота
на здобуття ступеня магістра

за спеціальністю 124 Системний аналіз

на тему:

ПОШУК ОПТИМАЛЬНИХ ШЛЯХІВ ДО ВИХОДІВ В МУЛЬТИАГЕНТНОМУ СЕРЕДОВИЩІ В УМОВАХ НАДЗВИЧАЙНОЇ СИТУАЦІЇ З УРАХУВАННЯМ ДИНАМІКИ СТАНУ ПОДІЇ

Виконав студент 2-го курсу магістратури
Онишкевич Артур Олегович

(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Івохін Євген Віктороович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичних основ комп'ютерних наук
«04» травня 2023 р.,
протокол No 11
Завідувач кафедри
О. Г. Наконечний

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 56 сторінок, 5 ілюстрації, 6 алгоритмів, 16 джерел посилань.

АЛГОРИТМИ ПОШУКУ. БАГАТОАГЕНТНИЙ ПОШУК ШЛЯХІВ, ПОШУК В ДИНАМІЧНОМУ ГРАФІ, ЕВАКУАЦІЯ, КЛІТИННИЙ АВТОМАТ, МОДЕЛЮВАННЯ РЕАКЦІЙНО-ДИФУЗІЙНИХ ПРОЦЕСІВ, МОДЕЛЮВАННЯ ПОЖЕЖІ.

Об’єктом роботи є пошук оптимальних шляхів до виходів для багатоагентної системи в динамічному графі, розробка програмного забезпечення, що моделює процес поширення небезпеки і евакуації з візуалізацією на мові Java.

Метою роботи є дослідження алгоритмів багатоагентного пошуку шляхів, пошуку шляхів в динамічних графах, моделювання процесів за допомогою клітинного автомату та розробка системи евакуації.

Методи розробки: розробка алгоритмів багатоагентного пошуку в динамічних графах та моделювання процесу пожежі на мові Java (OpenJDK 17); розробка графічного інтерфейсу з використанням бібліотеки JavaFX; використання патерну проектування MVC для поєднання логіки програмного забезпечення з GUI; використання технології Maven для збірки ПЗ.

Результати роботи: досліджено алгоритми пошуку шляхів в динамічних графах та алгоритми багатоагентного пошуку шляхів, адаптовано алгоритм TreeAA* для багатоагентного пошуку CBS, побудована КА-модель “поширення фронту” для моделювання пожежі, розроблено програмний продукт, який дозволяє моделювати ситуації евакуації.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AA* – Adaptive A*;

MAPF – Multi Agent Path Finding – багатоагентний пошук шляхів;

CBS – Conflict Based Search – пошук на основі конфліктів;

EES – Explicit Estimation Search – пошук за явною оцінкою;

ECBS – Extended Conflict Based Search – покращений пошук на основі конфліктів;

EECBS – Explicit Estimation Conflict Based Search;

КА – клітинний автомат;

DLA – Diffusion-Limited Agregation – агрегація обмежена дифузією;

MAEDO – Multi Agent Evacuation with Dynamic Obtancles – багатоагентна евакуація в динамічному середовищі;

ПЗ – програмне забезпечення;

GUI – Graphical User Interface – графічний інтерфейс користувача;

JVM – Java Virtual Machine – віртуальна машина Java;

JDK – Java Development Kit – набір інструментів Java;

ВСТУП

Кризові обставини, такі як катаклізми, сучасні нещасні випадки та напади психологічних гнобителів, становлять величезну небезпеку для безпеки та процвітання постраждалих людей. Щоб зменшити кількість смертей і матеріальних збитків, дуже важливо, щоб люди були евакуйовані з небезпечної зони якомога швидше та ефективніше. Динамічний характер небезпеки, непередбачуване поширення та різноманітна поведінка агентів, що здійснюють евакуацію, ускладнюють ці ситуації. Завдяки використанню багатоагентного пошуку шляхів у динамічному графі та моделюванню поширення різноманітних ризиків за допомогою клітинних автоматів нова система евакуації, яка є предметом цієї дисертації, вирішує ці проблеми.

Актуальність цього дослідження також пов’язана з необхідністю ефективного управління евакуацією. Своєчасна та ефективна евакуація має вирішальне значення для порятунку життів та пом’якшення наслідків надзвичайних ситуацій. Запропонована система має на меті підвищити ефективність зусиль евакуації шляхом визначення оптимальних маршрутів, які мінімізують ризик небезпеки, що зрештою сприяє кращому реагуванню на надзвичайні ситуації. Крім того, розробка надійної та адаптивної системи евакуації має значні практичні наслідки для управління надзвичайними ситуаціями, міського планування та проектування інфраструктури. Демонструючи ефективність запропонованої системи в різних сценаріях надзвичайних ситуацій, ця робота має на меті надати цінний інструмент для осіб, які приймають рішення, і практиків, залучених до планування та управління евакуацією.

Мета роботи полягає в розробці надійної та адаптованої системи евакуації, яка ефективно спрямовує людей до безпечних місць під час надзвичайних ситуацій. Завдяки координації визначення шляхів для людей, що знаходяться в небезпечній ситуації, у поєднанні з демонстрацією небезпек, запропонована система має на меті забезпечити інноваційне рішення для планування та управління евакуацією в реальному часі, яке

може значно підвищити безпеку та ефективність евакуації, зрештою рятуючи життя та пом’якшуючи наслідки катастроф.

Об’єктом дослідження є розробка надійної та адаптивної системи евакуації, яка може ефективно направляти людей у безпечне місце під час надзвичайних ситуацій, беручи до уваги динамічний характер стану події та різноманітні схеми поширення різних небезпек. Основна увага приділяється інтеграції багатоагентного пошуку шляху в динамічні графіки з моделюванням небезпек на основі клітинних автоматів для створення інноваційного рішення для планування та управління евакуацією в реальному часі.

Предметом дослідження є пошук оптимальних шляхів в багатоагентному динамічному середовищі.

Багатоагентне визначення шляху в динамічному графі є першою частиною цього дослідження. Існуючі схеми очищення часто залежать від статичного зображення клімату, який може нехтувати представленням обставин, що постійно змінюються, і впливу небезпеки на курси відправлення. Ця робота досліджує використання багатоагентних алгоритмів пошуку шляху, які можуть адаптуватися до мінливого характеру надзвичайної ситуації в динамічних графах, щоб подолати це обмеження. Запропонована система здатна ефективно визначати найкращі шляхи евакуації, які гарантують безпеку та ефективність пересування агентів, шляхом постійного оновлення графа на основі даних у реальному часі.

Використання клітинних автоматів для моделювання поширення різних видів небезпеки є другим розділом цієї дисертації. Техніка дискретного обчислювального моделювання, відома як клітинні автомати, пропонує зручний і адаптований метод для моделювання складних систем із численними взаємопов’язаними частинами. Запропоновані методи моделювання на основі клітинних автоматів можуть надати значний внесок у швидкий розрахунок поширення небезпечних явищ в режимі реального часу і

надати інформацію системам пошуку для визначення найбільш безпечних шляхів евакуації.

1 ПОШУК ШЛЯХІВ ДЛЯ ОДНОГО АГЕНТА

Однією з основних областей штучного інтелекту, інформатики та робототехніки є пошук шляху одного агента, який зосереджується на тому, щоб перевести агента від початкової точки до мети, уникаючи перешкод. Він широко використовується в транспорті та логістиці, пересуванні персонажів відеоігор, автономній навігації транспортних засобів і географічних інформаційних системах. Середовище зазвичай зображується у вигляді сітки або графіка, де вузли представляють розташування, а ребра представляють зв’язки між сусідніми розташуваннями. Агентом може бути будь-яка автономна сутність, наприклад робот, транспортний засіб або віртуальний персонаж у відеогрі.

Численні алгоритми та методи були розроблені в результаті значних досягнень у галузі пошуку шляху одного агента з часом. Основу для ефективного пошуку шляхів було закладено традиційними алгоритмами, такими як алгоритм Дейкстри та алгоритм A*, а включення евристик ще більше підвищило продуктивність і оптимальність цих алгоритмів.

Тим більше, що останнім часом дослідження почали досліджувати труднощі пошуку шляху, де умови змінюються в довгостроковій перспективі, а перешкоди можуть бути різними. Одним із яскравих прикладів підходу, який усуває труднощі навігації в таких середовищах, є створення алгоритму Space-Time A*.

Основу для більш складних багатоагентних завдань пошуку шляху та спільного планування було закладено шляхом вивчення визначення шляху одного агента, що забезпечило значний прогрес у розумінні та вирішенні труднощів навігації та планування в різних середовищах.

1.1. Задача пошуку шляху

Проблема пошуку шляху одним агентом може бути формально виражена математично в її контексті. Граф $G = (V, E)$ визначається як такий, що має набір вершин (вузлів) V і набір ребер E, що з'єднують ці вершини. Мета полягає в тому, щоб визначити найбільш ефективний маршрут для проходження агента від початкову вершину s з множини вершин V, до кінцевої вершини $g \in V$, обходячи будь-які перешкоди.

Граф G можна виразити двома способами, а саме у вигляді матриці суміжності A або списку суміжності L. Значення в матриці або списку відповідають витратам, понесеним під час переміщення між суміжними вершинами. Функція $cost$, яка визначена на множині ребер E, присвоює кожному ребру невід'ємні дійсні значення, вказуючи величину ресурсів, необхідних для переходу від однієї вершини до іншої.

Мета полягає в тому, щоб визначити шлях $P = (v_0, v_1, \ldots, v_n)$, де $v_0 = s$, $v_n = g$, а загальна вартість обходу суміжних вершин у P мінімізована:

$$\min_{v_i \in P} \sum_{i=1}^{n} cost(v_{i-1}, v_i) \ , \quad (v_{i-1}, v_i) \in E \quad \forall\, i \in \{1, \ldots, n\}$$

Цей математичний вираз містить у собі фундаментальну концепцію пошуку шляху для одного агента, мета якої полягає в тому, щоб мінімізувати загальну вартість проходження від початкової точки до пункту призначення, дотримуючись при цьому обмежень середовища, таких як перешкоди та допустимі рухи. Було розроблено декілька алгоритмів і методологій для ефективного вирішення цієї проблеми, враховуючи такі аспекти, як конфігурація графіка, характеристики функції вартості та наявність евристик для полегшення дослідження.

1.2. Алгоритм Дейкстри

У 1956 році Едсгер Дейкстра створив відомий алгоритм Дейкстри для пошуку графів. У зваженому графі з невід'ємними вагами ребер його метою є визначення найкоротшого шляху між початковим і цільовим вузлом.

Підтримка набору невідвіданих вузлів і попередньої відстані між кожним вузлом і початковим вузлом є основою алгоритму Дейкстри. Він встановлює відстань початкового вузла на нуль, а відстані всіх інших вузлів на нескінченність.

Потім обчислення знову і знову вибирає невідвідану вершину з найменшою відстанню, позначає її як відвідувану і оновлює відстані до його суміжних вершин. Якщо відстань від вибраного вузла до сусіда менша, ніж поточна попередня відстань, відстань для сусіда оновлюється новим значенням. Цикл триває, доки не буде відвідано цільову вершину або не буде відвідано всі вершини графу.

Нижче наведено псевдокод для алгоритму Дейкстри.

```
01 procedure RECONSTRUCT_PATH(cameFrom, node)
02     path := {node};
03     while node in cameFrom.keys) do
04         node := cameFrom(node);
05         add node to path;
06     return path;
07
08 function PROCESS_NODE(node)
09     for all node' in SUCCESSORS(node)do
10         tentativeScore := fScore(node) + cost(node,node')
11         if fScore(node') > tentativeScore then
12             cameFrom(node') := node;
13             fScore(node') := tentativeScore
16             if node' not in OPEN ∪ CLOSED then
17                 add node' to OPEN;
18         add node to CLOSED;
19
20 function Dijkstra (node_s, node_g)
21     OPEN := {node_s};
22     CLOSED := ∅;
23     cameFrom := an empty map;
24     fScore := map with default value of Infinity;
25     fScore[node_s] := 0;
28     while OPEN is not empty do
29         node := get node with the smallest fScore from OPEN
30         remove node from OPEN
```

```
31        if node = node_g then
32          return RECONSTRUCT_PATH(cameFrom, node)
33        PROCESS_NODE(node)
34      return NO_PATH
```

Алгоритм 1. Алгоритм Дейкстри

1.3. Алгоритм A*

Алгоритм A* (А Стар), створений Пітером Хартом, Нільсом Нільссоном і Бертрамом Рафаелем у 1968 році, є модифікацією алгоритму Дейкстри, що використовує спеціальний прийом для пріоритизації вершин з набору OPEN, який називається евристичною функцією. Цей трюк допомагає вгадати, яким шляхом йти далі, залежно від того, наскільки близько він до мети. Завдяки цьому алгоритм швидше знаходить найкращий шлях.

Алгоритм Дейкстри розглядає всі вершини з одинаковою пріоритетністю, тоді як алгоритм A* більш детально обирає наступну вершину для розгляду, намагаючись у першу чергу розглядати ті вершини, які найімовірніше ведуть до цільового вузла. Це пришвидшує пошук, оскільки A* не витрачає час на оглядання вершин, які знаходяться далеко від цілі. Проте ефективність роботи алгоритму A* залежить від того, наскільки правильно підібрана еврестична фукнція $h(x)$. Функція $h(x)$ повинна бути припустимою евристичної оцінкою, тобто не повинна переоцінювати відстані до цільової вершині. Завдяки добре підібраній евристиці, яка дозволяє розглядати вершини, що знаходяться ближче до цільової, раніше, алгоритм знаходить найкоротший шлях швидше в порівнянні зі звичайним алгоритмом Дейкстри. Використання хорошої евристики важливо для визначення ефективності A*. Значення $h(x)$ в ідеалі дорівнювало б точній вартості досягнення пункту призначення. Однак це неможливо, оскільки ми навіть не знаємо шляху. Однак ми можемо вибрати метод, який даватиме нам точне значення деякий час, наприклад, коли ви їдете по прямій без перешкод.

