EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSELMÉLET ÉS
SZOFTVERTECHNOLÓGIA TANSZÉK

NetWorkers - felderítés és alapvető hálózat kiépítése raj intelligencia segítségével

Konzulens(ek):
Dr. Istenes Zoltán
egyetemi docens (ELTE-PSZT)
Csikor Levente
PhD hallgató (BME-TMIT)

Készítette:
Nagy Zoltán (NAZHAET)
nappali tagozat
programtervező matematikus hallgató

Budapest, 2011.

temabejelento helye

Tartalomjegyzék

1. Bevezetés **3**

2. Feladat pontosítása **5**
2.1. Steiner-fa 7

3. Networkers algoritmus **8**
3.1. Áttekintés 8
3.1.1. Üzenetek 10
3.1.2. Elvárások a kliensponktól 11
3.1.3. Indulás 11
3.2. Pásztázás (***swéep***) 12
3.3. Kollaboratív felderítés (***récon***) 15
3.3.1. Rendeződés 15
3.3.2. Erőtér 16
3.3.3. Lánchossz 20
3.3.4. Pásztázási fázis nélkül 20
3.4. Gráfcsúcsok - ***tówer*** 23
3.4.1. Rugóerők 23
3.4.2. A hálózatot reprezentáló gráf 27
3.4.3. Több forrású mélységmérés 28
3.4.4. Prioritás 35
3.4.5. Kommunikációs minta 38
3.4.6. Tornyok működése 40

4. Összefoglalás **42**

5. Irodalomjegyzék **44**

Ábrák jegyzéke

1. *Szituáció* 5
2. *Tervezett feltöltés* 12
3. A legyező alapvektorai 13
4. A pásztázás végeredménye 14
5. *Felderítők beszúrása* 15
6. *Erőtér* 17
7. *Két torony kerésési törésszöge* 19
8. *A rugóerőkkel való probléma szemléltetése* 23
9. *Szabad mozgási tér (fdist)* 25
10. A $\hat{\boldsymbol{k}}$ irányába való ellépés 26
11. *Mélységi javítás* 28
12. *Terminálisok (*$\boldsymbol{T}$*)* 31
13. *A* $\boldsymbol{T}$*-t lefedő* $\boldsymbol{S_T}$ *Steiner-fa* 31
14. *Pont hozzáadása* 32
15. *A* $\boldsymbol{T'}$*-t lefedő* $\boldsymbol{S'_T}$ *Steiner-fa* 32
16. *Az* $\boldsymbol{u}$ *csúcs és szomszédai* 36
17. $\widehat{\boldsymbol{tower}}$ comm 38

1. Bevezetés

Napjainkban az autonóm rendszerek térhódításának köszönhetően már fel sem tűnik az embernek a mindennapi internetezés közben, hogy már néhány weboldalon ún. robotok „próbálják" hatékonyabbá és kényelmesebbé tenni a böngészést. Ugyanakkor a robotikával és autonóm ágensekkel foglalkozó szakirodalomban fellelhető írások gyakran már-már egy konkrét feladatot ragadnak meg, s néhány esetben már élesben működő robotokkal is dolgoznak [9], [12],[2]. A technológia gyors fejlődésének köszönhetően az utóbbi években elterjedt ARM processzorok is képesek már autonóm ágensek futtatására, így nem is tűnik olyan távolinak több ágensű alkalmazások megvalósítása ezen - a mobilitásra is képes eszközöket nagymértékben támogató - architektúrákon.

Hálózatok kialakítása és karbantartása az egyik alappillére a mai világunkban zajló kommunikációknak. A ma használatban lévő hálózati elemek nagyrészét szakszerűen és körültekintően telepítik, de előfordulhatnak olyan szituációk, amikor ezekre nincs lehetőség, vagy idő hiányában nem kivitelezhető, esetleg környezeti tényezők akadályozzák.

Ilyen esetekben előfordulhat, hogy bizonyos fontosnak megjelölt pontok - pl. városok - között kell egyfajta hálozati infrastruktúrát létesíteni, azonban a mai világban sokkal reálisabb helyzet lehet egy katasztrófasújtotta területen a mentőcsapatokkal történő kommunikáció biztosítása egy - a már infrastruktúrával nem rendelkező - területen. A vezetéknélküli technológiáknak köszönhetően lehetőség nyílik ezen problémák valamilyen szintű megoldására. Ebben az irányban már történtek előrelépések az úgynevezett késleltetés toleráns hálozatok terén[4][10], melynek lényege, hogy a kommunikáló felek az üzeneteiket úgy próbálják eljuttatni az egyik pontból a másikba, hogy nagyban támaszkodnak a csomópontok mozgására. A csomópontok tehát - a levelezési rendszerknél már megismert - „tárol-és-továbbít" elvet alkalmazzák az üzenetek célbajuttatására. Azonban az ilyen helyzetekben, ahol a legnagyobb ellenség az idő, ott a legfontosabb a közel 100%-os megbízhatóság, amit ezek nem képesek biztosítani. Ennek elérése érdekében jobbnak tűnik egy tényleges, de ideiglenes infrastruktúra kiépítése, amelyet a lehetőségekhez képest gyorsan és minimális számú eszközt felhasználva kell(ene) kivitelezni. Amennyiben maradandó hálózatot szeretnénk létesíteni a kliensek közé közvetítő egységek be-

vetésével, a Steiner-fa problémájának egy változatával találkozunk, melyről ismert, hogy NP-teljes.

Ezen téma sok más külföldi kutatót is foglalkoztat. A több ágens közötti kommunikációból adódó problémák megoldásához Torben Weis és Helge Parzyjegla [14] dolgozott ki egy, a szerepkörök kiosztásán alapuló módszert, mely az általam javasolt algoritmus megvalósításában is fontos szerepet játszott. Az ágensek közötti optimális távolság megtartásához (Andrew Howard és társai [7]) sikerrel alkalmaztak rugóerőket . Yi Zou és Krishnendu Chakrabarty [16] szenzor szétszóródási algoritmusa is rugóerős meggondolásokon alapszik csoport vezetők bevonásával. Sok szenzorhálózat kialakító algoritmus használ fel virtuális erőket a lefedés növeléséhez ([13, 16]). A mobil ágensek mozgásához kapcsolódik Fruchterman és társainak gráfrajzolási módszere is [5]. Jól koordinált mozgásra láthatunk példát Zhang és társai munkájában [15], melyben egy nagyon érdekes problémát oldanak meg: adott egy sokszög, melyet el szeretnénk szállítani, kérdés, hogyan álljanak fel a robotok.

A dolgozatban tárgyalt problémakörhöz a legközelebbi írások([6, 7, 12, 6, 8]), általában egy zárt környezetet próbálnak meg kitölteni szenzorokkal, és ennek segítségével kialakítani a hálózatot .

