Összefoglalás:

A cikk az önálló, memória nélküli robotokból álló robotraj által végzett mintaképzés problémáját vizsgálja végtelen téglalap alakú rácson. A robotok célja, hogy egy tetszőleges, előre meghatározott mintát hozzanak létre egy adott kezdő konfigurációból, anélkül, hogy két robot ugyanarra a helyre kerülne vagy összeütközne. A cikk egy aszinkron ütemezési környezetben működő algoritmust mutat be, amely térbeli (space) és mozgás-optimalitás (move-optimal) alapján minimalizálja a robotok helyhasználatát és lépéseinek számát. A robotok az OBLOT modell szerint működnek, azaz nem rendelkeznek memóriával, és nincs explicit kommunikációjuk egymással. Az algoritmus aszimptotikusan mozgás-optimális és közel tér-optimális, és ez az első olyan javasolt algoritmus, amely mindkét kritériumnak megfelel ebben a modellben.

Prezentáció ötletek:

Cím: Mintaképzés végtelen téglalap rácson memória nélküli robotrajjal

Lehetséges pontok:

Bevezetés

* + - Swarm robotics alapjai
    - OBLOT modell és aszinkron ütemezés
    - Mintaképzési probléma (Arbitrary Pattern Formation - APF)

Robotok működése

* + - Névtelen, homogén, autonóm robotok
    - Nézz-Számolj-Mozog (Look-Compute-Move) ciklus

Motiváció és kihívások

* + - A térbeli korlátok jelentősége
    - Láthatósági problémák végtelen környezetben

Meglévő megoldások áttekintése

* + - Korábbi algoritmusok tér- és mozgás-optimalitása
    - Az új algoritmus előnyei: aszimptotikusan mozgás-optimalis és közel tér-optimalis

Az algoritmus működése

* + - Kezdő konfiguráció és globális koordináta-rendszer kialakítása
    - Lépcsőzetes mozgások a mintaképzéshez (vezető kiválasztás, célmintába illesztés)

Teljesítmény és optimalizáció

* + - Tér-komplexitás: D + 4
    - Mozgás-komplexitás: O(kD), ahol k a robotok száma

Következtetés és jövőbeli munkák

* + - Az algoritmus előnyei és lehetséges fejlesztési irányok

Mi mit akarunk megvalósítani?

Ötletek:

- Szimmetria felismerése és megszüntetése

Létrehozni egy programot, amely képes felismerni a kezdeti konfigurációk szimmetriáját és azokat megszüntetni.

- Optimalizált mintaalakítás különböző ütemezési modellekkel

Szimulálni az algoritmus működését különböző ütemezési modellekkel (szinkron, félig szinkron, aszinkron).

- Robusztusság vizsgálata véletlenszerű hibákkal

Szimulálni a hibákat (pl. robotok véletlenszerű leállását vagy hibás mozgását) az algoritmus futása közben, és vizsgálni a rendszer robusztusságát.

- Tér-optimalitás különböző rácsformákon

Megvizsgálni, hogyan változik a mintaképzési algoritmus hatékonysága különböző rácsstruktúrák (nem csak téglalap alakú) esetén.

- Dinamikus célminta módosítás

Szimulálni, hogyan reagál az algoritmus, ha futás közben változik a célminta.

- Versengő robotcsoportok mintaképzése

Szimulálni, hogyan viselkednek két vagy több robotcsoport, akik ugyanazon a rácson próbálnak különböző mintákat létrehozni anélkül, hogy ütköznének.

- Idő-optimalitás szimulációja különböző kezdő konfigurációkkal

Megvizsgálni, hogyan változik a mintaképzés ideje különböző kezdő konfigurációk esetén.

**PROBLÉMAKÖR, ARCHITEKTÚRA, IMPLEMENTÁCIÓ**

A megoldáshoz először általánosabb irányból közelítjük meg a feladatot és megállapítjuk a fő szempontokat, végül ezen szempontok alapján meghatározzuk a részfeladatokat. Tovább iteráljuk a részfeladatokon ezt a módszert, amíg kellően le nem bontjuk az egész feladatot.