Існує кілька прийнятних евристик, які часто використовуються в алгоритмах пошуку шляху A*, кожна з яких має відмінні характеристики.

Манхеттенська відстань, яку ще називають міською квартальною відстанню, — це метод обчислення найпрямішого маршруту між двома точками на сітці шляхом проходження вздовж ліній сітки. Ця конкретна евристика особливо добре підходить для сценаріїв вирішення проблем, у яких допустимі лише горизонтальні та вертикальні рухи. Розрахунок передбачає підсумовування абсолютних різниць координат x і y.

$$h(n, n') = |n.x - n'.x| + |n.y - n'.y|$$

Евклідова відстань, відноситься до найкоротшої відстані між двома точками в евклідовому просторі, виміряної як пряма лінія. Вищезгадана евристика вважається придатною у випадках, коли дозволені діагональні рухи, і її можна обчислити за допомогою теореми Піфагора.

$$h(n, n') = \sqrt{(n.x - n'.x)^2 + (n.y - n'.y)^2}$$

Допустимість евклідової відстані залежить від пропорційності фактичної вартості проходження між вузлами та відповідної відстані по прямій. Тим не менш, у контексті визначення траєкторії на основі сітки, що передбачає рівномірні витрати, метрика евклідової відстані може демонструвати нижчий рівень точності порівняно з метрикою відстані Манхеттена.

Відстань Чебишева — це математична метрика, яка враховує найбільше значення абсолютних різниць між координатами x і y. Це рішення особливо підходить для задач, які включають сітку та дозволяють діагональні рухи, які за вартістю еквівалентні горизонтальним і вертикальним рухам.

$$h(x,y) = \max_{i=1,2,3,\ldots} |x_i - y_i|$$

Октильна відстань — це метрика, яка включає Манхеттенську та Чебишевську відстані та враховує витрати на діагональне переміщення, коли воно перевищує витрати на горизонтальне або вертикальне переміщення. Розрахунок може бути отриманий таким чином:

$$h(n) = D * (|n.x - n'.x| + |n.y - n'.y|)$$
$$+ (D2 - 2 * D) * \min(|n.x - n'.x|, |n.y - n'.y|)$$

де D — вартість горизонтального або вертикального переміщення, а D2 — вартість діагонального переміщення. Прийнятність метрики октильної відстані залежить від умови, що витрати, пов’язані з рухом по діагоналі, еквівалентні або перевищують витрати, пов’язані з горизонтальним або вертикальним рухом.

Підсумовуючи, різниця алгоритму A* від Дейкстри полягає у використанні додаткової функції для пріоритетизації вузлів у масиві OPEN.

```
01 procedure RECONSTRUCT_PATH(cameFrom, node)
02     path := {node};
03     while node in cameFrom.keys) do
04         node := cameFrom(node);
05         add node to path;
06     return path;
07
08 function PROCESS_NODE(node)
09     for all node' in SUCCESSORS(node) do
10         tentativeScore := gScore(node) + cost(node,node')
11         if gScore(node') > tentativeScore then
12             cameFrom(node') := node;
13             gScore(node') := tentativeScore
14             fScore(node') := gScore(node') + HEURISTIC(node');
15             RELAX(node, node');
16             if node' not in OPEN ∪ CLOSED then
17                 add node' to OPEN;
18         add node to CLOSED;
19
20 function AStar (node_s, node_g)
21     OPEN := {node_s};
22     CLOSED := ∅;
23     cameFrom := an empty map;
24     gScore := map with default value of Infinity;
25     gScore[node_s] := 0;
26     fScore := map with default value of Infinity;
27     fScore[node_s] := HEURISTIC(s_start);
28     while OPEN is not empty do
```

```
29          node := get node with the smallest fScore from OPEN
30        remove node from OPEN
31        if node = node_g then
32          return RECONSTRUCT_PATH(cameFrom, node)
33        PROCESS_NODE(node)
34      return NO_PATH
```

*Алгоритм 2. Алгоритм A**

2 ПОШУК ШЛЯХУ В ДИНАМІЧНОМУ ГРАФІ

Динамічний граф характеризується зміною в часі його сутностей графа, таких як вершини, ребра та ваги. Однією з поширених варіацій, які відбуваються з часом, є модифікація ваг ребер графа. Ці зміни також можуть служити як представлення встановлення або припинення крайових з'єднань за умови, що значення ваги дуги набуває нескінченного значення. Різні задачі комбінаторної оптимізації, які стосуються графів, можуть бути сформульовані в контексті динамічних графів. У багатьох випадках ця характеристика фундаментально змінює визначення проблеми, що вимагає різних підходів у статичному та динамічному сценаріях. Незважаючи на те, що це добре досліджена проблема комбінаторної оптимізації в існуючій літературі, питання визначення найкоротшого шляху отримало відносно менше уваги щодо динамічних варіацій графіка.

Класифікація динамічних графів може визначатися різними типами операцій, які вони здатні підтримувати. Динамічні графи можна класифікувати як повністю або частково динамічні. До таких операцій входять:

- Вставка вершини $x,\ x \notin V$.
- Видалення вершини $x,\ x \in V$.
- Вставка ребра (x,y,w), де $x,y \in V,\ (x,y) \notin E,\ x \neq y$.
- Видалення ребра (x,y), де $x,y \in V,\ (x,y) \in E$.
- Збільшення ваги (x,y,w), де $x,y \in V,\ (x,y) \in E,\ w > W(x,y)$.
- Зменшення ваги (x,y,w), де $x,y \in V,\ (x,y) \in E,\ w < W(x,y)$.

Частково динамічні графи підлягають певним обмеженням, згідно з якими дозволено лише вставлення та зменшення ваги ребер (інкрементні операції) або видалення та збільшення ваги ребер (декрементні операції).

Щодо проблеми найкоротшого шляху, вставка вершини x у граф вважається неефективною, оскільки x наразі не має спільних меж із жодною вершиною. У контексті другої операції передбачається, що видалення вершини x тягне за собою одночасне видалення всіх ребер, які з'єднані з нею

в межах графа. Акт усунення ребра можна витлумачити як збільшення його ваги до нескінченності. Вплив вставки ребра залежить від відсутності будь-якого іншого ребра, що з'єднує відповідні вершини, або від ваги нового ребра, що є нижчим за вагу попереднього ребра.

Повторне обчислення шляху з кожною зміною в графіку спричиняє великі витрати та неефективність. Було розроблено нові алгоритми для ефективного визначення найкоротшого шляху для динамічних графіків шляхом використання попередньо отриманої інформації для прискорення поточного процесу пошуку. Вищезазначені алгоритми включають, серед інших, D*, D* Lite, LPA*, HPA*, Adaptive A*.

2.1. Алгоритм Tree-Adaptive A*

Tree-AA* — алгоритм інкрементного евристичного пошуку найкоротшого шляху. Алгоритм Tree-AA* є розширенням алгоритму Adaptive A* (AA*), який застосовується для планування шляху. Він узагальнює адаптивний алгоритм Path-AA*, використовуючи суфікс шляху мінімальної вартості поточного пошуку A*, щоб дозволити наступному пошуку A* завершити процес пошуку раніше. Алгоритм Tree-AA* розширює алгоритм Adaptive A* повторно використовуючи суфікси шляхів мінімальної вартості поточного та всіх попередніх пошуків A*.

Алгоритм Adaptive A* працює за «принципом оновлення» в рамках ієрархічного шляху A* (HPA*). Якщо значення евристики h кожної вершини, розглянутої пошуком A* з послідовним h-значенням, встановлюється f-значення цільової вершини мінус g-значення вершини, тоді отримані значення h знову збігаються і слабо домінують над вихідними значеннями h. Тобто нова евристика $h'(s) = f(s) - h(s)$ є негіршою за $h(s)$ при пошуку шляху до тієї самої цільової вершини. Таким чином, пошук A* з отриманими оновленими значеннями h проглядає не більше вершин, ніж пошук A* з вихідними значеннями h. Цільова вершина повинна залишатися незмінною від пошуку A* до пошуку A*, але стартовий стан може змінюватися, а деякі

витрати руху можуть збільшуватися, але не зменшуватися. Таким чином, Adaptive A* може бути використаний для планування шляху з припущенням про вільний простір, що зазвичай робить пошук A* більш цілеспрямованим і, таким чином, пришвидшує їх.

Алгоритм Path-AA* базується на «принципі завершення» та розширенні «стратегії кешування шляху». Зокрема, якщо алгоритм має відомості про найкоротший шлях від даної вершини до цільової вершини (тобто використовує багаторазове дерево шляхів), і всі вершини шляху мають евристичні значення h, які еквівалентні їхнім цільовим витратам, тоді стандартний A* пошук може завершитися, коли він знаходиться на межі перевірки вершини в багаторазовому дереві шляхів повторного використання, включаючи цільову вершину. Отже, алгоритм Path-AA* має потенціал завершити пошук раніше, ніж стандартний пошук A*, який зупиняється лише тоді, коли цільова вершина знаходиться на межі обстеження. Мінімальна відстань між поточною вершиною агента та вершиною, яка є членом дерева повторного використання, у поєднанні зі шляхом від вершини дерева повторного використання до цільової вершини вздовж дерева повторного використання, становить найбільш прямий маршрут від теперішнього часу агента розташування до цільової вершини.

Алгоритм Tree-AA* є розширенням алгоритму Path-AA*, який використовує суфікси шляхів з мінімальною вартістю, отриманих із поточного та всіх попередніх пошуків A*. Такий підхід полегшує створення багаторазової деревовидної структури. Структура зберігає шляхи з найнижчою вартістю від різних вершин до певної цілі, організовані у структурі дерева з цільовою вершиною, яка служить коренем. Якщо h-значення всіх станів у дереві повторного використання дорівнюють їхнім кінцевим значенням і відомі шляхи з мінімальною вартістю від кількох вершин до цільової вершини, тоді прямий пошук A* може бути припинено, коли алгоритм знаходиться на грані розширення стану в дереві повторного використання.

Tree-AA* повинен підтримувати дві операції, а саме додавання шляху до дерева багаторазового використання та видалення шляхів із дерева багаторазового використання:

- **Додавання шляху до дерева багаторазового використання**: Якщо пошук A* завершується, коли він збирається розглянути вершину у дереві багаторазового використання, включаючи цільову вершину, тоді Tree-AA* додає шлях-гілку від поточної вершини агента до вершини в дереві багаторазового використання до дерева багаторазового використання. Це робиться тому, що найкоротший шлях від поточного стану агента до вершини у дереві багаторазового використання та найкоротший шлях від вершини у дереві багаторазового використання до цільової вершини вздовж гілки дерева багаторазового використання формують найкоротший шлях від поточної вершини агента до цільової, а $h-$значення усіх вершин на шляху дорівнюють своїм кінцевим значенням (оскільки Adaptive A* оновлює значення h таким чином).
- **Видалення шляху із дерева багаторазового використання**: Коли вага ребра у дереві багаторазового використання зростає, тоді Tree-AA* використовує найбільший префікс дерева багаторазового використання, який не містить ребр зі збільшеними витратами. Під префіксом дерева ми маємо на увазі верхню частину дерева, що включає його корінь. Це працює тому, що всі гілки результуючого дерева є шляхами з мінімальними витратами від деякої вершини до цілі та h-значенням усіх вершин в отриманому дереві все ще рівні цільовим витратам. Коли вартість ребра від вершини s до вершини s' зростає, тоді Tree-AA* знаходить найбільший префікс дерева багаторазового використання, видаляючи з нього як ребро (s,s') , так і піддерево з коренем у вершині s.

Алгоритм Tree-AA* ефективно виконує вищезгадані операції шляхом підтримки двох змінних для кожної вершини та трьох змінних для кожного шляху $x = [v_0,...,v_n]$ у багаторазовому дереві. Тут v_0 позначає початкову вершину шляху, тоді як v_n представляє кінцеву вершину, яку пошук A*

розширив би після завершення. Кожен окремий маршрут у багаторазовому дереві виділяється окремим цілим значенням, яке відповідає номеру пошуку A*, що вказує точку, в якій він був включений у багаторазове дерево, з нумерацією, починаючи з одиниці. Кожен маршрут у дереві повторного використання представляє префікс шляху з найменшою вартістю від даного стану до цільового стану. Вершини в дереві мінімальних шляхів виявляють властивість, згідно з якою їхні значення h еквівалентні їхнім кінцевим значенням, що призводить до строго монотонного зменшення вздовж шляху.

Далі в *алгоритмі 3* зображено псевдокод до алгоритму TreeAA*.