Jelen munka bemutat egy algoritmust (networkers), melynek célja, hogy minél nagyobb hálózatot hozzon létre egy síkon különböző fontosnak megjelölt pontok között, külső beavatkozás nélkül, folyamatosan adaptálódva a változásokhoz, behatásokhoz. Ezen kijelölt pontok lehetnének városok, melyek esetében egyfajta hálózatkiépítésben segíthet, illetve lehetnének sérültek, akiknek így lehetőséget nyújthat a mentőcsapatokkal való kommunikációra. A feladat természetéből adódóan a kezdeti felderítési fázisra is nagy hangsúlyt kell fektetni.

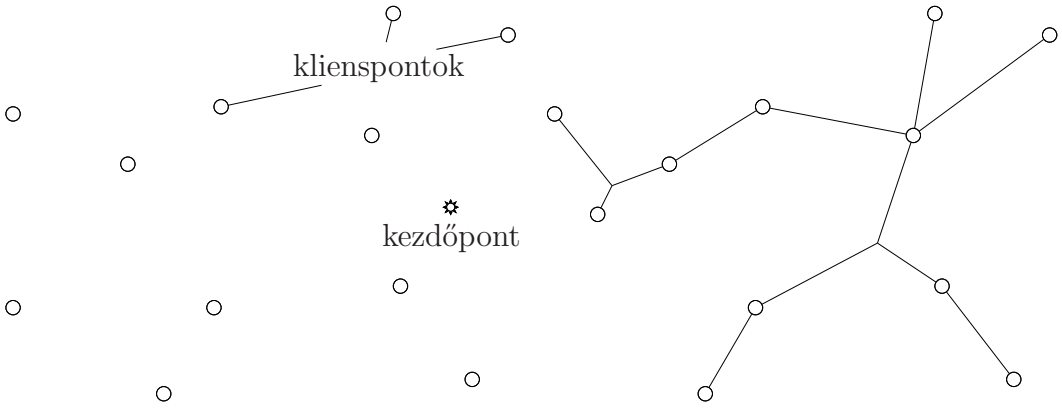
A következő fejezetben bemutatásra kerül a konkrét megoldandó probléma, ezután ismertetem a javasolt networkers algoritmust. Végül következtetéseket vonok le, és kitérek a jövőbeni lehetőségekre.

2. Feladat pontosítása

Mivel jelen esetben nem feltétlenül egyértelmű, hogy mi is a feladat, ezért ennek tisztázása elkerülhetetlen. Adott egy sík, melyen előre nem ismert helyeken olyan egységek vannak (kliens pontok), melyek nem tudnak kommunikálni egymással. A cél egy olyan kommunikációs réteg kialakítása, melynek segítségével ezen pontok egymással kapcsolatba léphetnek.

A kliens pontok idővel akár meg is szűnhetnek, ezért az algoritmusnak meg kell tudni birkóznia a változó világgal, melyben ezen pontok megszünhetnek vagy meghibásodhatnak.

A feladat megoldásához azonos felépítésű autonóm hálózati eszközök (networkerek) állnak rendelkezésre, melyek egy adott sugáron belül megbízhatóan tudnak kommunikálni. Helyváltoztatásra képesek, de az elérhető maximális sebesség korlátozott. Ezen robotok egy starthelyről indulnak, ahol mindannyian egymás kommunikációs távolságán belül helyezkednek el. Az 1a ábrán látható egy elképzelt kezdeti állapot. A networkerek semmilyen információval nem rendelkeznek a kliens pontokról, de a céljuk, hogy valamilyen hálózatot építsenek ki azok közé (1b).



(a) Kezdeti állapot (b) Egy lehetséges hálózat

1. ábra. *Szituáció*

Feltételezem, hogy a rendelkezésre álló eszközökön a memória korlátos, ezért a megoldó algoritmus nem használhat fel tetszőleges mennyiségűt ebből az erőforrásból. Így csupán csak lokális információkra alapozhatóak az ágensek által meghozott döntések.

A robotok az alábbi fontos jellemzőkkel rendelkeznek:

Azonosítás:

Mindegyik egység rendelkezik egy saját azonosítóval, mely garantáltan nincs kiosztva mégegyszer a bolyon (ezalatt az összes robot értendő) belül.

Látókör:

A robotok rendelkeznek egy korlátozott látókörrel ($\boldsymbol{R}$), mely funkcióját egy valódi alkalmazásnál egy szonár vagy egy radar láthatná el. A kommunikáció szoros összefüggésben áll a látókörrel. Két egység között akkor és csak akkor zajlik le a kommunikáció hiba nélkül, ha azok látják egymást.

Mobilitás:

Az egységek minden lépésben bármelyik irányba képesek maximálisan $\boldsymbol{v_{max}}$ sebességgel a pozíciójuk megváltoztatására.

Közös óra:

A szimuláció egyszerre kezdődik minden résztvevő robot számára és mindannyiuk órája a **0**-n áll.

Összefüggőség:

Ha a robotkra úgy tekintünk mint pontokra és minden olyan csúcspárt, amely $< \boldsymbol{R}$ távolságra van egymástól összekötünk, akkor egy összefüggő gráfot kapunk.

Kommunikációs sebesség:

Minden információ pontosan egy időegység alatt ér el a küldőtől a címzetthez.

Körökre osztott világ

Az összes networker egy időegység alatt egyet lép.

2.1. Steiner-fa

Átfoglamazva a felvetett problémát a matematika nyelvére: „adott a síkon k darab pont, keressük azt a minimális élköltségű fát, mely lefedi az összes pontot”, ez a probléma az euklideszi Steiner-fa probléma néven vált ismertté - és az NP-nehéz feladatok körébe tartozik, így a kiépülő kezdetleges gráf javításához olyan módszereket kell bevetni, ami Steiner fa előállítását segíti elő.

3. Networkers algoritmus

Úgy próbáltam megoldani a feladatot, hogy a kialakuló algoritmus továbbfejleszthető lehessen. Az egyik távoli cél, hogy a hálózat kiépítésében résztvevő networkerek esetleges meghibásodási hibáival szembe tudjon majd szállni. Ennek a célkitűzésnek eléréséhez az egyes networkerek autonóm egységekként önálló döntéseket hoznak és rajként próbálják megoldani együtt a problémát - ez a hosszú távú cél megkönnyíti, de sok helyen megnehezíti az algoritmus kidolgozását.

3.1. Áttekintés

Mielőtt még mélyebben szó lenne a konkrét megoldó algoritmusról, először fontos, hogy az olvasó egy vázlatos képet kapjon róla általánosságban.