Mi a fő cél? - Demó program írása a papírban leírt algoritmushoz. Mivel csak egy algoritmus van, nem kell foglalkozni több módszer összehasonlításával, párhuzamos bemutatásával.

A demó alapvető feladata, hogy vizuálisan bemutassa, azaz megjelenítse az algoritmus működését. Mivel az algoritmus 2-dimenziós síkon működik, értelemszerűen a megjelenítésnek is elég csupán 2-dimenziósnak lennie.

Elválasztjuk egymástól az algoritmus tényleges működését, illetve a megjelenítést. Adja magát valamilyen MVC architektúra kialakítása.

A **Model rétegen** valósítjuk meg az algoritmus működését, a szimulációt. Mindenképp szükségesek:

* Robot osztály megvalósítása
* Aszinkron ütemező (lényegében az algoritmus) megvalósítása

A **View réteg** valamilyen 2D-s grafikus csomag felhasználásával megjeleníti a szimulációt. A legminimálisabb funkcionalitás a következő:

* Grid megjelenítése
* Robotok kirajzolása a griden
* Vezérlő gombok megjelenítése

A **Controller réteg** köti össze a Model és View rétegeket. Lekezeli a felhasználói interakciókat. Szintén ez fogja kezelni a szimuláció aktuális állapotát és a nézet frissítését.

A Model szinten a következő eseményekre lesz szükség:

* **SIMULATION\_STARTS**
* **ROBOT\_LOOKS\_AROUND (index)**: Adott indexű robot aktiválva lett és épp szétnéz.
* **ROBOT\_COMPUTES (index)**: Adott indexű robot a ciklusában épp számol, hogy hova lépjen.
* **ROBOT\_MOVES\_TO (index, pozíció)**: Adott indexű robot az adott koordinátára mozdul.
* **ROBOT\_BECOMES\_IDLE (index)**: Adott indexű robot ciklusa véget ért és a robot készenlétben áll.
* **SIMULATION\_STOPS**
* **SIMULATION\_RESUMES**
* **SIMULATION\_COMPLETES**

Ezeket az eseményeket egy queue-ból kiolvassuk és feldolgozzuk a Controllerben, hogy mindig csak a legminimálisabb változást rajzoltassuk ki. (Azaz, ha egy robot nem mozdult, semmi sem történt vele, akkor ne rajzoltassuk ki feleslegesen újra). További optimalizálás, hogy mindig csak a nézetünkben levő robotokat rajzoltatjuk ki.

A feladat eddigi szempontjai alapján szükség lesz az alábbi metódusokra a Controlleren:

* **OpenView\_StartingPattern**: Megjelenít egy nézetet a kezdőállapot beállítására.
  + Célszerű egy gombot megjeleníti, amivel fájlból betölthetünk egy kezdőállapotot.
  + Szintén célszerű az állapotot kirajzolni (de még csak előnézetként)
  + Opcionálisan beletehetjük azt is, hogy az előnézetet interaktívan módosítani lehessen (adott cellába kattintás: ToggleCellForStartingPattern)
  + Legyen egy gomb, amivel elfogadjuk az előnézetben látható kezdőállapotot: **AcceptStartingPattern**. Elfogadás hatására állítsa is be a Modelnek.
* **OpenView\_TargetPattern**: OpenView\_StartingPattern metódussal analóg módon működik a célállapot beállítására.
* **SetStartingPattern**: A kapott kiindulási minta szerint elhelyez robotokat.
* **SetTargetPattern**: A kapott mintát beállítja célnak.
* **SetSimulationSlowness**: Az adott szorzó szerint lelassítja a szimulációt a Model.
* **StartSimulation**: Elkezdi a szimulációt (kezdő és célállapotok beállítása után).
* **StopSimulation**: Blokkolja a további számításokat az ütemezőben és a robotokban.
* **ResumeSimulation**: Engedélyezi az ütemezőnek és a robotoknak a számítások elvégzését.
* **MoveCamera**: A nézetet egy adott koordinátára mozdítja.
* **ZoomCamera**: A nézetet egy adott szintre zoomolja.
* **GetRobotsInView**: Csak azoknak a robotoknak az állapotát adja vissza a Model, amik a nézetbe esnek.
* **PopQueue**: Kiolvassa a legutóbbi lekérdezés óta történt változásokat.
* **EmptyQueue**: Kiüríti a változások sorát.
* **Draw**: Ez egy tipikus “render” metódus, ami egy végtelen ciklusban fut és kirajzoltatja a szimuláció állapotát. Itt valósítjuk meg a fenti optimalizálásokat.
  + Lekérdezzük a látható robotok állapotát.
  + Ha történt zoom, vagy nézetmozgatás, mindent újra rajzolunk és ürítjük a queue-t.
  + Egyébként kiolvassuk a queue-t és a nézetben levő robotok változásait rajzoljuk csak ki (beleértve azokat is, amik épp belépnek a nézetbe):
    - LOOK, COMPUTE és IDLE állapotok esetén nincs mozgás, csak reprezentáljuk a státuszt.
    - MOVE állapotban kirajzoljuk, vagy jelöljük a mozgás irányát
    - Az állapotokat különböző színekkel jelölhetjük:
      * IDLE: kék
      * LOOK: zöld
      * COMPUTE: piros
      * MOVE: sárga

Továbbá, a Controllernek le kell kezelnie a következő felhasználói eseményeket:

* Kattintás
  + Vezérlő gombokon: megfelelő esemény kiváltása  
    (start, stop, input megadása stb.)
* Drag (bal egérgomb lenyomva tart + egér mozgatása)
  + Szimulációs nézet mozgatása
* Görgő fel
  + Szimulációs nézet közelebb zoomolása
* Görgő le
  + Szimulációs nézet távolabb zoomolása
* **Opcionálisan**: egér mozgatásakor:
  + Kurzor egy robot fölött van:
    - Robot állapotának szöveges kiírása
      * Pozíció
      * Pillanatnyi státusz
      * Lefutott ciklusok száma
      * Megtett lépések száma (tényleges)
      * A négy állapotban töltött eddigi idő tételesen

A Model működéséhez az alábbi szempontokat vegyük figyelembe:

* A szimuláció egy külön szálon fusson, amit szabadon blokkolhatunk Stop utasítással, majd folytathatunk Resume utasítással.
* A robotokat úgy kell megvalósítani, hogy ne tároljanak információt. Egyik sem tudja, eredetileg honnan indult. Egyik sem tudja, milyen lépéseket tett meg. A robotokkal kapcsolatos statisztikát maga a Model vezesse/számolja!
* Valahányszor egy robot aktiválódik, a saját pozícióját tekinti az origónak. LOOK fázis során ehhez viszonyítva méri fel a többi robot helyzetét. Továbbá, új gyermek szálon induljon el a ciklusukat lebonyolító folyamatuk. Amint végez a ciklus, érjen véget a gyermek szál.
* A robotok minden státuszváltásnál jelezzenek a Modelnek. A ciklusuk végén egy IDLE jelzést küldjenek.
* Kezdetben a szimulációnak ne legyen kezdő és célállapota. Ezeket a felhasználó fogja megadni.
* A szimuláció csak akkor kezdhető el egy Start utasítással, ha van beállítva kezdő és célállapot, és ezek azonos mennyiségű cellát határoznak meg.
* Minden utasítás (Start, Stop, Resume) váltson ki egy eseményt a Modelnek.
* Minden eseménykiváltás tartalmazzon egy időpontot is, legalább ezredmásodpercre pontosan!
* A Model az eseményeket gyűjtse egy queue-ba. A queue mindig üresen induljon egy szimuláció elején. A Controller számára vezessen a Model egy indexet, hogy meddig olvasta azt ki, és csak az az után következőket adja neki át.
* A robotok aktiválását egyetlen ütemező (Scheduler) határozza meg. Ennek a működése gyakorlatliag az algoritmus szíve.