Алгоритм використовуються наступні змінні:

a) Id(s) – номер шляху в дереві мінімальних шляхів, до якого належить вершина s. Початкове значення для кожної вершини рівне нулю.
b) Reusabletree(s) – наступна вершина шляху у дереві для вершини s, якщо s не є кінцевою вершиною.
c) $H_{max}(x)$ – найбільше h-значення серед вершин $v_1,v_2,\ldots,v_k$, які належать шляху під номером x у дереві. $H_{max}(x) = h(v_1),\ H_{max}(0) = -1$.
d) $H_{min}(x)$ – найменше значення серед вершин $v_1,v_2,\ldots,v_k$, які належать шляху під номером x у дереві. $H_{min}(x) = h(v_k)$.
e) Paths(x) – набір усіх шляхів у дереві, приєднані до деяких вершин $v_1,v_2,\ldots,v_k$ шляху x.

```
01 function COMPUTE_PATH()
02   while (OPEN ≠ ∅) do
03     remove state s with the smallest
04         g(s) + h(s) value from OPEN;
05     if s = s_goal OR h(s) <= H_max(Id(s)) then
06       /* s in reusable tree */
07       for all s' ∈ CLOSED do
08         h(s') := g(s) + h(s) - g(s');
09       ADD_PATH(s);
10       return true;
11     insert s into CLOSED;
12     for all s' ∈ Succ(s) do
13       INITIALIZE(s');
14       if g(s') > g(s) + c(s, s') then
15         g(s') := g(s) + c(s, s');
16         Searchtree(s') := s;
17         if s' in OPEN then
18           remove s' from OPEN;
19           insert s' into OPEN with
20               value g(s') + h(s');
21   return false;
22
23 function MAIN
24   Counter := 1
25   H_max(0) := -1
26   for all s in S
27     Generated(s) := Id(s) := 0;
28     Reusabletree := ∅;
29   while s_start ≠ s_goal do
30     INITIALIZE(s_start);
31     g(s_start) := 0;
32     OPEN := ∅;
33     CLOSED := ∅;
34     insert s_start into OPEN
35         with value g(s_start) + h(s_start);
36     if (COMPUTE_PATH() = false) then
37       return false;
38     while (h(s_start) < H_max (ids_start))) do
39       s_start := Reusabletree(s_start);
40       for all increased costs c(s, s') do
41         if (Reusabletree(s) = s') then
42           RemovePath(s);
43     Counter := Cou14nter + 1;
44   return true;
45 function INITIALIZE(s)
46   if Generated(s) = 0 then
47     g(s) := inf;
48     h(s) := H(s);
49   else if (Generated(s) ≠ Counter) then
50     g(s) := inf
51   Generated(s) := Counter;
52
53 function ADD_PATH(s)
54   if s ≠ s_goal then
55     insert Counter into Path(Id(s))
56   H_min (Counter) := h(s);
57   H_max (Counter) := h(s_start);
58   Paths(Counter) := ∅;
59   while s ≠ s_startl do
60     s_aux := s;
61     s := Searchtree(s);
62     Id(s) := Counter;
63     Reusabletree(s) := s_aux;
64
65 function REMOVE_PATH(s)
66   x := Id(s);
67   if H_max (x) > h(Reusabletree(s)) then
68     H_max (x) := h(Reusabletree(s));
69   QUEUE := ∅;
70   for all x' ∈ Paths(x) do
71     if H_max(x) < H_min(x') then
72       add x' to the end of QUEUE;
73       remove x' from Paths(x);
74   while QUEUE ≠ ∅ do
75     remove x from the head of QUEUE;
76     if (H_max(x) > H_min(x)) then
77       H_max(x) := H_min(x);
78       for all x' ∈ Paths(x)
79         add x' to the end of QUEUE;
80         remove x' from Paths(x);
```

*Алгоритм 3. Алгоритм Tree-AA**

На *рис.1* зображено процес алгоритму Tree-AA* на якому видно супроводження агента та пошук наступних оптимальних шляхів якщо це необхідно. Починаючи з другого кроку алгоритм завершує роботу розглянувши вершину, що належить дереву найкоротших шляхів, проте не є цільовою вершиною.

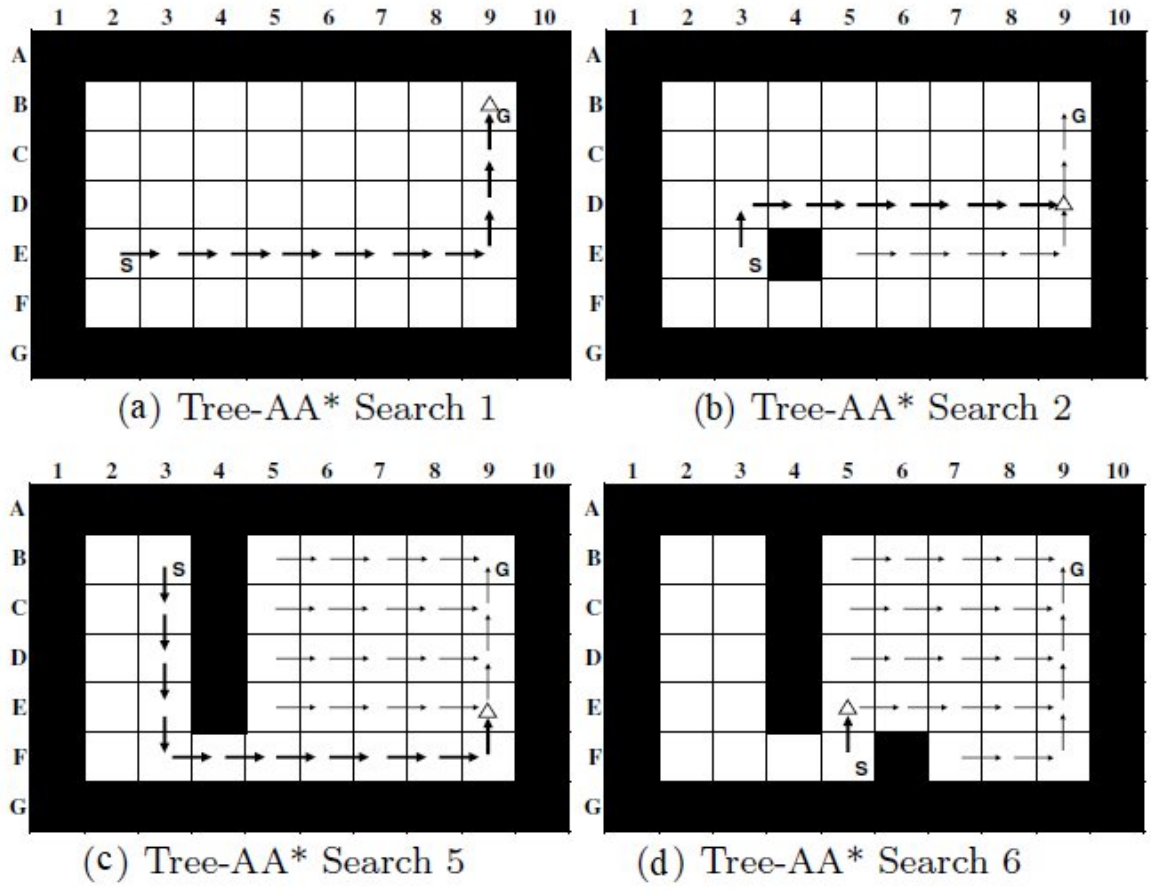


*рис.1. Побудова багаторазового дерева шляхів для повторного використання в алгоритмі Tree-AA**

3 ЗАДАЧА БАГАТО АГЕНТНОГО ПОШУКУ ШЛЯХІВ

Задача Мульти-Агентного Пошуку Шляхів (Multi Agent Path Finding, MAPF) є важливим розділом планування у мультиагентних системах, завдання якої полягає у знаходженні шляхів для кожного агента таким чином, щоб вони не конфліктували між собою. Задача MAPF є важливим та складним розширенням задачі пошуку шляху для одного агента. Проблема MAPF знаходить застосування в багатьох завданнях реального світу, таких як робототехніка, автономні транспортні засоби та управління складами, де кільком агентам потрібно переміщатися в спільному середовищі та досягати відповідних цільових місць, не стикаючись один з одним.

Проблема MAPF передбачає пошук набору неконфліктних шляхів для кількох агентів, враховуючи обмеження, накладені наявністю інших агентів у середовищі. На відміну від пошуку оптимального шляху для одного агента, проблема MAPF вимагає координації рухів кількох агентів, гарантуючи, що вони уникають зіткнень і “тупикових” ситуацій, водночас мінімізуючи загальну вартість знайдених неконфліктних шляхів.

Дослідження MAPF привернуло значну увагу в останні роки, оскільки воно представляє фундаментальну проблему в дослідженнях багатороботних і багатоагентних систем. Було запропоновано різні підходи та алгоритми для вирішення проблеми MAPF, кожен зі своїми перевагами, обмеженнями та припущеннями.

Математично класична задача багатоагентного пошуку шляхів описується трійкою $< G,s,g >$, де $G = (V,E)$ - ненаправлений граф, s:$[1,..., k] \rightarrow V$ - вершини графа, що позначають початкове розташування агентів, g: $[1,...,k] \rightarrow V$ - вершини, що позначають цільові вершини для кожного агента. Параметр часу дискретизований і в кожен момент часу кожен агент знаходиться в одній вершині графа і здатен здійснити одну дію. Позначимо цю дію функцією a:$V \rightarrow V$, що відображає перехід агента з однієї вершини на іншу, або очікування агента в цій вершині. Для агента і і послідовності дій

$\pi = (a_1, \ldots a_n)$, $\pi_i[x]$ позначатиме розташування агента в момент часу x. Формально $\pi_i[x] = a_x\ (a_{x-1}(\cdots a_1\ (s(i))))$. Якщо послідовність $\pi = (a_1, \ldots, a_n)$ для агента i починається з вершини $s(i)$ і закінчується вершиною $g(i)$, тобто $\pi_i\ [|\pi|] = g(i)$, то ця послідовність є планом для агента i. Розв'язком називається набір k планів для кожного агента.

3.1. Типи конфліктів MAPF

У контексті задачі мультиагентного пошуку шляхів коли два або більше агентів перешкоджають шляху один одного, або претенують на одну і ту ж позицію у графі, виникають конфлікти. Конфлікти повинні бути вирішені, щоб забезпечити безпечну та ефективну навігацію для всіх агентів. Існують такі основні типи конфліктів (*рис. 2*) для планів π_i та π_j:

- **Конфлікт вершин**: виникає коли в планах π_i та π_j обох агентів заплановано зайняти одну і ту ж вершину в один і той же час $\left(\pi_i[x] = \pi_j[x]\right)$.
- **Конфлікт ребер**: виникає коли існує такий час x такий що обидва агенти планують скористатись одними і тим же ребром графа в час x $\left(\pi_i[x] = \pi_j[x]\ \&\ \pi_i[x+1] = \pi_j[x+1]\right)$.
- **Конфлікт переслідування**: виникає коли хтось з агентів на наступний часовий крок планує зайняти вершину, яка уже зайнята іншим агентом $\left(\pi_i[x+1] = \pi_j[x]\right)$.
- **Конфлікт кола**: виникає між набором планів $\pi_i,\ \pi_{i+1}, \ldots,\ \pi_j$ коли хтось з агентів планує зайняти вершину, що уже зайнята іншим агентом в попередній часовий крок, формуючи коло, що обертається $(\pi_i[x+1] = \pi_{i+1}[x]\ \&\ \pi_{i+1}[x+1] = \pi_{i+2}[x]\ \&\ \ldots\ \&\ \pi_j[x+1] = \pi_i[x])$.
- **Конфлікт обміну**: виникає коли два агенти планують в один момнте часу обмінятись позиціями $\left(\pi_i[x] = \pi_j[x+1]\ \&\ \pi_i[x+1] = \pi_j[x]\right)$.

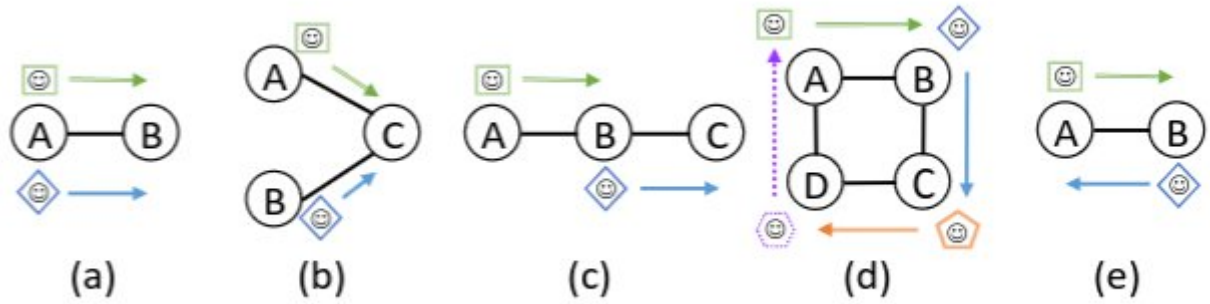


рис. 2. Типи конфліктів у задачі MAPF

Для різних постановок задач багато-агентного пошуку шляхів можуть бути прийнятні різні набори допустимих конфліктів. Для знаходження розв'язку контретної задачі необхідно спочатку визначити які типи конфліктів можливі у контексті задачі.

3.2. Оцінка рішення задачі MAPF

Очевидно що для однієї задачі MAPF може існувати багато різних розв'язків. Якісь з них є кращі за інші. Тому для оцінки цих розв'язків використовується деяка *цільова функція* $\boldsymbol{C}: \pi = \{\pi_1, \pi_2, \dots, \pi_k\} \rightarrow R$. Дві найбільш поширені цільові функції, що використовуються в класичній задачі MAPF є:

- **Sum of Cost** - сумма часових кроків для кожного агента для досягнення цільової вершини. Мінімізуючи суму вартостей шляхів, заощаджуються ресурси системи, наприклад, як енергія, час або паливо, тобто оптимізовується загальна ефективність системи.

$$\mathrm{C}(\pi) = \sum_{1 \le i \le k} |\pi_i|$$

- **Makespan** - загальне число часових кроків, необхідне щоб усі агенти досягнули своїх цільових вершин. Мінімізація *makespan* корисно, коли основна мета полягає в тому, щоб завершити завдання якомога швидше, наприклад, у сценаріях екстреної евакуації або критичних за часом додатках.

$$C(\pi) = \max_{1 \le i \le k} (|\pi_i|)$$

Вирішуючи задачу MAPF намагаємось оптимізувати цільову функцію розв’язку задачі.