Az algoritmus több szerepkörben tekint az egyes networkerekre, a különböző szerepkört betöltő networkerek együttműködve segítik egymást. Az egyes networkerek a szerepük szerint más-más céllal cselekszenek:

Pásztázó:

Az algoritmus indulását megnehezíti az egy pontból való indulás. A kezdeti zsúfolt helyzetet a pásztázók oldják fel egy formációval, melyről a 3.2 fejezetben lesz szó. A pásztázók feladata megszüntetni azt, hogy egy-egy networker látótávolságában ne legyen túl sok társuk.

Felderítők:

Feladatuk, hogy láncokba fejlődve új kliens pontokat kapcsoljanak az épülő hálózathoz. A felderítők működéséről a 3.3 fejezetben lesz szó.

Torony:

A toronyként viselkedő robotok a hálózat infrastruktúráját biztosítják. Jelentősen nem változtatják meg a helyzetüket, javarészt a hálózat életbentartásához szükséges robotszámot próbálják csökkenteni.

További mindig teljesülő tulajdonságok:

- mindig létezik legalább egy torony állapotú robot, s ezáltal mindig létezik az a gráf, melyet a torony állapotú robotok feszítenek.

- nem szakadhat le egy ágens sem

Minden körben három fő műveletet végeznek el az egyes ágensek:

1. üzenetek feldolgozása
2. döntés az állapotváltásról
3. elmozdulás

3.1.1. Üzenetek

Az egyes robotok a betöltött szerepüknek megfelelően különböző üzeneteket használnak az egymással történő kommunikációhoz, melyek típustól függetlenül a következő időpillanatra érnek el a feladótól a címzettig.

Az üzeneteket egységesen dőlt betűkkel és felülvonalással jelölöm.

- $\overline{scan\_info}$: az algoritmus tartalmaz egy véletlenen alapuló üzenetküldést, mely lehetővé teszi a fogadó számára, hogy tudomást szerezzen valamely szomszédja azonosítójáról és állapotáról.

- $\overline{interlock}$: Két ágens közötti kommunikációs él kialakításához szükséges üzenet. Az ilyen élek mindkét végpontja torony szerepet tölt be legkésőbb ezen üzenet fogadása után.

- $\overline{scan\_update\_req}$: a felderítő robotoknak mindig van egy „ismerőse ”, akihez mérten a láncformációt fel szeretné venni - de ehhez szüksége van a formáció irányvektorára, ami időközben változhat, ezért minden körben megkérdezi.

- $\overline{scan\_update}$: a felderítésben résztvevő ágensek minden körben tájékoztatják a szomszédjukat a jelenleg alkalmazott keresési irányról - ezt az üzenetet csak válaszként kaphatja egy robot a $\overline{scan\_update\_req}$ -re.

- $\overline{link\_error}$: Az előbbi üzenet hatására kialakított kommunikációs élek megszüntetéséhez szükséges.

  Ahogy az $\overline{interlock}$ üzenetnél, itt is a feladó-küldő élről történik nyilatkozás.

- $\overline{measure\_depth}$: a gráf maximális átmérőjét méri

- $\overline{length\_opt}$: beágyazásra került egy véletlen többforrású gráfmélységmérés - ezt az üzenettípust külön fejezetben tárgyalom

3.1.2. Elvárások a kliens pontoktól

Mivel az algoritmusnak megvan a maga menete, ezért a kliens pontoktól is megkövetel egy bizonyos szintű protokollt, mely a következőképp fogalmazható meg: amennyiben egy összekötővel nem rendelkező kliens pont $\overline{scan\_info}$ üzenetet kap, azt egy $\overline{interlock}$ üzenettel nyugtáznia kell. Emellett, ha már létezik összekötője, de az távolabb van, mint a kérdező, akkor azt a megfelelő üzenetekkel cserélje le a közelebb lévőre.

3.1.3. Indulás

Minden egység kereső($\widehat{\boldsymbol{recon}}$) és továbbító($\widehat{\boldsymbol{tower}}$), ezt az aktuális állapota határozza meg. Mindemellett egy kezdeti($\widehat{\boldsymbol{init}}$) állapottal, valamint egy szétszóródási fázissal($\widehat{\boldsymbol{sweep}}$) is rendelkeznek.

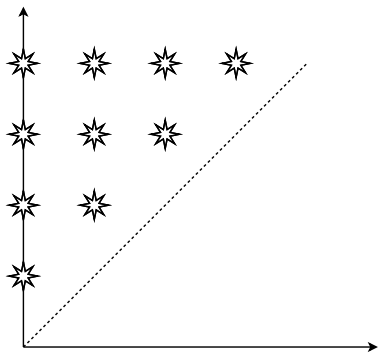
A kezdeti állapotban minden ágens a szétszóródásra készül fel, kivéve az **1**-es sorszámú robotot, amely azonnal toronnyá válik.

Az **1**-es ágens toronnyá válása kötelező, mert az algoritmus csak a már meglévő gráfot bővíti az összefüggőség biztos megőrzése érdekében.

3.2. Pásztázás ($\widehat{sweep}$)

A lényegi működés eléréséhez elengedhetetlen egy inicializálási fázis. Ez egy előfelderítési állapotot takar, mely nélkül a hasznos munka beindulásáig sok időnek kellene eltelnie.

Ennek a késlekedésnek az okáról a felderítők bemutatása után a 3.3.4 részben lesz szó.



2. ábra. *Tervezett feltöltés*

Alap esetben a síkon egy pontból indul mindegyik robot, s mivel minden robot minden másikat lát, ezért a beépített véletlen algoritmus lassan és túl nagy láncot kezdene építeni.

Ezért szükséges egy kezdeti alakzat, melynek megválasztásakor fontos volt, hogy minél hosszabb láncok alakulhassanak majd ki belőle - ha lehet minden irányba. Ennek elérése érdekében én egy 3 levelű legyezőszerű formációt választottam.

Legyező

A cél egy olyan $\boldsymbol{f(s)} \rightarrow \mathbb{N} \times \mathbb{N}$ függvény meghatározása, mely feltölti a sík $\boldsymbol{y \geq 0, x < y}$ részét sorról-sorra az 2. ábrán látható módon. Kezdve:

$$\boldsymbol{f(0) = [0,1], f(1) = [0,2], f(2) = [1,2], f(3) = [0,3]...}$$

Tehát a kérdés:

$\boldsymbol{f_y(s * (s+1)/2) \approx s}$, ha felteszem azt, hogy $\boldsymbol{n}$ lenne az $\boldsymbol{f_y(s)}$ értéke, akkor az

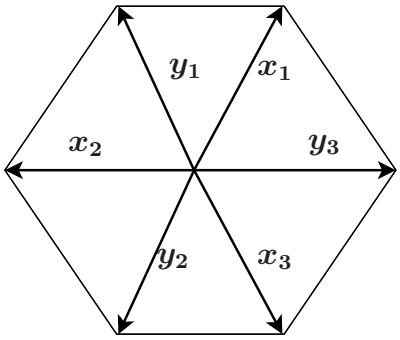
alábbiak teljesülnének:

$$s = n * (n + 1)/2$$
$$0 = n^2 + n - 2s$$
$$n = (-1 + \sqrt{1 + 8s})/2$$

így:

$$f_x(s) = s - x(s) * (x(s) + 1)/2 \tag{3.1}$$
$$f_y(s) = x(s) \tag{3.2}$$

Mivel minden robot rendelkezik egyedi azonosítóval - , melyet 1-től kezdve kapnak meg - így a kialakuló pontok közül nem marad ki egy sem. Azaz a (3.1) és (3.2) egyenletek sorozatszámra való alkalmazása esetén az 2. ábrán látható eredményt kapjuk.