Az ütemező megvalósításához alaposan értelmezni szükséges a papírt.

Elsőként, a kiinduló fázis alapján meg kell határozni két kinevezett robotot: egy tail és egy head robot. Ezeknek speciális szerepe van az algoritmus működése során.

Az algoritmus HÉT fázisból áll. Minden robot kap egy pillanatnyi konfigurációt, amikor aktiválódik. A robot a konfiguráció alapján eldönti, hogy melyik fázisban van.

**... további értelmezés szükséges az algoritmus működését illetően**

A szimuláció futását célszerű úgy kialakítani, hogy tetszés szerint lassítható legyen. Ezáltal látványosabb lehet, ahogy a robotok dolgoznak és mozognak. Ehhez egy globális szorzót állítunk be, amivel mesterségesen megtöbbszörözzük a műveletek futási idejét. A legkisebb értéke 1 lehet, azaz ilyenkor nincsen lassítás. Egynél nagyobb értékek esetén a következőt tesszük: (Legyen **S** a lassítás mértéke!)

* Megmérjük, hogy egy adott ciklus-fázis mennyi idő alatt végez, ez legyen **t**
* Az adott szálat **(S - 1) \* t** időre felfüggesztjük, és csak utána engedjük tovább dolgozni

A View réteghez az alábbiakat érdemes megvalósítani

* Tudjon kezelni külön nézeteket (szimuláció, inputmegadás, és ami még eszünkbe jut)
* Pár alapvető komponenst, vezérlőt meg kell tudnia jeleníteni, akár különféle állapotokban.
  + **Gomb**
  + **Egyszerű szöveg**
  + **Grid**
  + **Robot**
    - Amikor egy robot mozog, hasznos lehet egy nyilat is elé helyezni, hogy a szimuláció megállításakor is egyértelmű legyen, mi történik.
  + **Csúszka**

A **szimulációs nézet** jelenítse meg a gridet, a látható robotokat az aktuális státuszukban, tegye lehetővé a zoomolást és a szabad mozgatását a látható területnek!

Kezdetben a szimulációs nézet jelenjen meg, üresen. A következő gombok látszódjanak a felületen (a feliratukat érzés szerint nevezzük el). Nevezzük ezt a MAIN állapotnak:

* **[ Kezdőállapot** **megadása ]**: Megnyitja a kezdőállapot megadására szánt nézetet
* **[ Célállapot** **megadása ]**: Megnyitja a célállapot megadására szánt nézetet
* **[ Szimuláció** **indítása ]**: Elindítja a szimulációt. Csak akkor legyen aktív a gomb, ha már vannak valid inputok!

Egy **minta megadására szánt nézet** lehet generalizált is, de mivel csak két lehetőség van, mehetünk egyszerűbb megoldással, két külön komponenssel.

Általánosan, az input nézet jelenítsen meg egy gridet. Ha még nincs megadva az adott inputnak érték, üres legyen a grid, egyébként jelölje azokat a mezőket, amik a mintát képezik.  
Jelenítse meg továbbá a következő gombokat:

* **[ Betöltés** **fájlból ]**: Megnyit egy tallózót, amivel mintát tartalmazó fájlt lehet betölteni. A betöltés felülírja a pillanatnyi gridet (nincs overlapping, vagy ilyesmi). Ha sikertelen a betöltés, ne változzon az addigi input a griden, csupán egy hibaüzenet jelenjen meg.
* **[ Véletlen ]**: A griden bejelölt mezők számával megegyező mennyiségű mezőt bejelöl, véletlenszerűen.
* **[ Elfogad ]**: A grid aktiális állapotát elfogadja és beállítja a Modelnek input mintaként, illetve visszavált a szimuláció nézetre. Validáció: üres és egyelemű mintát ne fogadjon el az alkalmazás! (triviális esetek)
* **[ Reset ]**: Üresre állítja a gridet.
* **[ Mégsem ]**: Nem állít át semmit, pusztán visszalép a szimulációs nézetbe.