3.3. Алгоритми вирішення MAPF

У цьому розділі надається огляд існуючих алгоритмів Multi-Agent Path Finding (MAPF). Ці алгоритми можна розділити на чотири групи: оптимальні алгоритми, обмежено-субоптимальні алгоритми та необмежено-субоптимальні алгоритми.

Оптимальні алгоритми це категорія методів, які гарантують знаходження найкращого можливого рішення, мінімізуючи обрану цільову функцію. Важливими характеристиками для оптимальних алгоритмів вирішення задачі MAPF є гарантованість знаходження розв’язку, якщо він існує та гарантованість оптимальності знайденого розв’язку. Проте зазвичай алгоритми цієї групи потребують великих обчислювальних можливостей, тому є мало масштабованими.

M* (Wagner and Choset 2011) — алгоритм на основі A*, який динамічно змінює коефіцієнт розгалуження на основі конфліктів. Загалом вузли розширюються лише одним дочірнім елементом, де кожен агент робить свій оптимальний рух до цільової вершини. Коли виникають конфлікти між агентами, коефіцієнт розгалуження збільшується, щоб охопити всі внутрішні можливості.

Ще одним алгоритмом на основі A* є алгоритм Enhanced Partial Expansion A* (EPEA*) (Goldenberg 2014), який уникає генерації надлишкових вузлів і є одним найкращим алгоритмом на основі A* для MAPF в своєму класі. EPEA* використовує апріорні знання середовища, щоб уникнути створення надлишкових вузлів.

Серед запропонованих алгоритмів, що не базуються на A*, відносяться алгоритми Increasing Cost Tree Search (ICTS), який перетворює задачу MAPF

на набір простіших швидко-розв'язуваних задач, та алгоритм Conflict Based Search (CBS), деталі якого будуть розглянуті в наступному розділі.

Обмежені суб-оптимальні алгоритми: забезпечують баланс між обчислювальною ефективністю субоптимальних методів і якістю рішення оптимальних алгоритмів. Ці алгоритми гарантують, що знайдені ними рішення знаходяться в межах обмеженого коефіцієнта оптимальної вартості. Ця група алгоритм використовує параметр $w = 1 + \epsilon$ і знаходить розв'язок задачі, оцінка якого C яке гарантовано менше або дорівнює зваженій оцінці оптимального рішення, $C \leq w \cdot C^*$, де C^* — це вартість оптимального рішення.

До списку таких алгоритмів відносять алгоритм Weightet A* (WA*) або Focal Search, який для швидшого пошуку використовує модифіковану зважену версію евристичної функції. На основі застосованих модифікацій, можливо знаходити субоптимальні рішення різними похідними алгоритмами, такі як EPEA* чи M*.

Подібні прийоми знаходження субоптимальних рішень можна застосувати до алгоритму CBS. Для прикладу алгоритм EECBS (Explicit Estimation Conflict Based Search), який пізніше буде розглянуто біль детально, використовує адаптовані суб-оптимальні альтернативи алгоритмів A* для верхнього і нижнього рівнів алгоритму.

Необмежені суб-оптимальні алгоритми: націлені на швидкий пошук рішення, дозволяючи, щоб повернуте рішення було неоптимальним. Більшість субоптимальних вирішувачів MAPF є необмеженими, тобто вони не надають жодних гарантій щодо якості поверненого шляху.

Важливий набір субоптимальних алгоритмів MAPF заснований на пошуку. Серед них привертають увагу є алгоритм Cooperative A* (CA*) та його варіації HCA* і WHCA* (Silver 2005). Ці алгоритми послідовно розробляють шлях для кожного окремого агента і зберігаються в структуру даних резервування. Наступним агентам не дозволяється планувати шлях, який суперечить маршрутам у таблиці резервування.

Іншою групою суб-оптимальних алгоритмів є алгоритми на основі правил, які використовують попередньо визначені правила для координації рухів агентів і уникнення конфліктів. Одним з таких алгоритмів є алгоритм ORCA (Optimal Reciprocal Collision Avoidance), що обчислює зміни швидкості для кожного агента на основі геометричних міркувань, забезпечуючи навігацію без зіткнень. Ще один алгоритм з цієї групи є алгоритм Multi-Agent Path Planning (MAPP) використовує попередньо визначені правила, щоб визначити, який агент має пріоритет, коли кілька агентів змагаються за ту саму позицію, дозволяючи скоординовану навігацію без явного планування шляхів агентів.

Кожна категорія алгоритмів MAPF має власні переваги, обмеження та припущення, а вибір алгоритму залежить від конкретних вимог і обмежень програми. Наприклад, оптимальні алгоритми найкраще підходять для ситуацій, коли необхідне рішення найвищої якості, тоді як пріоритетні алгоритми та алгоритми на основі правил є більш доцільними, коли обчислювальні ресурси обмежені або продуктивність у реальному часі є пріоритетною.

3.4. Conflict Based Search (CBS)

Класичний Conflict Based Search (CBS) — це дворівневий алгоритм пошуку, розроблений для оптимального вирішення задачі багатоагентного пошуку шляхів. CBS зарекомендував себе як доволі зручний алгоритм, який, завдяки своїй модульності, легко масштабувати на задачі з великою кількістю агентів.

Для опису алгоритму визначимо наступні визначення:

- *Шлях* застосовується лише у контексті одного агента, термін що позначає набір шляхів для усіх k агентів називатимемо *розв'язком*.
- *Обмеження* це трійка (a_i, v, t), яка означає, що агенту a_i заборонено займати вершину v в момент часу t. Шлях для агента a_i називатимемо *узгодженим*, якщо він задовольняє усі обмеження для цього агента.

Відповідно розв'язок є *узгодженим* лише коли всі шляхи цього розв'язку є *узгодженими*.

- *Конфлікт* є четвірка (a_i, a_j, v, t), що позначає ситуацію, коли агенти a_i та a_j намагаються одночасно в момент часу t зайняти вершину v. Розв'язок називатимемо *допустимим*, якщо в ньому немає конфліктів.

Алгоритм CBS використовує два рівня. На верхньому рівні виявляються конфлікти та додаються обмеження, а на низькому відбувається пошук шляхів для окремих агентів, які враховують нові обмеження.

3.4.1. Верхній рівень CBS

Для пошуку розв'язку на верхньому рівні CBS використовує двійкове *дерево обмежень (constraint tree, CT)* кожна вершина N якого є трійкою $(constraints, solution, cost)$, тобто складається з набору обмежень, набору шляхів для агентів та оцінкою розв'язку.

Під час розширення вузла N дерева обмежень CBS перевіряє наявність конфліктів в розв'язку цього вузла $N.solutions$ між його шляхами. Якщо їх немає, то вузол N є цільовим вузлом CT, і CBS припиняє роботу. Після виявлення конфлікту (a_i, a_j, v, t) CBS вирішує його, генеруючи нові обмеження (a_i, v, t) та (a_j, v, t) для залучених в конфлікт агентів. Для кожного з цих агентів створюється новий вузол у дереві обмежень (CT). Кожен новий вузол успадковує обмеження від свого батьківського вузла та вводить додаткове обмеження. Після цього знаходяться нові шляхи для конфліктних агентів уже з урахуванням нових обмежень для них.

Псевдокод класичного CBS алгоритму описано в *Алгоритмі 4*.

Input: MAPF instance
01 *Root.constraints* = ∅
02 *Root.solution* = find individual paths by the low level()
03 *Root.cost* = SIC(Root.solution)
04 insert Root to OPEN
05 **while** OPEN not empty **do**

```
06      P ← best node from OPEN    // lowest solution cost
07      Validate the paths in P until a conflict occurs.
08      if P has no conflict then
09          return P.solution              // P is goal
10      C ← first conflict (a_i, a_j, v, t) in P
11      foreach agent a i in C do
12          A ← new node
13          A.constraints ← P.constraints + (a_i, v, t)
14          A.solution ← P.solution
15          Update A.solution by invoking low level(a_i)
16          A.cost = SIC(A.solution)
17          if A.cost < ∞                  // A solution was found then
18              Insert A to OPEN
```

Алгоритм 4. Пошук CBS

Зауважимо, що CBS гарантує повноту розв'язку, оскільки досліджує обидва способи вирішення кожного конфлікту, а також гарантує оптимальність, виконуючи пошук оптимальними алгоритмами на високому та низькому рівнях.

3.4.2. Нижній рівень CBS

На низькому рівні алгоритму CBS відбувається одноагентний пошук шляху для заданого агента a_i, і набору обмежень, що пов'язані з a_i. Він виконує пошук у оригінальному графі, щоб знайти оптимальний шлях для агента a_i повністю ігноруючи інших агентів, проте враховуючи всі накладені на нього обмеження. Для цього низькорівневий алгоритм повинен здійснювати пошук у просторі який має два виміри: просторовий вимір і часовий вимір. Будь-який одноагентний алгоритм пошуку шляху може бути використаний, щоб знайти шлях для агента a_i, одночасно перевіряючи, що обмеження задовольняються. В класичному CBS для пошуку на низькому рівні використовується адаптований під врахування обмежень алгоритм A* – Space Time A*. Кожного разу, коли генерується стан (v, t), де v є місцем, а t

— часовим кроком, і існує обмеження (a_i, v, t) у поточному вузлі *CT*, цей стан відкидається. Отже, на відміну від одноагентного пошуку шляху, алгоритм низького рівня CBS включає часовий вимір і динамічні «перешкоди», викликані обмеженнями.

3.4.3. Оптимізація CBS

Алгоритм пошуку на основі конфліктів (CBS) є ефективним і результативним способом вирішення проблем пошуку шляху кількох агентів (MAPF). Але все для задач з великою кількістю агентів він потребуватиме великих обчислювальних можливостей, адже з ростом кількості агентів k кількість вузлів дерева CT зростає експоненційно. Проте вчені впродовж майже десятиліття запропонували чимало технік для покращення швидкості пошуку алгоритму CBS.

Техніка пропуску конфлікту, відома як «обхід конфліктів» (*Conflict Bypassing*) (Boyarski et al., 2015a), передбачає модифікацію шляхів агентів, залучених у даний конфлікт, на відміну від поділу вузла CT. В процедурі розширення вузла N та створення його дочірніх вузлів, якщо вартість дочірнього вузла N', еквівалентна вартості його батьківського вузла N, і кількость конфліктів, присутніх у розв'язку N' менше, ніж N, тоді N' замінює N шляхом заміни своїх шляхів, і всі дочірні вузли CT, згенеровані з N, відкидаються. В іншому випадку, N ділиться таким же чином, як і раніше. Дослідження показали, що обхід конфліктів може призвести до скорочення часу пошуку допустимого розв'язку у CT і скорочення часу виконання для пошуку на основі конфліктів (CBS).

Пріоритезація конфліктів (Boyarski et al. 2015b) є технікою вибору конфліктів. Конфлікт вважається кардинальним, якщо у випадку, якщо при розширенні вузла N, витрати, пов'язані з обома отриманими дочірніми вузлами, перевищують вартість $N.cost$. Вузол N у дереві вважається напівкардинальним, якщо вартість одного з його дочірніх вузлів перевищує вартість N, тоді як вартість іншого дочірнього вузла еквівалентна вартості

$N.cost$. Вузол N вважається некардинальним тоді і тільки тоді, коли витрати обох дочірніх вузлів еквівалентні вартості $N.cost$. Визначення класу конфлікту передбачає побудову багатозначної діаграми прийняття рішень (Multi-valued Decision Diagram, MDD) для кожного агента (Sharon та ін. 2013). CBS може значно підвищити свою ефективність, спочатку вирішуючи кардинальні конфлікти, потім напівкардинальні конфлікти і, некардинальні конфлікти останіми, оскільки генерація вузлів дерева обмежень з більшими витратами спочатку зазвичай покращує нижню межу вартостей вузлів CT.

Симетричне обґрунтування (Symetry Reasoning) (Li та ін. 2019c, 2020a) — це техніка для уникнення повторного вирішення конфліктів між тією самою парою агентів через симетричні шляхи та конфлікти. Для кожного агента є кілька найкоротших шляхів, яких можна досягти, змінюючи послідовність окремих рухів. Існування конфлікту виникає між найкоротшим шляхом одного агента та іншим агентом. Щоб прийти до рішення, CBS має спробувати різні перестановки цих маршрутів, зрештою визначивши, що певний агент повинен зробити паузу на один часовий крок. Швидкість зростання CT експоненціальна пропорційно розширенню області, яка представляє простір, де можуть виникнути потенційні конфлікти. Використання прийому *обґрунтування симетрії* дає змогу ефективно ідентифікувати кожну симетрію та її вирішення за допомогою окремої дії розщеплення, яка включає спеціальні обмеження. Цей підхід призводить до створення менших дерев обмежень і скорочення загального часу виконання пошуку CBS.