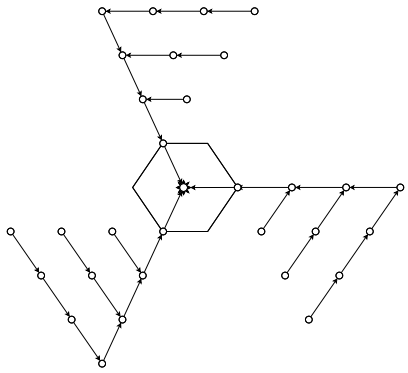


3. ábra. A legyező alapvektorai

A tervezett alakzat eléréséhez még szükség van arra, hogy **3** részre osszuk a robotokat a sorozatszámuk alapján $s'(x) = s(x) \mod 3$, valamint a formáció felvételéhez x, y irányoknak egy hatszög irányvektorait vegyük a 3. ábra szerint. Ezek után már fel tudnak állni a helyükre az ágensek.

A kialakított struktúra eredményeként a keresők egy olyan formációt vettek fel, melyben minden elem **1** sugarú környezetében van a nála **3**-mal kisebb sorozatszámmal rendelkező társa. Ez az elem lesz az, amit a kereső a vezetőjének fog választani, s az így kialakuló kezdeti struktúra a 4. ábrán látható.

A pásztázás utolsó lépéseként a networkerek üzenetet küldenek ($\overline{scan\_info}$) minden látható szomszédjuknak és felderítő állapotba lépnek át. Ezeket az üzenete-



4. ábra. A pásztázás végeredménye

ket már keresőként értelmezik, majd fel fogják venni a helyes szomszédjukat mint útmutatót.

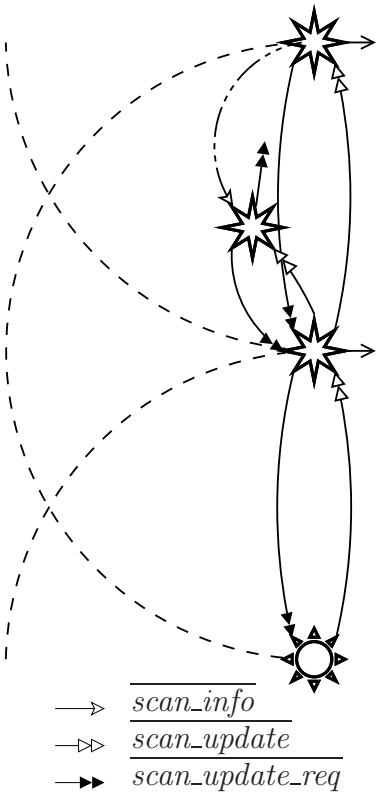
3.3. Kollaboratív felderítés ($\widehat{recon}$)

A feladat megoldásához szükséges, hogy az ágensek együttműködve derítsenek fel minél nagyobb területet. Mivel nem „veszíthetik el egymást szem elől", ezért a legjobb, ha minél hosszabb láncokba fejlődve kutatnak új kliens pontok után.

A felfeljődéshez szükség van arra, hogy minden ágens egy másik robotot mint ismerőst tartson számon, akihez mérten próbálja felvenni a formációt. Ettől a robottól fog útmutatást kérni és kapni az alakzatba történő rendeződéshez.

3.3.1. Rendeződés

Minden „társulni vágyó" robotnak egyértelmű helyének kell lennie egy már kialakult láncban. Ennek a feltételnek a sérülése esetén nem lehetne konkrétan eldönteni, hogy ki vezeti a láncot. A kezdetben adott azonosítók felhasználhatók arra, hogy a keresőláncokban minden ágensnek legyen fenntartott helye, mivel ha minden kereső a nála kisebbek között keresi a maximálist, akkor az tetszőleges részhalmazban egyértelműen létezik. Tehát egy keresőt vezető társnak az azonosítója minden esetben kisebb kell legyen a megvezetett sorozatszámánál.



5. ábra. *Felderítők beszúrása*

A 5. ábrán látható a keresők láncbaszerveződése. Minden egyes kereső az őt kalauzolótól kérdezi a keresési irányt (teli nyilak), s válaszként megkapja a jelenlegi irányt, ami alapján a vezető mellé kell állnia d_s távolságra.

Ha egy időpillanatban egy r sorozatszámú ágensnek lehetősége van egy C halmazból új vezető választására, akkor azt az s azonosítójú ágenst választja mely:

$$s = \max\{i | i \in C \wedge i < r\} \tag{3.3}$$

Tehát egy r robot kalauza mindig a nála kisebb sorozatszámúak közül a legnagyobb

azonosítójú.

Megjegyzés:

A fenti feltétel nem zárja ki, hogy egy időpillanatban egy felderítő több társának nyújtson útmutatást. Sőt egy éppen épülő lánc felépítése közben az új elemek megkeresik a láncban a helyüket.

Legrosszabb eset: egy pont körüli forgás

Amennyiben egy $\vec{t}$ irányban keresnek olyan körülmények között, hogy minden időpillanatban számítani kell arra, hogy jelenleg éppen egy körmozgást kell végezni egy O pont körül, akkor ha a keresőlánc n pontból áll - s feltételezve a legrosszabb esetet, az O ponttól sugár irányba helyezkednek el az ágensek - akkor az $i.$ ágens maximum $\frac{i}{n}v_{max}$ sebességgel mozoghat.

Ebből látható, hogy ennek a megközelítésnek van egy olyan mellékhatása, ami miatt a keresőlánc sebessége $\frac{1}{n}$-re fog visszaesni.