Elfogadás esetén az adott mintát adja át a View a Controllernek, az pedig a Modelnek.

Szimuláció indítása után ne látszódjanak a kezdeti gombok a MAIN állapotból. A **[ Szimuláció** **indítása ]** gomb helyén **[ Stop ]** gomb legyen, ami leállítja a szimulációt ideiglenesen. Ha áll a szimuláció, ugyanez a gomb a**[ Folytatás ]** funkcióval rendelkezzen, ami folytatja a szimulációt. A szimuláció végeztével a gomb ne csináljon semmit, inaktív legyen.

Látszódjon továbbá egy csúszka, amivel a szimuláció sebességét lehet valós időben befolyásolni.

Ha a szimuláció áll, vagy véget ért, egy további **[ Reset ]** gomb látszódjon, ami véget vet a szimulációnak és a MAIN állapotba lép a felület. Az inputokat ne ilyenkor törölje a program, hanem lehessen azonnal újraindítani a szimulációt.

A zoom és nézetmozgatás végig maradjon elérhető, amíg a szimulációs nézetben vagyunk.

**EXTRÁK**

Pár extra funkció még eszembe jutott, ha esetleg lesz időnk és fancy dolgokat akarunk.

**Szimulációs idővonal**

Mivel a szimuláció minden eseményt eltárol, így erre építve egy idővonalon ki lehetne rajzolni valós időben, hogy az algoritmus melyik fázisa mikor zajlott. Ehhez elég lenne a robotoknak az események jelzésekor azt is közölni, hogy melyik fázist állapították meg, amelyikben épp voltak. Az adott fázishoz tartozó eseményeket valamilyen egyedi színnel jelölhetnénk az idővonalon.

Azt is meg lehetne valósítani, hogy ha épp áll a szimuláció, akkor az idővonalon szabadon vissza lehessen menni egy korábbi pillanatra és onnan lehessen folytatni a szimulációt.

**Megtett útvonal**

Szintén a tárolt események alapján lehetne olyat csinálni, hogy amikor rákattintunk egy robotra, akkor kirajzolódik az útvonal, amit megtett.

Egy kapcsolót is felvehetünk, ami az összes robot útvonalát kirajzolja. Vizualizáció szempontjából érdekes lehet.

Meet memo: 20240925

Bevezető diák: csapat, téma összefoglalása, bemutatása, alapfogalmak, 4-5 dia

Megvalósítás: mit, hogyan, környezet (Java + library),

Bevezetés

* + - Swarm robotics alapjai
    - OBLOT modell és aszinkron ütemezés
    - Mintaképzési probléma (Arbitrary Pattern Formation - APF)

Robotok működése

* + - Névtelen, homogén, autonóm robotok
    - Nézz-Számolj-Mozog (Look-Compute-Move) ciklus

Motiváció : ezt kidolgozni pár gondolatban

Meglévő megoldások áttekintése (néhány érdekesebb bemutatása)

* + - Korábbi algoritmusok tér- és mozgás-optimalitása
    - Az új algoritmus előnyei: aszimptotikusan mozgás-optimalis és közel tér-optimalis

Az algoritmus működése (átfogó képben)

* + - Kezdő konfiguráció és globális koordináta-rendszer kialakítása
    - Lépcsőzetes mozgások a mintaképzéshez (vezető kiválasztás, célmintába illesztés)

Teljesítmény és optimalizáció

* + - Tér-komplexitás: D + 4
    - Mozgás-komplexitás: O(kD), ahol k a robotok száma (ebbe ne menjünk bele túl mélyen)

Mi mit fogunk csinálni

* + - Cél - implementáció, szimuláció készítése
    - Miben valósítjuk meg és hogyan (Java-ban MVC architektúrában)
    - Vizualizáció (tervben, ha marad rá idő)