Евристика Weighted Dependency Graph (WDG) (Li et al. 2019a) є допустимою евристикою для високорівневого пошуку CBS. Процес передбачає побудову зваженого графа залежностей для кожного вузла N. Вершини графа представляють агентів, тоді як його ребра представляють залежність між двома відповідними агентами. Ця залежність визначається шляхом порівняння мінімальної суми вартості їх безконфліктних шляхів, які задовольняють $N.constraints$, яка обчислюється шляхом розв’язання задачі

MAPF із 2 агентами за допомогою CBS, із сумою вартості їх шляхів $N.paths$. Вони є найкоротшими шляхами, які задовольняють $N.constraints$, але можуть не бути безконфліктними. Ваги ребер графа представляють різницю між мінімальною сумою витрат безконфліктних шляхів, які задовольняють $N.constraints$, та сумою витрат їх шляхів у $N.paths$. Ця допустима евристика, яка використовується у високорівневому пошуку, визначається значенням мінімального покриття вершин графа, зваженого по ребрам. Незважаючи на обчислювальні витрати, пов'язані з побудовою зважених графів залежностей і визначенням їх мінімального покриття вершин, зважених по ребрам, інтеграція евристики WDG часто призводить до зменшення дерева обмежень і зменшує час пошуку CBS.

Алгоритм CBS з покращеннями працює доволі швидко та дає оптимальний розв'язок задачі MAPF. Проте незважаючи на усі перелічені покращення, йому доволі важко знаходити розв'язки для задач з величезною кількістю агентів. Та й для багатьох практичних задач не так важливо знайти оптимальне рішення, як просто знайти існуючий розв'язок, близький до оптимального. З цією метою були розроблені модифікації алгоритму CBS для пошуку субоптимальних рішень, які здатні працювати для випадків з набагато більшою кількістю агентів.

3.5. Explicit Estimation Conflict Based Search (EECBS)

Одним з субоптимальних варіантів алгоритму CBS є алгоритм *Explicit Estimation Conflict Based Search (EECBS)* (Li та ін. 2021). Він появився як розвиток іншого субоптимального варіанту *Enchanced CBS (ECBS)* (Barer та ін. 2014), в якому на верхньому на низькому рівнях використовується *Focal Search* (Pearl та Kim 1982) — це модифікація алгоритму A*, спрямована на прискорення процесу пошуку, зберігаючи при цьому якість рішення, близьку до оптимальної. Основною ідеєю Focal Search є використання двох списків: OPEN і FOCAL. Список OPEN в алгоритмі A* є стандартним списком, який сортується на основі функції допустимої вартості f. Вершина із набору

OPEN з мінімальним знаенням f позначимо як $bestf$. Нехай w представляє коефіцієнт субоптимальності, заданий користувачем. Множина FOCAL містить ті вершини N у OPEN, для яких $f(N) \leq w \cdot f(bestf)$, відсортовані відповідно до функції d, яка оцінює відстань, яку потрібно пройти, тобто кількість переходів від вузла N до цільового вузла. Focal Search завжди розширює вершину із мінімальним значенням d у FOCAL. Оскільки $f(bestf)$ є нижньою межею оптимальної вартості рішення C^*, фокусний пошук гарантує, що вартість повернутого рішення не перевищуватиме $w \cdot C^*$.

Проте використання Focal Search у алгоритмі CBS має свої обмеження. Перше полягає в тому, що алгоритм ECBS покладається виключно на показник відстані, коли обирає вузли CT для розширення. Зокрема, він вимагає, щоб вартість вибраного вузла N була в межах межі оптимальності $w \cdot lb(best_{lb})$, де $lb(N) = \sum_{i=1}^{m} f_{min}^{i}(N)$, а $best_{lb}$ – вершина з OPEN з мінімальним значенням lb, не беручи до уваги, що вартість рішення нижче вузла N може перевищувати $N.cost$ і, отже може бути більшою за межу субоптимальності.

Другий недолік полягає в тому, що нижня межа ECBS рідко вагомо підвищується. Таким чином, якщо оптимальна сума витрат не знаходиться в початковій межі субоптимальності, ECBS може мати труднощі з пошуком рішення протягом розумного часу.

Для вирішення цих недоліків на верхньому рівні алгоритму EECBS застосовується пошук *Explicit Estimation Search*.

3.5.1. Explicit Estimation Search

Explicit Estimation Search (EES) вводить третю функцію $\hat{f}$, яка оцінює вартість рішення нижче даного вузла. EES поєднує оцінки відстані, яку потрібно пройти, і ціну розв'язку, щоб передбачити розгляд вершин, які найшвидше приведуть до цільової вершини в межах заданого параметра субоптимальності. Алгоритм використовує три списки вузлів: CLEANUP, OPEN і FOCAL. CLEANUP — це звичайний відкритий список з алгоритму

A*, відсортований відповідно до допустимої функції вартості f. Нехай $best_f$ — вузол у CLEANUP із мінімальним значенням f, а w — заданий користувачем коефіцієнт субоптимальності. OPEN також є іншим звичайним відкритим списком A*, відсортованим відповідно до більш інформованої, але потенційно недопустимої функції ціни $\hat{f}$. Нехай $best_f$ буде вузлом у OPEN з мінімальним значенням $\hat{f}$. FOCAL містить вузли N у OPEN, для яких $\hat{f}(N) \leq w \cdot \hat{f}$, відсортовані відповідно до функції відстані до проходження d. $\hat{f}(best_{\hat{f}})$ – це оцінка ціни оптимального рішення, тому EES очікує, що розширення вузлів у FOCAL може призвести до рішень, які не більше ніж у w разів відрізняються від оптимальних. Нехай $best_d$ буде вузлом у FOCAL з мінімальним значенням d. EES вибирає наступну вершину для розгляду в такій послідовності:

- спочатку розглядається $best_d$, лише якщо $f(best_d) \leq w \cdot f\left(best_f\right)$;
- якщо $best_d$ не вибрано, наступною розглядаєтсья вершина $best_{\hat{f}}$, лише якщо $f\left(best_{\hat{f}}\right) \leq w \cdot f\left(best_f\right)$;
- якщо ні $best_d$, ні $best_{\hat{f}}$ не вибрано, обирається $best_f$, що може збільшити субоптимальність межі $w \cdot f\left(best_f\right)$, дозволяючи EES розглядати $best_d$ або $best_{\hat{f}}$ у наступній ітерації.

```
01 Generate root CT node R;
02 COMPUTE_WDG_HEURISTIC(R);
03 PUSH_NODE(R);
04 while OPEN is not empty do
05     P ← SELECTNODE();
06     if P.conflicts is empty then
07         return P.paths; // P is a goal node
08     if P is selected from CLEANUP and the WDG
         heuristic of P has not been computed yet then
09         COMPUTEWDGHEURISTIC(P);
10         PUSHNODE(P);
11         continue;
12     CONFLICTPRIORITIZATION(P.conflicts);
13     SYMMETRYREASONING(P.conflicts);
14     conflict ← CHOOSECONFLICT(P.conflicts);
```

15 *constraints* ← RESOLVECONFLICT(*conflict*);
16 *children* ← ∅;
17 **for** *constraint* in *constraints* **do**
18 Q ← GENERATECHILD(*P, constraint*);
19 **if** P is not selected from *CLEANUP* and
$\forall i \, |Q.paths[i]| \leq w \cdot f^i_{min}(P)$ and
$cost(Q) \leq w \cdot lb(best_{lb})$ and
$h_c(Q) < h_c(P)$
then // Bypassing
20 *P.paths* ← *Q.paths*;
21 *P.conf licts* ← *Q.conflicts*;
22 **Go to** Line 6;
23 Add Q to *children*;
24 **for** Q in *children* **do**
25 PUSHNODE(Q);
26 UPDATEONESTEPERRORS(*P, children*);
27 **return** “No solutions”;

Алгоритм 5. Пошук EECBS

EECBS для верхнього пошуку використовує алгоритм EES і Focal Search для нижнього рівня. Зауважимо, що покращення швидкості пошуку для алгоритму CBS теж можна застосувати до EECBS з нескладними модифікаціями. Псевдокод для високого рівня EECBS описано в *алгоритмі 5*.

4 МОДЕЛЮВАННЯ ПОВЕДІНКИ ПОШИРЕННЯ НЕБЕЗПЕКИ

Моделювання перетворень і рухів реальних або абстрактних частинок у дискретному часі та просторі тепер можливо в сучасних обчислювальних системах, що дозволяє моделювати складні фізичні та хімічні явища. Процеси реакції-дифузії є одним із прикладів цієї широкої категорії подій. Вони являють собою наочне уявлення про дифузію речовин, нелінійну або порогову природу хімічних, фазових і біологічних змін і навіть прямі заміни типу «було – стало». Використання звичайних математичних моделей, заснованих на диференціальних рівняннях, може бути складним, якщо не неможливим, коли маємо справу з такими явищами через нелінійність і непостійність компонента реакції. Дискретне моделювання, побудоване навколо концепції клітинного автомата (КА) фон Неймана як заміни, виникло в результаті цього пошуку альтернативного методу для моделювання процесів.

4.1. Клітинний автомат

Математично модель на основі клітинного автомату можна задати четвіркою

$$\aleph = \langle A, X, \Theta, \rho \rangle,$$

де X – набір клітин, представлений їх іменами; A - алфавіт станів комірки в X; Θ – набір операторів локальної комірки, також званий глобальним оператором; ρ – режим роботи моделі. Стани клітин можуть бути представлені числовими значеннями, булевими векторами або символами. Крім того, імена комірок можуть складатися з натуральних чисел або векторів, що позначають координати дискретних просторових точок. Позначення (a, x) використовується для представлення комірки $x \in X$, яка знаходиться в стані $a \in A$. Отже, добуток множини A та множини X є множиною, що охоплює всі можливі комірки в усіх можливих станах. Будь-яка його підмножина $\Omega = \{(a_1, x_1), \dots, (a_s, x_s)\}$, де $\{x_1, \dots, x_s\} = X$ називається

клітковим масивом КА-моделі. Масив станів комірок $(a_{1,}\dots,a_s)$ складають глобальний стан КА-моделі.

Локальний оператор визначається за допомогою функцій $\varphi_i : X \to X$, $i = 1, 2, \dots, q$, такі що $x \neq \varphi_1(x) \neq \varphi_2(x) \neq \dots \neq \varphi_q(x)$. Їх підмножини

$$N(x) = \{x,\varphi_{1(x)},\dots,\varphi_{n(x)}\}, \quad E(x) = \{\varphi_{n+1}(x),\dots,\varphi_q(x)\}, \ 0 \le n \le q,$$

$$S(x) = \{(u_{0,}x),\dots,(u_{n,}\varphi_n(x))\}, C(x) = \{(u_{n+1}, \varphi_{n+1}(x)),\dots,(u_{q,}\varphi_q(x))\}$$

Називаються сусідством, округом, локальною конфігурацією та контекстом клітки x.

Локальні оператори з множини Θ, призначаються коміркам клітинного автомату, таким чином щоб кожна комірка була пов'язана з одним або кількома операторами. Режим роботи КА-моделі можна визначити шляхом визначення розподілу ймовірностей на множині $\Theta(x)$ операторів, відображених у комірку x, яка керує порядком їх вибору під час процесу функціонування. Клітинний оператор x, який є довільним локальним оператором у $\Theta(x)$ представляється записом $\theta(x)$. Відповідно, застосувавши цей оператор до всіх комірок клітинного автомату, що перебувають в певній конфігурації $S(x)$ з контекстом $C(x)$, можемо отримати наступну конфігурацію

$$S'(x) = \theta(x)\,[S(x), C(x)] = \{(u'_{0,}x), (u'_{1,\varphi_1(x)}),\dots,(u'_{n,\varphi_n(x)})\},$$

Стани якої обчислюються $u_k = f_k(u_{0,}u_{1,}\dots,u_q)$, $k = 0,1,\dots,n$, використовуючи функції переходів $f_k : A_{q+1} \to A$.

КА-модель працює на дискретній шкалі часу, позначеній $t = 0, 1, \dots$, де кожен інтервал часу називається ітерацією. Робота моделі базується на початковому масиві комірок $\Omega(0)$. Масив клітинок Ω на ітерації $t+1$ дорівнює функції Θ, застосованій до Ω на ітерації t. Процес передбачає застосування локальних операторів $\theta(x) \in \Theta$ до всіх комірок $(u, x) \in \Omega(t)$ у заздалегідь визначеному порядку, що призводить до глобального переходу $\Omega(t) \to \Omega(t+1)$ від поточного масиву комірок до наступного. Акт

застосування функції $\theta(x)$ до комірки (u,x), яка належить набору $\Omega(t)$, передбачає заміну станів комірок, що належать набору $S(x)$ у $\Omega(t)$ на стани комірок, які мають однакові назви з множини $S'(x) = \theta(x)[S(x), C(x)]$. Зауважимо, що в результаті виконання оператора значення контексту $C(x)$ не змінюється, але в місце конфігурації $S(x)$ записується нова — $S'(x)$.

Відповідно до КА-моделі, глобальні переходи вважаються коректними, коли немає колізій під час обчислення $\Omega(t+1)$ за $\Omega(t)$. Зокрема, це означає, що не повинно бути спроб змінити стан однієї клітини більше одного разу в один і той же момент часу t.

Еволюція змодельованої моделі представлена у вигляді послідовності масивів комірок $\Omega(0), \Omega(1), ..., \Omega(t), ...$ через ітераційне застосування глобального оператора Θ до масивів $\Omega(t)$. У випадку, коли ітераційний процес досягає збіжності, тобто існує таке t, що $\Omega(t) = \Omega(t+1) = ...$, система досягає стану рівноваги. В іншому випадку клітинний автомат емулює процес, який демонструє коливальну або хаотичну поведінку.