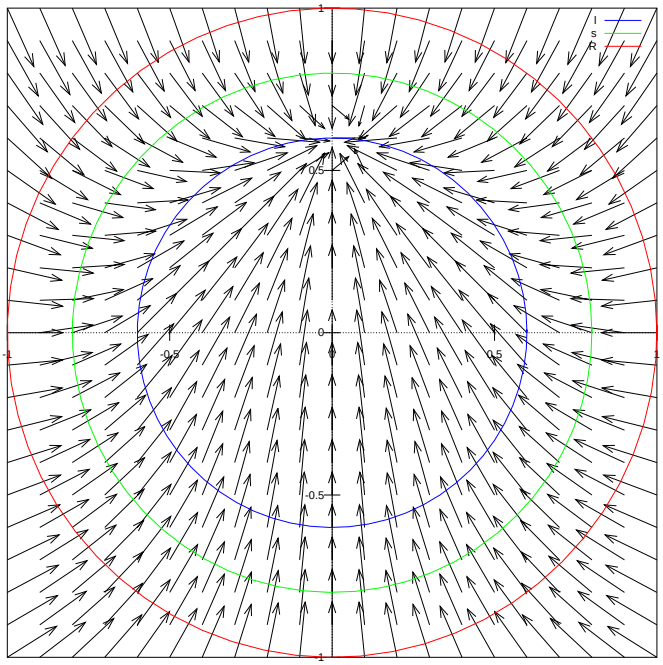
3.3.2. Erőtér

A robotok a lánc formációt egy úgynevezett erőtérfüggvény segítségével érik el. Tehát egy olyan függvény meghatározása a cél, ami egyrészről nem engedi, hogy elveszítsék egymást, s kommunikációs távolságon kívülre kerüljenek. Mindemellett azzal a tulajdonsággal is rendelkezik, hogy az energiaminimum-pontja a láncba szerveződést segíti elő. Ennek a meghatározásához a függvény koordinátarendszerét úgy rögzítjük, hogy az relatív legyen a vezető robothoz képest (így az az origóba kerül), valamint a keresési irányt rögzíthetjük, hogy az az x tengellyel párhuzamosan a pozitív irányba történik.

Amit keresünk az egy $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$ függvény. A kitűzött cél egy olyan irányítási rendszer leírása, mely feltételezi az ismert jelenlétét az origóban, és ez alapján a $(0, t)$-be küldi a jelenlegi pontot iterációs lépések folytán. Ott egy adott mértékű előrehaladást feltételezve próbálja meg előre követni a lépesben úgyszintén elmozduló origóban álló kalauzt.

Az 6. ábrán látható a kialakított erőtér, mely az origóban elhelyezkedő vezetőhöz képest adja meg az elmozdulás irányát és nagyságát. A s körön kívüli pontokban az erőtér 1 hosszt ír elő a középpont irányába, ezért amennyiben a networker vezetője nem léphet el nagyobb sebességgel mint alanyunk, abban az esetben biztosan köze-

lebb kerül hozzá a következő körben és helyre tudja állítani a formációt. A körök sugarai az egy lépés alatt maximálisan megtehető sebességtől függenek: $z = \frac{v_{max}}{R}$ esetén: az s kör sugara $R - z$, valamint az l kör sugara pedig $R - 2z$.



6. ábra. *Erőtér*

A követhetőség miatt legyen $l = R - 2z$ $(= d_s)$, valamint $s = 1 - z$.

Az erőtér alkotóelemei:

1. Szükség van arra, hogy a minimum a $(0, l)$-ben legyen, azaz a csúcs „odataláljon" ha esetleg teljesen máshol lenne:

$$u(x, y) = \frac{(-x, -y + l)}{|(-x, -y + l)|} \tag{3.4}$$

2. Amennyiben a csúcs a $(0, l)$-ben van, akkor arra egy jobbra irányuló körmozgásnak kell hatnia, hogy a vele párhuzamosan elmozduló vezető pozícióját kövesse. A $(0, d_s)$ pontban a jobbra irányuló körmozgást leíró egyenletek:

$$t(x, y) = \frac{(y, -x)}{|(x, y)|} \tag{3.5}$$

3. A kalauz egység sugarú környezetében maradásához szükséges lesz:

$$b(x, y) = \frac{(-x, -y)}{|(x, y)|} \tag{3.6}$$

4. Csak akkor kell védekezni a kalauz kommunikációs távolsága ellen, amennyiben az s körön kívülre került:

$$r = \max\left(\frac{|(x,y)| - s}{1-s}, 0\right) \tag{3.7}$$

5. Ötvözni kell majd a körmozgást elősegítő és a helyreküldő függvényeket, mivel a körmozgásra csak a $(0,l)$ pont z sugarú környezetében van szükség, ezért:

$$m(x,y) = \min\left(\frac{|(x,y-l)|}{z}, 1\right) \tag{3.8}$$

A fentiek felhasználásával az f függvény konstrukciója:

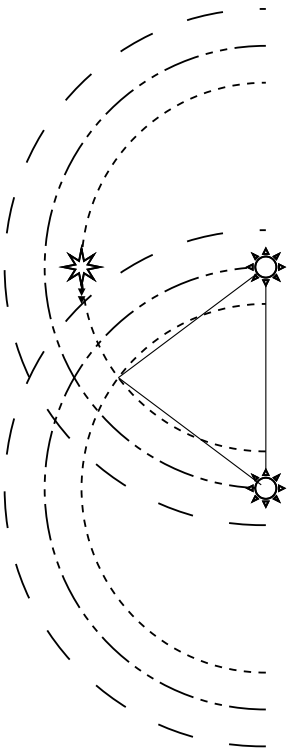
$$f = \underbrace{r * b}_{1} + \underbrace{(1-r)}_{2}\left(\underbrace{m * u}_{3} + \underbrace{(1-m) * t}_{4}\right) \tag{3.9}$$

1. az b függvény normalizált, az origóba mutató vektorokat tartalmaz, ezt megszorozva az origótól t távolságik 0 majd az egység távolsági 1-et elérő r függvénnyel az eredmény mindig a s sugarú körre kerül.

2. a fennmaradó energiát már szabadon használhatjuk a körmozgás kialakításához

3. amennyiben nem a kívánt pozícióban tartózkodik a helyére kell „küldeni”.

4. ha már a megfelelő pozíció közelében van, akkor számítani kell arra, hogy a kalauz elmozdul az x tengely irányába



7. ábra. *Két torony kerésési törésszöge*

Amennyiben a felderítőlánc vezetője, mely közvetlenül egy toronytól kapja az iránymutatást új toronyról szerez tudomást, mely közelebb van a meglévőnél, azt leváltja. De ekkor a keresési irány hirtelen nagyon megváltozik.

Egész pontosan a probléma a következő: Két torony legyen egymástól $\boldsymbol{d_t}$ távolságra és vezessünk végig egy felderítőt $\boldsymbol{d_s}$ távolságra a tornyoktól (7. ábra). Ekkor ha éppen tornyot vált a kereső, akkor a keresési irány változásának szöge:

$$\cos\gamma = \frac{2d_s^2 - d_t^2}{2d_s^2} = 1 - \frac{d_r^2}{2d_s^2} \tag{3.10}$$

Ez a jelenség a tornyoktól távolabb álló felderítőkön látszik erősebben; a lánc meg is szakadhatna, amikor túl hirtelen változik meg az irány - de ez az irány frissítésekor annak elfordulási szögének maximumának megkötésével beleszámítható a $\boldsymbol{d_s}$ paraméter meghatározásába.

A keresés folyamán a lánc mérete alapján egy csúcs maximum $\frac{k}{l}$ sebességgel mozoghat, ahol $\boldsymbol{l}$ a teljes lánc hossza, $\boldsymbol{k}$ pedig a szóban forgó robot távolsága a kiindulási ponttól. A fenti mozgató függvénynek ez alapján az alábbiakat kell teljesítenie: tegyük fel a legrosszabbat, a keresőnk kicsúszott a vezetőtől mért $\mathbf{1}$ sugarú körre. Vegyük ezt az elemet a $\boldsymbol{k}$-nak az $\boldsymbol{l}$ hosszú láncban. És tegyük fel azt is, hogy a $\boldsymbol{k-1}$**.** csúcsnak a vezetője is kicsúszott, és a lehető legrosszabb irányba lép tovább. Azonban a $\boldsymbol{k-1}$. csúcs csak maximálisan $\frac{k-1}{l}$ sebességgel mozdulhat el, ezért ha az $\boldsymbol{f}$ függvény $\mathbf{1}$ sugarú körén a vektorok tökéletesen befele mutatnak, akkor soha nem fordulhat elő egy csúcs keresőtől történő leszakadása.

Ennek a tulajdonságnak az ellenőrzésére használható az alábbi módszer: Vegyük a következő függvényt:

$$\boldsymbol{w_x(x,y) = -x} \tag{3.11}$$

$$\boldsymbol{w_y(x,y) = -y} \tag{3.12}$$

Amennyiben $\boldsymbol{\forall(x,y) \in \mathbb{R} : x^2 + y^2 \leq 1 : f(x,y) + \frac{w(x,y)}{|w(x,y)|} > 0}$, akkor ez a tulajdonság teljesül, így egy megengedett állapotból kiindulva soha nem veszhet el

egy elem sem.

3.3.3. Lánchossz

A keresés során az egyes ágensek (nagyon) aktív kommunikációt folytatnak, de a megfelelő sebességgel történő haladáshoz és egymás el nem hagyásához ismerniük kell saját pozíciójukat a láncban, valamint annak hosszát.

Ezért a $\overline{scan\_update}$ üzenete tartalmazza a fordulási ponttól való távolságot (szomszédban mérve), valamint hasonlóan a lánc másik végétől a $\overline{scan\_update\_req}$ csomagok is tartalmazzák a külsőbb íveken álló robotok számát. Ezen két információ összege a lánc minden tagja számára elérhetővé teszi a lánc jelenlegi hosszát, valamint az elfoglalt pozícióját is megtudja - melyek segítségével meghatározható a maximálisan megengedett sebesség.

3.3.4. Pásztázási fázis nélkül

A pásztázók bemutatásakor azok szükségességét nem tisztáztam.

Amennyiben kimaradna a kezdeti pásztázási fázis, akkor egy olyan helyzet állna elő, melyben az egyes egységek mindegyik másikat látják, és így a fent leírt lánckialakításhoz használt véletlenen alapuló szerveződés gyenge hatékonysággal kezd el rendet tenni. Ennek az oka, hogy az $\boldsymbol{i}$ sorozatszámú egység minden egyes körben $\frac{\boldsymbol{i-1}}{\boldsymbol{n}}$ valószínűséggel szerezhet vezetőt, ami a **2**-es sorozatszámú robotot tekintve $\frac{\mathbf{1}}{\boldsymbol{n}}$. A láncok ugyan kialakulnak, de nem találnak igazi tornyot, és így sokáig nem végeznek hasznos tevékenységet.

Az algoritmus pszeudo kódjában használt jelölések

$\boldsymbol{s(x)}$	az $\boldsymbol{x}$ robot állapotát adja meg $\boldsymbol{\{\widehat{recon}, \widehat{tower}, \widehat{sweep}\}}$
$\boldsymbol{p(x)}$	az $\boldsymbol{x}$ robot pozícióját adja meg
$\boldsymbol{v(x)}$	az $\boldsymbol{x}$ robot által megadott irányvektor(2d irányvektor)
$\boldsymbol{\Gamma(x)}$	az $\boldsymbol{x}$ robot sorozatszáma
$\boldsymbol{k}$	a jelenlegi robot ismerőse kezdetben: $\perp$
$\boldsymbol{r}$	az üzenet küldője
$\boldsymbol{m}$	maga az üzenet

***recon_step*()**

// üzenetek feldolgozása

$\boldsymbol{chain\_len\_u = 0}$

while (type,m,r)=fetch_message()

switch(type)

$\overline{interlock}$:

elküld egy $\overline{interlock}$ -t $\boldsymbol{k}$-nak.

$\widehat{tower}$ állapotba áll át

$\overline{scan\_info}$:

$\boldsymbol{if(s(r) = \widehat{tower} \wedge ((k = \perp) \vee (k \neq \perp \wedge tower\_connection \wedge}$

$\boldsymbol{\wedge d(k) > d(r)))\vee}$

$\boldsymbol{\vee(s(r) = \widehat{recon}) \wedge \Gamma(r) < \Gamma \wedge \Gamma(k) < \Gamma(r))}$

$\boldsymbol{tower\_connection = (s(r) = \widehat{tower})}$

$\boldsymbol{k = r}$

$\overline{scan\_update}$:

irányvektor frissítés az $\boldsymbol{m}$-ben találhatóval, maximum $\boldsymbol{\alpha}$ szöggel fordulhat el

$\boldsymbol{chain\_length\_d = m.chain\_len + 1}$

$\overline{scan\_update\_req}$:

// ebből kaphat akár többet is, mindre válaszolnia kell

$\boldsymbol{chain\_len\_u = MAX(chain\_len\_u, m.chain\_len + 1)}$

$\boldsymbol{d = \max(d', d)}$

visszaküld egy $\overline{scan\_update}$ -t

// lépés

$\boldsymbol{if(phase = RECON\_SCAN\_INFO)}$

véletlenszerűen kiválaszt egy másik robotot($\neq \boldsymbol{k}$) és

elküld neki egy $\overline{scan\_info}$ -t

$\boldsymbol{if(k \neq \perp)}$

kiszámítja:

$\underline{\boldsymbol{m}} = \boldsymbol{f(\langle p - p(k), (v_y, -v_x)\rangle, \langle p - p(k), (v_x, v_y)\rangle)}$

elmozdul $\underline{\boldsymbol{m}} * \boldsymbol{v_{max}} \frac{\boldsymbol{chain\_len\_d+1}}{\boldsymbol{chain\_len\_u+chain\_len\_d+1}}$-vel

elküld egy $\overline{scan\_update\_req}$ -t $\boldsymbol{k}$-nak.