В залежності від задання оператора переходів, КА класифікують на три рівня складності:

- *прості* КА-моделі, які використовують один локальний оператор, $|\Theta| = 1$, тобто здатні імітувати тільки одну елементарну дію;
- *складні* КА-моделі, у яких глобальний оператор працює з безліччю локальних операторів, $|\Theta| > 1$, і тому здатні імітувати складні процеси;
- *композиції глобальних операторів* Θ_i, такі, як послідовна та паралельна. Вони всі Θ_i працюють послідовно чи паралельно, оновлюючи запропоновані їм підмасиви $\Omega_i \subset \Omega$ і використовуючи як контекстів набори клітин із усього масиву Ω.

Ітерація глобального оператора $\Theta(\Omega(t))$ у всіх випадках розглядається як процедура, яка передбачає застосування всіх $\theta_i(x) \in \Theta(x)$ до всіх $(u,x) \in \Omega(t)$ і складається з $|X| \cdot |\Theta|$ кроків. Послідовність дій визначається режимом роботи ρ. Для складної моделі цей режим складається з двох складових ρ_x та

ρ_θ, визначеними як просторова компонента, яка керує послідовністю вибору комірки, і операторна компонента, яка керує послідовністю локального вибору оператора. У випадку композицій глобальних операторів, кожен оператор функціонує у своєму окремому режимі, і їхня взаємодія залежить від типу композиції.

4.2. Режими функціонування КА-моделей

В залежності від вибраного режиму роботи КА-моделі спостерігатимуться доволі різні еволюції КА при таких самих початкових конфігураціях. Просторова компонента може функціонувати в режимах:

- *синхронний* – оператор $\theta(x)$ застосовується до всіх клітин $(u, x) \in \Omega(t)$ в будь якому порядку, проте нові значення (u', x) спочатку зберігаються в додаткову пам'ять Ω', після чого заповнюється конфігурація $\Omega(t+1)$;
- *асинхронний* – оператор $\theta(x)$ застосовується до клітин $(u, x) \in \Omega(t)$ послідовно у випадковому або заданому порядку, причому нові значення станів клітин заміняють теперішні відразу.

Для складної КА-моделі для задання режиму роботи необхідно додатково визначити режим функціонування другої операторної компоненти. Є два різних варіанти задання оператора $\theta(x) \in \Theta(x)$ для застосування до обраної клітини $x \in X$: *детермінований* і *стохастичний*. Зауважимо, що при складній КА-моделі ітерація складається з $|X| \cdot |\Theta|$ виборів клітини і оператора на її застосування.

Композиція глобальних операторів передбачає об'єднання кількох глобальних операторів в уніфіковану структуру, де кожен складовий оператор може мати просту або складну структуру та працювати відповідно до свого попередньо встановленого режиму. Ця композиція може бути *послідовною*, коли ітерація складається з обчислень L проміжних глобальних переходів $\Omega(t+1) = \Theta L(\Theta L-1(\dots(\Theta 1(\Omega(t)))))$, або *паралельною*, коли множини клітин X і операторів Ω поділені рівномірно на L підмножин

$X = X^{(1)} \cup \ldots \cup X^{(L)}$ та $\Omega = \left\{\Omega^{(1)} \cup \ldots \cup \Omega^{(L)}\right\}$, при чому усі підмножини $\Omega(j) \in \Omega$ оновлюються одночасно відповідними глобальними операторами $\Theta(j)$.

4.3. Інваріанти КА-моделей

В своїй роботі О. Л. Бандман в пошуках методу переходу від фізико-хімічного опису моделюючого процесу до КА-відображення і його інтерпретації ввела поняття інваріанту КА-моделі. Інваріант, який є безрозмірним і не залежить від підходу математичного представлення, дозволяє встановити коефіц14ієнти масштабування між фактичними та модельними значеннями. Інваріантна модель пов'язує просторовий (розмір комірки в метрах) і часовий (тривалість ітерації в секундах) масштаби, оскільки КА моделюють просторову динаміку. Таким чином, ці шкали не можуть бути самостійно обрані взагалі. КА-моделі реакційно-дифузійних процесів включають дифузію та реакцію, кожна зі своїм інваріантом. Інваріанти складних процесів не завжди є функцією інваріантів їх складових, а функції масштабування не завжди можна виразити аналітично через інваріант. У цих складних обставинах лише обчислювальний експеримент і таблиці та криві, що їх з'єднують, можуть визначити інваріанти та масштаби.

Інваріанти дифузійних процесів. Математичні моделі дифузійних процесів зазвичай описують в неперервному вигляді за допомогою рівнянь Лапласа. Вони характеризуються безрозмірним коефіцієнтом $D = d \cdot {}^{\tau}/_{h^2}$, де d — коефіцієнт дифузії, виражений в $м^2$ за секунду; τ - крок за часом у секундах; h — крок по простору в метрах. Для цих моделей можна використовувати інваріант, що рівний відповідному параметру D, оскільки він не залежить від початкових умов.

З першого закону Фіка для дифузії в ізотропному середовищі, який говорить, що коефіцієнт дифузії - це маса речовини, яка рухається через одиницю площі за одну одиницю часу з градієнтом концентрації одиниці, легко зрозуміти, яке значення D є. У булевому масиві комірок градієнт

концентрації дорівнює 1 на межі між областю, де всі клітини мають стан $u = 1$, і областю, де всі клітини мають стан $u = 0$. Отже, кількість речовини (кількість частинок), яка розповсюджується на $\Delta t = 1$ дорівнює ймовірності того, що дві граничні комірки поряд одна з одною змінять стани.

Два найбільш відомими булевими КА, які здатні моделювати процес дифузії є:

- *Синхронний КА* $\aleph_\sigma = \langle A,X,\Theta \rangle, A = \{0,1\}, X = \{x{:}x = (i,j)\}$. Для його функціонування визначаються парна і непарна підмножини X і шаблон $T_{2\times2} = \{(i,j), (i,j+1), (i+1,j+1), (i+1,j)\}$, який визначає локальну конфігурацію і локальний оператор $\Theta(i,j)$, який є стохастичним

$$u'_k = \begin{cases} u_{(k+1)\ mod\ 4}, & if \quad rand < p, \\ u_{(k-1)\ mod\ 4}, & if \quad rand \geq (1-p), \end{cases}$$

де p – ймовірність застосування $\Theta(i,j)$. Ітерація спершу відбувається для множини X_0 а потім для X_1. В [x] показано, що

$$D_{2\sigma} = \begin{cases} 1 & if \quad p = 0.5 \\ p & if \quad p < 0.5 \end{cases}$$

- *Асинхронний КА* $\aleph_\alpha = \langle A,X,\Theta \rangle, A = \{0,1\}, X = \{x{:}x = (i,j)\}$. В цьому КА кожна випадково обрана клітина обмінюється станом з однією із сусідніх клітин з рівною ймовірністю. Для цього задається шаблон з локальною конфігурацією

$$S(x) = \{(u_0, x), (u_1, x + a_1), (u_2, x + a_2), (u_3, x + a_3), (u_4, x + a_4)\}$$

Очікується, що на одній ітерації частинка перейде з комірки $(1, x)$ в одну з суміжних комірок $(0, x + a_k)$ з імовірністю 0,5, причому

$$D_{2\alpha} = \begin{cases} 0.5 & if \quad p = 1 \\ 0.5 \cdot p & if \quad p < 1 \end{cases}$$

Визначення масштабів параметрів моделі КА, а саме довжини h сторони комірки в метрах і часу ітерації τ в секундах, полегшується знанням

інваріанта, оскільки можна вибрати одну зі шкал на основі заданих умов, тоді як іншу можна визначити за допомогою математичної залежності

$$D = \frac{d \cdot \tau}{h^2}$$

Інваріант реактивних процесів. У контексті КА-моделей хімічних реакцій інваріантом рекомендується вважати безрозмірний коефіцієнт $R = k \cdot \tau$. Тут k являє собою константу швидкості реакції за секунду часу, яка є добре встановленим параметром для більшості досліджених реакцій. Крім того, τ позначає тривалість ітерації клітинного автомата, виміряну в секундах. Слід зазначити, що інваріантність хімічної реакції моделі КА не залежить від величини клітини, таким чином, коефіцієнти масштабування h і τ виключають один одного. Отже, h можна визначити на основі умов задачі. Інваріант R фізично означає зміну концентрації $(\Delta C/C)$ реагенту протягом однієї ітерації τ. Параметр τ служить зв'язком між теоретичними значеннями моделі та відповідними фактичними значеннями. Якщо модель включає одиночну реакцію, доцільно вибрати τ як еквівалент тривалості зазначеної реакції. У випадках, коли змодельований процес охоплює численні реакції та потенційно додаткові дії, співвідношення швидкостей залежать від ймовірностей реалізації відповідних локальних операторів, які залежать від усіх інваріантів.

Інваріант процесу «поширення фронту». Явища, які належать до сімейства реакційно-дифузійних явищ, у яких реакція описується нелінійною функцією

$$F(u) = \alpha u(1-u), \qquad F(0) = F(1) = 0,\ \ 0 < \alpha \le 1,\ \ 0 \le u \le 1,$$

є процесами «дифузії зі зростанням кількості речовини». Для характеристики цих процесів використовуються диференціальні рівняння типу

$$u_t = D \cdot u_{xx} + F(u),$$

де D – оператор дифузії.

Для моделювання процесу поширення фронту можна використати асинхронний КА $\aleph_\alpha = \langle A, X, \Theta(X) \rangle$ з булевим алфавітом $A = \{0,1\}$, множиною

клітин $X = \{(i,j) : i,j = 0,\dots,N\}$ і глобальним оператором $\Theta(X) = \Phi_z(\theta_d(x),\theta_r(x))$ (Φ_z – один з способів композиції). Локальний оператор дифузії θ_d визначається на шаблоні

$$T(i,j) = \{(i,j),(i,j+1),(i+1,j),(i,j-1),(i-1,j)\},$$

і при його застосуванні на обрану клітку (u_0, x_k) її стан u_0 заміняється на стан клітини-сусіда u_l з однаковою ймовірністю:

$$(u_0 = u_l)\ \&\ (u_l = u_0), if\ \ (l-1)/4 < rand_1 < l/4\ \&\ rand_2 < p_d\ , l = \overline{1,4}.$$

Інваріантом процесу є *швидкість поширення фронту*. В [15] аналітичним шляхом доведено, що швидкість поширення фронту при t → ∞ залежить від коефіцієнтів дифузії (D) та швидкості реакції (α) наступним чином

$$V = 2\sqrt{D \cdot \alpha}.$$

Інваріант процесу обмеженої дифузією агрегації. Процес *агрегації обмеженої дифузією (diffusion-limited agregation, DLA)*, не може бути представлений у вигляді диференціального рівняння. Його перша модель була заснована на русі в дискретному просторі та взаємодії частинок.

Процесу відповідає еволюція асинхронного КА $\aleph = \langle A, X, \Theta(X)\rangle$, $A = \{0,1,b\}, X = \{(i,j)\}$. Локальний оператор $\Theta(i,j)$ визначається як композиція локального оператора наївної дифузії θ_d та оператора прилипання θ_r : $\Theta(i,j) = \Phi_z\left(\theta_d(i,j), \theta_r(i,j)\right)$. Таким чином, функція переходу тут визначається

$$u'_0 = \begin{cases} b, & if\ (u_0 = 1)\ and\ (u_l = b)\ and\ (rand < p_r) \\ u_0, & else \end{cases}$$

Зауважимо, що коефіцієнтом прилипання однозначно визначається фрактальна розмірність, яка є основною характеристикою реального явища DLA. Тому її можна брати за інваріант для КА-моделі процесу агрегації обмеженої дифузією.

5 БАГАТО-АГЕНТНИЙ ПОШУК ШЛЯХІВ ПІД ЧАС ЕВАКУАЦІЇ

Багато-агента евакуація в динамічному середовищі (Multi Agent Evacuation with Dynamic Obtancles, MAEDO), далі просто *евакуація* — це анонімна форма задачі багатоагентного пошуку шляхів (MAPF) у динамічно змінному графі. Процес евакуації відбувається в неорієнтованому графі, позначеному як $G = (V, E)$, де вершини класифікуються як безпечні або небезпечні. Зокрема, множина вершин V може бути виражена як об'єднання двох непересічних підмножин, а саме S і D, які відповідають безпечним і небезпечним групам вершинам відповідно. Анонімна форма MAPF передбачає що цільова вершина агента не є чітко визначеною вершиною графа, а може бути будь-якою вершиною з визначеної групи цільових вершин. Граф G виступає середовищем на якому відбувається поширення небезпеки.

Мета полягає в тому, щоб визначити набір шляхів для групи агентів $A = \{a_{1,}a_{2,}\ldots,a_k\}$, що дозволяє їм пройти до безпечних вершин S уникаючи небезпечні вершини D. Множина вершин D може змінюватись відповідно до оператора, який відображає процес поширення небезпеки. Оскільки граф являє собою регулярну сітку, для моделювання небезпеки зручно обрати КА-моделювання. Кожен агент з A, починає свій шлях з окремої вершини, тим самим гарантуючи, що в будь-який даний момент вершину займає не більше ніж один агент. Розв'язок задачі включає план $\pi = [c_{0,}c_{1,}\ldots,c_m]$, де кожен елемент плану $c_m(a) \in S \ \forall\, a \in A$, і $c_t : A \to V$.