3.4. Gráfcsúcsok - $\widehat{tower}$

Azon felderítők, melyek egy új értékes kliens pontot találnak, tornyokká válnak. Ebben az állapotban az ágens feladata néhány szomszédjával történő kapcsolattartás (oly módon, hogy azoknak ne kerüljön hatósugarán kívülre), valamint az arra tévedő felderítőknek való útmutatás.

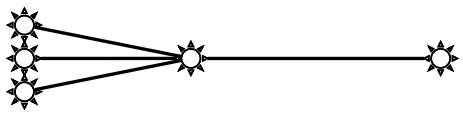
Minden torony állapotban lévő robot rendelkezik szomszédai halmazával ($\boldsymbol{K}$). Ez a halmaz mindig tartalmaz legalább egy elemet - amennyiben pontosan egy elemet tartalmaz, akkor az ágens megpróbál leválni, és felderítőként tovább folytatni a kutatást egy új kliens pont után.

Ha egy kereső új csúcsot talál, akkor ennek hatására egy $\overline{interlock}$ üzenet végigfut a keresőláncon, amíg el nem éri a már meglévő gráf peremét. Habár ezután az új kliens pont hozzá van kapcsolva a már létező hálózathoz, de könnyen megeshet, hogy a keresők elég ügyetlenül találták meg (például a kelleténél sűrűbben álltak fel). Ezért mindenképp jó lenne kihasználni a tornyok esetében is azok mobilitási képességét, és megpróbálni csökkenteni a hálózat fenntartásához szükséges ágensek számát, és így egyúttal növelni a keresők számát.

Olyan módszerre van szükség, mely segítségével a tornyok elmozdulhatnak, de meg kell tudni tartaniuk a kapcsolataikat a szomszédos tornyokkal. Az irodalomban több helyen találkoztam a rugóerőkkel ([16],[8]), mely egy nagyon egyszerű és jól használható módszer.

3.4.1. Rugóerők

Közelebbi vizsgálódás során néha anomáliák léptek fel, mely annak volt köszönhető, hogy rugóerők alkalmazása esetén a 8. ábrán látható problémával kell szembesülni. Ugyanis a rugóerők képesek egy adott irányba túlságosan is nagy erőt kifejteni, amennyiben több pont csoportosul egy másikkal szemben.



8. ábra. *A rugóerőkkel való probléma szemléltetése*

A helyettesítésére egy olyan módszert kerestem, ami megtartja a rugóerő pozitív tulajdonságait, de kevésbé érzékeny az 8. ábrán látható csoportosulásra. Egy olyan középértéket kellett keresni, ami a rugóerőhöz hasonlóan valamilyen energiaminimum-pont felé tendál, viszont nem annyira érzékeny arra, hogy a pontok milyen formában helyezkednek el. Ez a probléma hasonlít egy pontcsoport legjobb belső pontjának meghatározásához.

Feladat:

Adott $\boldsymbol{n}$ pont a síkon $\boldsymbol{p_i} \in \mathbb{R}^2$, cél meghatározni egy olyan pontot, mely közel azonos távolságra van a pontoktól azok csoportosulásától függetlenül.

Megjegyzés:

A súlypont elég közel áll a probléma megoldásához, csak a pontcsoportot kell jól megválasztani.

Az alapötlet (***gavg***) az, hogy úgy szeretném súlyozni, hogy a pontok száma ne számítson annyira. Ezért a pontok által kifeszített sokszög éleinek összes pontja alapján állapítom meg a középpontot.

gavg

Vegyük a pontokat valamilyen körüljárási irány szerint rendezve, például az átlagolt pontból nézve (ez az amit megadnak a **0**-ban eltűnő rugóerők), ezután konstruálhatunk olyan $\boldsymbol{f_i(x)}$ függvényeket, melyek az $\boldsymbol{i}$**.** él pontjait végiglátogatják. $\boldsymbol{f_i(\lambda) = x_{i-1} * \lambda + (1 - \lambda) * x_i}$. Ahova szeretnénk eljutni az a következő: átlagoljuk az alakzatunk éleinek pontjainak vektorait, ezt úgy tehetjük meg, hogy az $\boldsymbol{f}$ függvényünk minden pontját megsúlyozzuk az ottani deriválttal.

$$\frac{\int_0^1 \|\boldsymbol{f'}\| \boldsymbol{f}}{\int_0^1 \|\boldsymbol{f'}\|} \tag{3.13}$$

Némi fejtegetés után a következő egyszerűsített képlet adódik:

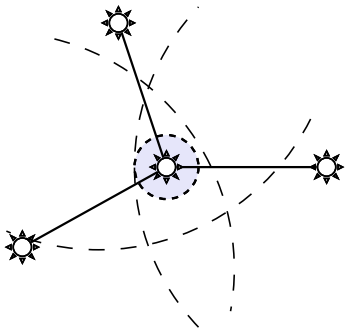
$$\boldsymbol{gavg(P)} = \frac{\sum_{i=1}^{n} \vec{\boldsymbol{x_i}} * (||\boldsymbol{x_{i-1}} - \boldsymbol{x_i}||_2 + ||\boldsymbol{x_i} - \boldsymbol{x_{i+1}}||_2)}{\boldsymbol{2} \sum_1^n ||\boldsymbol{x_i} - \boldsymbol{x_{i-1}}||_2} \tag{3.14}$$

Megjegyzés:

Az egyszerűség kedvéért az $\boldsymbol{x_0}$ pont megegyezik az $\boldsymbol{x_n}$-el.

Megjegyzés:

A $\boldsymbol{P = \{x_1, x_2 ... x_n\}}$ ponthalmaz, például az óramutató járása szerint rendezve.



9. ábra. *Szabad mozgási tér (fdist)*

Függetlenül az alkalmazott középpont meghatározási módszertől, az adott középpont még nem biztosít tervezett távolságot a kiindulópontoktól. Ezért szükség van a következő segítő függvényre:

A szabadtávolság (fdist)

Azt adja meg, hogy mekkora az a távolság, amit az adott pont még a távolsági korlátok megsértése nélkül elmozdulhat (9. ábra):

$$\boldsymbol{fdist(q, P, r) = \max(0, \frac{r - \max_{p \in P} d(p, q)}{r})} \tag{3.15}$$

A fenti két ötlet segítségével kifeszítik a gráfot az ágensek és a távolságokat egyenletesen elosztják, úgy ahogy rugóerők esetében történt volna.

Viszont ennek mellékhatása, hogy két ágens közötti távolság nagyon kicsivé is válhat, ezért szükséges egy másik módszer is, melyek felváltva alkalmazásával a tervezett távolságra kerülnek a csúcsok egymástól.