Математично задачу багатоагентної евакуації в динамічному середовищі можна задати п'ятіркою $E = [G = (V,E), \aleph, A, c_0, D, S]$ де G представляє середовище, $\aleph$ – КА-модель поширення небезпеки в середовищі G, A — набір агентів, $c_0 : A \to V$— початкова конфігурація агентів, D і S такі, що $D \subseteq V$, $S \subseteq V$, $V = D \cup S$ вважаючи, що $D \cap S \neq \emptyset$ і $|S| \geq k$ представляють набір під загрозою та набір безпечних вершин відповідно.

Як правило, намагаються скоротити тривалість евакуації, яка визначається як загальна кількість часу, необхідної для досягнення останнім агентом цільової вершини від початку евакуації. Тому, за цільову функцію для задачі MAEDO вибрано функцію *«makespan»*.

Для вирішення поставленої задачі пропонується використання ієрархічної моделі поведінки агентів, у якій зазвичай більш обізнані агенти дотримуються плану евакуації, розробленого централізованим алгоритмом. Для цього агенти діляться на два види: *лідери* (далі *leaders*) та *послідовники* (далі *followers*). Мотивація такого рішення – евакуаційна ситуація в середовищі коли не усі агенти здатні самостійно знаходити шляхи до виходів або є обмеженими у можливостях. Наприклад, агенти можуть представляти викладачів і учнів школи. Лідери шукають вихід із будівлі, а послідовники утворюють зграї навколо лідерів, слідують за ними та евакуюються з їх допомогою.

5.1. Агенти – Лідери

В цій моделі ми припускаємо, що лідери мають повні знання та можуть бути керовані централізовано в принципі. Можна припустити, що вони можуть бути оснащені електронним пристроєм, який надає їм інформацію щодо плану та поточних обставин.

Мета лідера полягає в тому, щоб визначити найбільш ефективний шлях для виходу з небезпечного регіону, зберігаючи певний відрив від інших лідерів. Кожен лідер має групу послідовників, які йдуть за ним до найближчого виходу. Після кожної фази лідери повинні реагувати на зміни в середовищі, які можуть включати динаміку перешкод, небезпек або рух інших лідерів. У випадку, якщо послідовники заважають лідеру просуватися вперед, вони мають можливість змістити їх із поточних позицій, щоб продовжити рух по своєму маршруту. В загальному лідера можна описати як індивіда, який має заздалегідь визначену мету та всебічне розуміння свого оточення. Для знаходження маршрутів лідерів використовується MAPF

алгоритм. Це може бути алгоритм пошуку на основі конфліктів CBS або, за необхідності, його субоптимальні варіанти, такі як EECBS. Оскільки середовище має можливість до динамічних змін, CBS пошук необхідно адаптувати до таких умов.

Для вирішення такої задачі можна використовувати адаптовані алгоритми пошуку шляхів в динамічних графах, які ми розглядали раніше, на нижньому рівні пошуку на основі конфліктів. Алгоритм TreeAA* чудово підходить для таких задачі, проте потребує деяких уточнень. Так, при звичній ситуації зміну станів графу середовища, алгоритм відкидає частину багаторазового дерева повторного використання і відбувається пошук нового шляху до цільової вершини. Проте адаптований алгоритм повинен виконувати пошук з урахуванням обмежень, накладеними CBS вузлом. Механізм дії адаптації наступний. При розгляді вершини з багаторазового дерева, алгоритм не завершує пошук, але проходить через вершини дерева шляхів з перевіркою на накладені обмеження. Якщо протягом частини шляху дерева шляхів, не знайдено відповідного обмеження, то шлях знайдено. Інакше алгоритм додає усі вершини-сусідів вершин з багаторазового дерева шляхів, пройдені до першого знайденого обмеження, до множини OPEN і алгоритм продовжує пошук у звичному йому режимі.

При аналізі конфлікту двох лідерів недостатньо розглядати лише конкретні координати, але необхідно брати до уваги територію навколо обох осіб, яка визначається їхніми відповідними параметрами конфліктної відстані. Цей параметр відноситься до мінімальної відстані, яку агенти повинні підтримувати між собою.

Для сімейства алгоритмів CBS задачі MAEDO термін «конфлікт» визначається як п’ятірка $\langle a_1, a_2, N_1, N_2, t \rangle$, де $a_{1,}\ a_2$ представляють агентів у конфлікті, N_1, N_2 – конфліктні позиції агентів, t – час виникнення конфлікту. Конфлікт стосується двох різних позицій, кожна з яких дотримується відповідним агентом. Цей вибір був зроблений на основі можливості виникнення конфліктів між агентами, навіть за відсутності фізичної

близькості. Близькість одного агента до іншого можна встановити за їхніми відповідними координатами під час входу в певну територію. Достатньо, щоб один агент увійшов у зону, що знаходиться в безпосередній близькості від іншого агента, при цьому кожен агент матиме окрему координату в тій самій зоні. Отже, кожен лідер має чітке обмеження щодо своїх координат. Псевдокод пошуку конфліктів, що виконується в межах функції перевірки допустимості розв'язку, описано в *алгоритмі 6*.

Input: Node N
01 conflicts = ∅;
02 **for** $i \in \{1,2,\ldots,len(N.solution)\}$ **do**
03 **for** $j \in \{i+1, i+2, \ldots, len(N.solution)\}$ **do**
04 steps ← min(len(solution[i],len(solution[j]));
05 **for** $t \in \{1,\ldots, steps\}$ **do**
06 geometry ← compute geometry(solution[i],solution[j], N.state);
07 **if** geometry.in_sight(ai, a j, t) **then**
08 conflicts ← conflicts ∪ {(i, j, solution[i][t], solution[j][t], t)};
09 **return** conflicts;

Алгоритм 6. Пошук конфліктів у CBS вузлі з MAEDO валідацією.

Функція перевірки передбачає порівняння позицій кожної пари лідерів на кожному кроці їх відповідного шляху. Коли два лідери знаходяться в безпосередній близькості один до одного без будь-яких перешкод, на певних етапах може виникнути конфлікт.

5.2. Агенти – Послідовники

У ієрархічній моделі агенти евакуації з нижчими рівнями інформації називаються *послідовниками* (далі *follower*) та підлягають локальному контролю за допомогою різноманітних підходів на основі правил.

Послідовник повинен мати можливість спостерігати за лідером у його безпосередній близькості, якщо жодна перешкода не заважає йому це зробити. У випадках, коли є кілька лідерів, послідовник може обрати свого, наприклад того, хто з них знаходиться ближче, або лідер може бути призначеним. У випадку, якщо лідер стає поза полем зору, послідовник

намагається переміститися до останнього місця, де лідер був видимим для послідовника.

В даній моделі послідовник поводиться як рефлекторний агент (Russell та Norvig, 2010) без додаткових когнітивних можливостей. Таке рішення було прийнято на основі принципу імітації процедур евакуації в “шкільному” середовищі, де від послідовників очікується виконання вказівок, виданих відповідальною особою - лідером. Таким чином, послідовник не намагається самостійно евакуюватися, скоріше він шукає лідера, який може направити його до виходу. Агент цього типу має обмежене сприйняття свого найближчого оточення в межах свого зорового сприйняття та намагається переміщується до незайнятої клітини, яка знаходиться ближче до лідера. У випадку, якщо агент не може знайти лідера в його найближчому оточенні, він продовжить переміщення до сусідньої незайнятої комірки випадковим чином або, альтернативно, збереже свою поточну позицію.

```
Input: MAPF instance, list of leaders' positions
01  follower.leader ← GET_LEADER(leaders);
02  if distance(follower, leader) < follower.sight length or
      not follower sees leader() then
03      return random free neighbour ;
04      if not follower.obeys() then
05              return move(follower, leader, away)
06      else
07              return move(follower, leader, towards)
```

Алгоритм 7: рух агента-послідовника

Модель руху агентів-послідовників описана в *алгоритмі 7*. В залежності від моделі вибору агента-лідера, функція руху може бути адаптована під потреби моделі.

5.3. Відношення Лідер – Послідовник

Середовище представлено у вигляді сітки, яка розташована в двовимірному просторі, а агенти розташовані в окремих клітинках сітки.

Агенти-послідовники використовують локальні правила, які враховують безперервний візуальний доступ для визначення позиції свого агента-лідера.

Здатність послідовника слідувати за лідером залежить від видимості лідера. Візуальне сприйняття лідера L послідовником F залежить від евклідової відстані D між агентами: якщо D менше за значення параметра поля зору послідовника і в той же час між агентами немає перешкоди, якою може бути стіна, то послідовник F бачить лідера L.

До агента-послідовника можна застосувати різні моделі поведінки під час процесу евакуації:

- *Просте переслідування* – послідовник обирає найближчого лідера для переслідування і не намагаються евакуйовуватись самостійно. Лідери в свою чергу, не наглядають за своїми послідовниками, а лише слідують своєму шляху до виходу.
- *Призначений рій* – для кожного послідовника назначається свій лідер на час евакуації. Як і раніше, послідовники не намагаються самостійно шукати шлях до виходу, проте лідери знають про своїх послідовників і час від часу перевіряють чи ніхто з них не “загубився”. Якщо так, лідер може почекати або вернутись на певну кількість кроків назад щоб віднайти свого послідовника.
- *Частково інформовані послідовники* – послідовники, втративши лідера, можуть почати шукати шлях до виходу самостійно. Причому шлях не обов’язково повинен бути оптимальний.

В усіх моделях якщо послідовник, після втрати лідера, бачить в полі зору іншого агента-лідера, він стає частиною “зграї” цього лідера.

6 МОДЕЛЮВАННЯ ЕВАКУАЦІЇ ПІД ЧАС ПОЖЕЖІ

Для моделювання евакуації задачі MAEDO було розроблено модуль програмного забезпечення на мові програмування Java (JDK 17) з графічним інтерфейсом для спостереження за процесом поширення небезпеки та управління евакуацією. Графічний інтерфейс був розроблений на основі бібліотеки JavaFX. Програмне забезпечення побудоване з використанням паттерну проектування MVC.

Для моделювання процесу поширення пожежі в торговому центрі використовувався клітковий автомат. Оскільки поширення вогню є дифузійно-реакційним процесом, для його моделювання була побудована КА-модель процесу “поширення фронту”.

Загальна модель поширення вогню, записана у вигляді системи диференційних рівнянь, має вигляд

$$\frac{\partial u}{\partial t} + \nabla \cdot (w(x,t)\,u) = \nabla \cdot (K(u)\,\nabla u) + f(u,v,x), \frac{\partial v}{\partial t} = g(u,v),$$

де t — час, $x \in \Omega$ — просторова змінна, де область $\Omega \subset R^2$ представляє середовище, у якому може поширюватися пожежа, а $u = u(x,t)$ і $v = v(x,t)$ — скалярні невідомі. Тут u — безрозмірна температура, v — безрозмірна масова частка твердого палива. Більше того, $K = K(u)$ — заданий коефіцієнт дифузії. Функції $f(u,v,x)$ і $g(u,v)$ є реактивною частиною моделі. З [X] відомо, що коефіцієнт дифузії для процесу рівний $D \approx 2.3816$, а швидкість реакції α визначається за співвідношенням $\alpha = \frac{\lambda \Delta T}{h^2}$. За цими даними побудовано вищеописану КА-модель поширення пожежі.

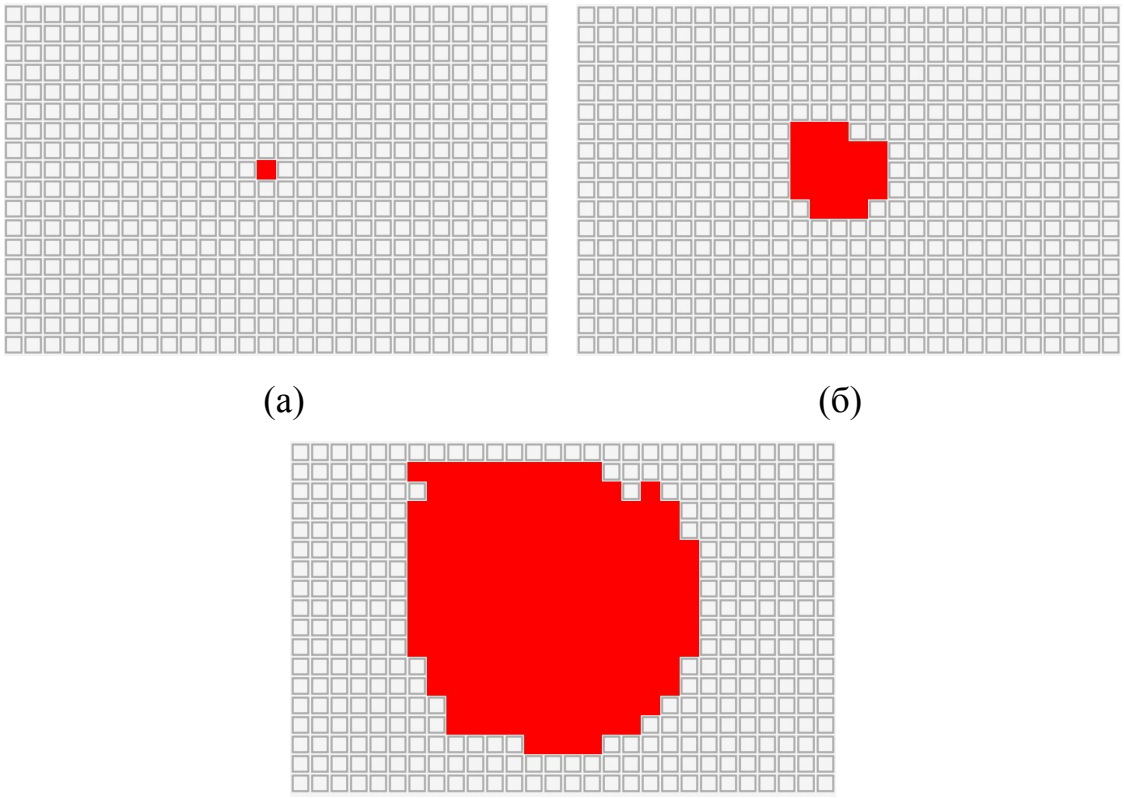


Рис. 3. Приклад поширення вогню КА-моделі.

Для тестування моделі була програмно змодельована будівля торгового-розважального центру Forum Lviv (рис.4.1). Всього в будівлі 4 виходи (позначенні зеленими клітинами).

На рис.4.2. – рис.4.4. зображений процес евакуації з торгового центру Forum Lviv з вищенаведеними параметрами моделі. З самого початку евакуації, модифікований алгоритм побудував шляхи до виходів, які уникають зустрічі з вогнем (рис.4.3). Коли вогонь поширився трохи більше, тим самим збільшив ризик поточних шляхів частини людей, алгоритм змінив їх шлях до іншого виходу, який є безпечнішим (рис.4.4). В результаті всі люди безпечно виходять з небезпечної зони

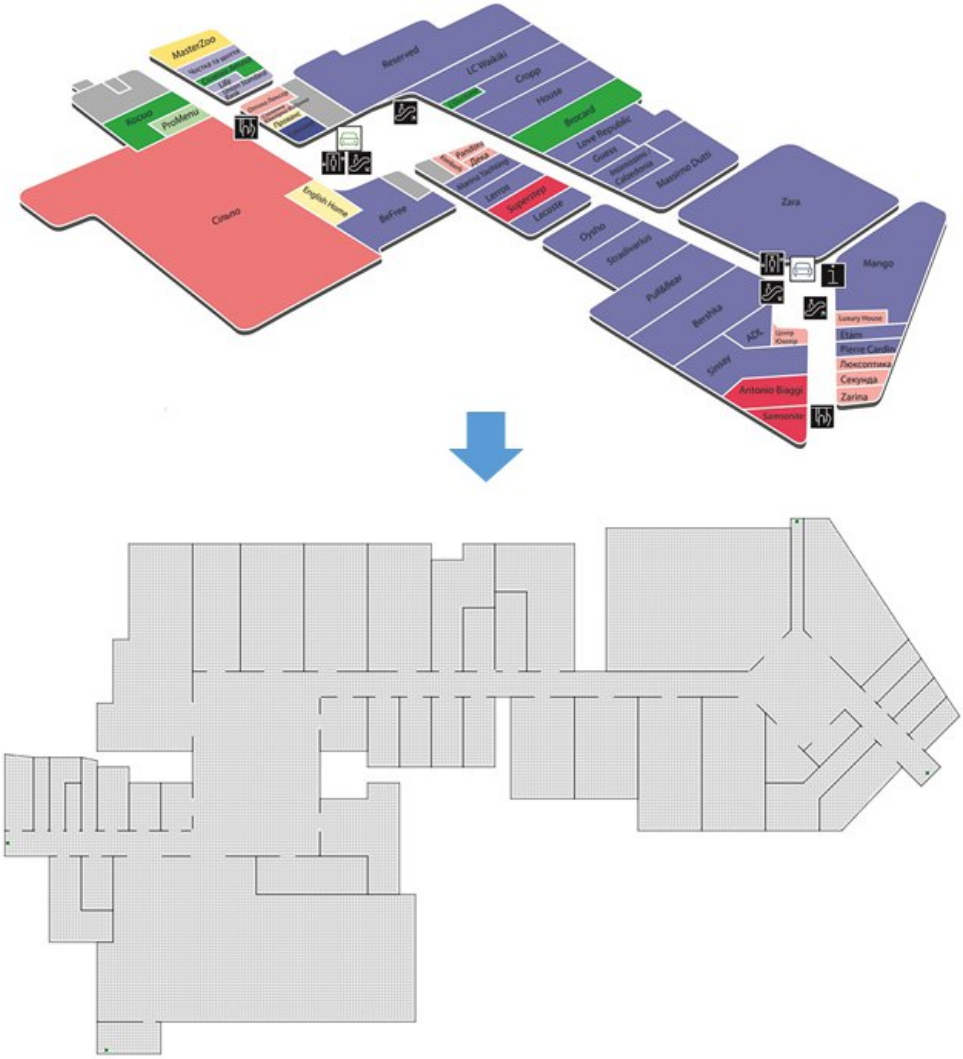


рис.4. Програмне зображення ТЦ Forum Lviv

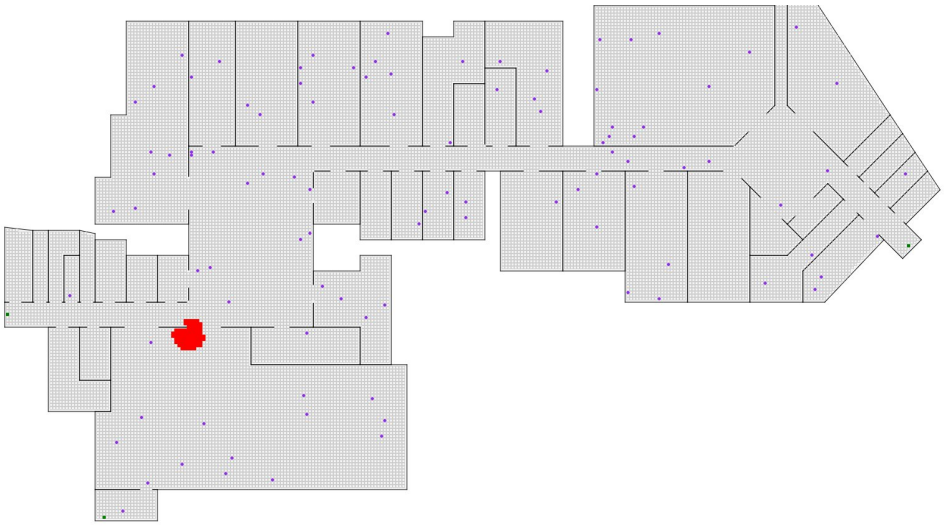
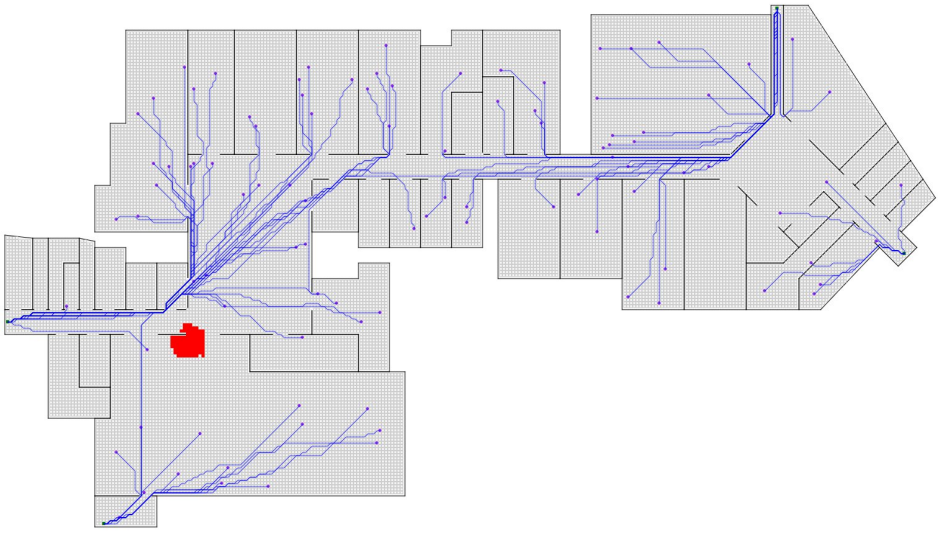
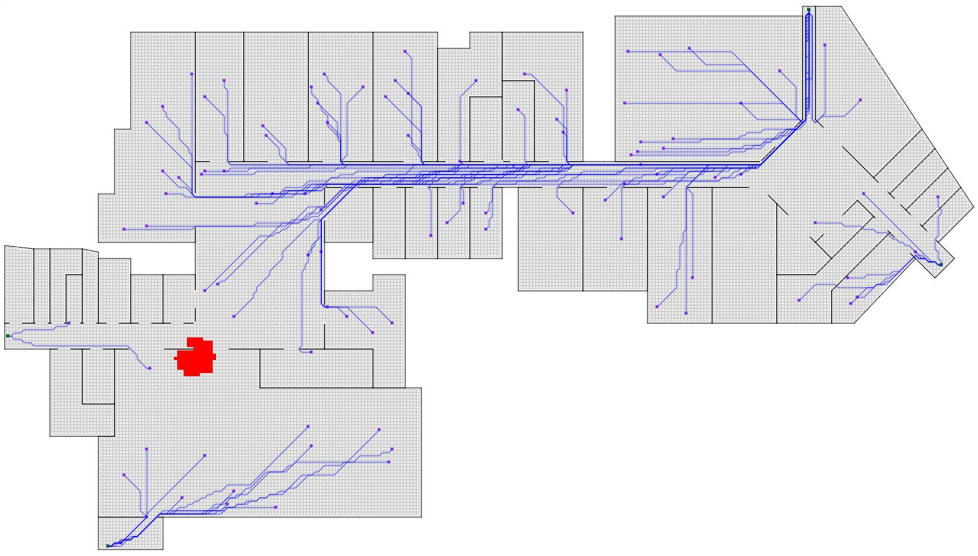


рис.5. Початок процесу евакуації



(а)



(б)

рис.6. Процес евакуації.

ВИСНОВКИ

В ході написання роботи були розглянуті алгоритми пошуку в динамічному графі та алгоритми багатоагентного пошуку. Була посталена комплексна задача багатоагентної евакуації в динамічному середовищі (MAEDO) та запропонований спосіб вирішення цієї задачі використовуючи ієрархічний підхід пошуку шляхів для агентів шляхом поділу їх на два класи: агенти-лідери та агенти-послідовники. Для лідерів застосовується багатоагентний пошук CBS (або суб-оптимальний варіант EECBS) у поєднанні з адаптованим до нього низькорівневим алгоритмом Tree-AA* для динамічного пошуку.

Для моделювання поширення небезпеки був використаний апарат клітинного автомату та механізм побудови КА-моделей реакційно-дифузійних процесів через інваріант. Інваріант є доволі зручний спосіб зв’язку КА-моделі з неперервною моделлю розгляданого процесу, адже дає змогу легко підібрати параметри для клітинного автомату. Таким чином була побудована КА-модель поширення пожежі в будівлі.

В результаті було розроблене програмне забезпечення для моделювання евакуаційних ситуацій шляхом задання небезпеки та початкової конфігурації в середовищі. ПЗ розроблений на платформі JVM з версією JDK 17, графічний інтерфейс побудований з використанням бібліотеки JavaFX, система зібрана за допомогою технології Maven. Розроблене ПЗ зручно застосовувати для тестування шляхів евакуації з будівель або інших середовищ, а також для тестування інших підходів планування евакуації.

Подальшим розвитком системи є побудова набору КА-моделей реакційно-дифузійних процесів та їх тестування і відкалібрування параметрів моделей, а також оптимізація алгоритму пошуку шляхів під час евакуації в умовах динамічного середовища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hern´andez, C.; Sun, X.; Koenig, S.; and Meseguer, P. 2011. Tree adaptive A*. In Proceedings of the 10th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS).
2. DoHoon Lee. 2015. Safe Pathfinding Using Abstract Hierarchical Graph and Influence Map.
3. Ураков А.Р.; Тимеряев Т.В. 2017. Алгоритм вирішення динамічної задачі пошуку найкоротших шляхів в графі.
4. R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, S. Kumar, E. Boyarski and R. Bartak. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks . In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2019.
5. G. Sharon, R. Stern, A. Felner and N. Sturtevant. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence,* 219, 40-66, 2015.
6. J. Li, A. Felner, E. Boyarski, H. Ma and S. Koenig. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* (in print), 2019.
7. E. Boyarski, A. Felner, G. Sharon and R. Stern. Don't Split, Try To Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS),* 47-51, 2015.
8. H. Zhang, J. Li, P. Surynek, S. Koenig and S. Kumar. Multi-Agent Path Finding with Mutex Propagation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS),* (in print), 2020.
9. J. Li, D. Harabor, P. Stuckey, H. Ma and S. Koenig. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI),* (in print), 2019.

10. J. Li, D. Harabor, P. Stuckey, H. Ma, G. Gange and S. Koenig. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. *Artificial Intelligence,* 301, 103574, 2021.
11. J. Li, W. Ruml and S. Koenig. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI),* 12353-12362, 2021.
12. О. Л. Бандман. Инварианты Клеточно-Автоматных Моделей Реакционно-Диффузионных Процессов. *Дискретные Модели Реальных Процессов №3(17), 109-120,* 2012.
13. Фон Нейман Дж. Теорія Самовідтворювальних Автоматів.
14. Kireeva A. Parallel Implementation Of Totalistic Cellular Automata Model Of Stable Patterns Formation.
15. Колмогоров А. Н., Петровский И. Г., Пискунов И. С. Исследование уравнения диффузии, соединенной с возрастанием количества вещества и его применение к одной биологической проблеме. *Бюлетень МГУ, секція А. 1937. Вип. 6. С. 1–25.*
16. О. Л. Бандман Нежими функціонування асинхронних клітинних автоматів, що моделюють нелінійну просторову динаміку.
17. Thomas C. Hasley. Diffusion-Limited Aggregation: A Model For Pattern Formation.