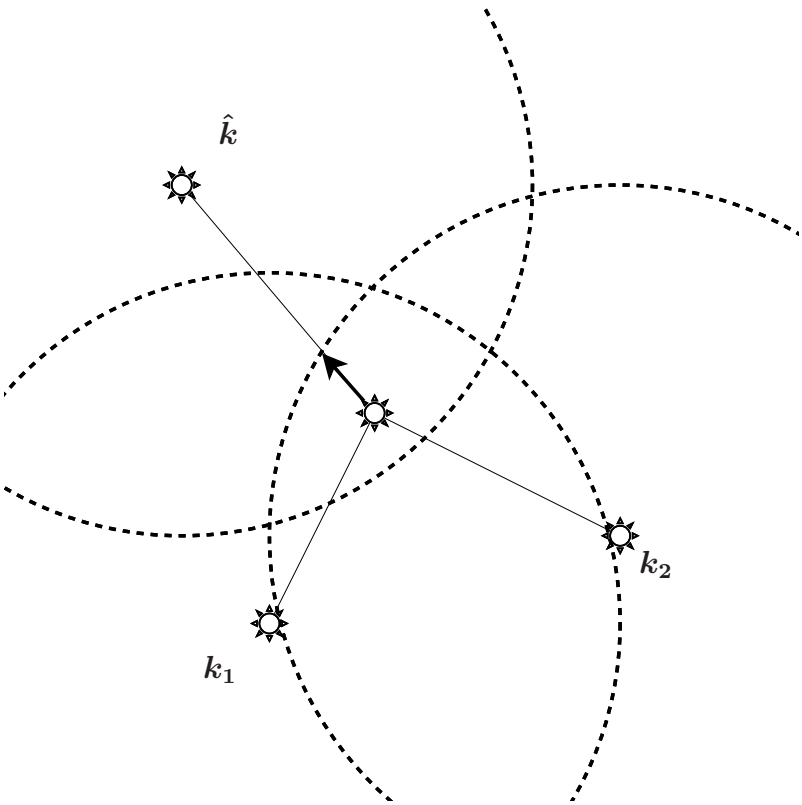
Optimalizációs fázisok

A gráf kifeszítése két alapvető fázisból áll:

1. A tornyok az előbb bemutatott rugóerőszerű dolgot felhasználva megkeresik az energiaminimum-szintjüket és egyenletes távolságra kerülnek egymástól.

2. A gráf mélységméréséből ismertté válik az a szomszéd, mely leginkább a gráf

belseje fele van. Amennyiben ehhez a csúcshoz közelebb kerül az ágens, abban az esetben a hálózat széleit feszíti ki.



10. ábra. A $\hat{k}$ irányába való ellépés

A 10. ábrán látható egy torony, melynek a szomszédai $\hat{k}, k_1$ és k_2. A mélységi mérésből ismert, hogy a $\hat{k}$ szomszédján át érhető el a hálózat nagyobb része. Az ágens ebben az esetben a $\hat{k}, k_1$ és k_2 által megengedett maximális elmozulási határok között a $\hat{k}$ szomszéd irányába maximálisan lép el, így rákényszeríti a k_1 és k_2 csúcsokat, hogy mégjobban megfeszítsék saját szomszédaikat. Sok lépésen keresztül alkalmazva a módszert minden csúcs a tervezett távolságba kerül egymástól.

A gráf optimalizálásának második fázisában minden torony üríti az optimalizáló halmazt($\boldsymbol{O}$). Akik kommunikációs pont mellett állnak, felveszik azt az $\boldsymbol{O}$ halmazba, és elküldik minden szomszédjuknak a mélységüket. Innentől minden csúcs hasonlóként tesz, de bevárja, hogy $\boldsymbol{d(v) - 1}$ darab optimalizációs csomagot kapjon, mielőtt továbbküldené azt, meghatározva ezzel a fa pontbeli átmérőjét.

3.4.2. A hálózatot reprezentáló gráf

A torony szerepet betöltő networkerek egy fát próbálnak meg kifeszíteni a kliens-pontok közé. Ahhoz, hogy a tornyok képesek legyenek csökkenteni a hálózat fenntartásához szükséges robotok számát, ezt a gráfot kell megvizsgálniuk úgy, hogy az egyes networkerek alapvető tulajdonságait megismerhessék ennek a gráfnak.

Átmérőmérés

Nagyon sokat elárul egy ágens számára, ha ismeri, hogy az egyes szomszédai mekkora átmérőjű részét képezik a teljes gráfnak. Ezen információ birtokában képesek lesznek megfeszíteni a gráfot, és ezáltal csökkenteni annak fenntartásához szükséges ágensek számát.

Definíció

Gráf átmérője: egy gráf átmérője $\boldsymbol{k}$, amennyiben bármely két pontja között létezik egy legfeljebb $\boldsymbol{k}$ hosszú út.

A gráf átmérőjét a következő párhuzamos algoritmus([1]) segítségével mérhetjük le($\boldsymbol{K}$ a szomszédok halmaza, $\boldsymbol{O}$ egy kezdetben üres halmaz, valamint $\boldsymbol{d} = \mathbf{0}$)

1. amennyiben $|\boldsymbol{O}| = |\boldsymbol{K}| - \mathbf{1}$ elküldi $\boldsymbol{d}$-t a $\boldsymbol{t} = \boldsymbol{O} \backslash \boldsymbol{K}$ elemnek és a $\mathbf{3}$ fázisba lép

2. várja, hogy valamelyik szomszédjától $\boldsymbol{s}$ egy $\boldsymbol{d'}$ mérést kapjon
   $\boldsymbol{d} = \mathbf{max}(\boldsymbol{d}, \boldsymbol{d'} + \mathbf{1})$
   $\boldsymbol{O} = \boldsymbol{O} \bigcup \{\boldsymbol{s}\}$
   ugrás az $\mathbf{1}$ pontra

3. várja, hogy az egyetlen szomszédjától ($\boldsymbol{t} = \boldsymbol{O} \backslash \boldsymbol{K}$), akinek elküldte a saját mélységét, üzenetet kapjon $\boldsymbol{d'}$ méréssel
   ekkor a gráf átmérője: $\boldsymbol{d'} + \mathbf{1} + \boldsymbol{d}$
   Ahhoz, hogy mindenki rendelkezzen ezzel az információval, még tovább kell küldenie a $\boldsymbol{d'} + \mathbf{1}$-et minden az $\boldsymbol{O}$ halmazban felsorolt szomszédjának.

A tornyok algoritmusa tartalmazza a fenti módszert, valamint az $\boldsymbol{O}$ halmazt még arra is felhasználja, hogy annak ismeretében tisztában van vele az ágens, hogy a gráf nagyrésze a $\boldsymbol{t}$ csúcs irányában van, ezért ahhoz az elemhez megy közelebb - így a hálozat peremvidékén maximális távolságba kerülhetnek az ágensek.

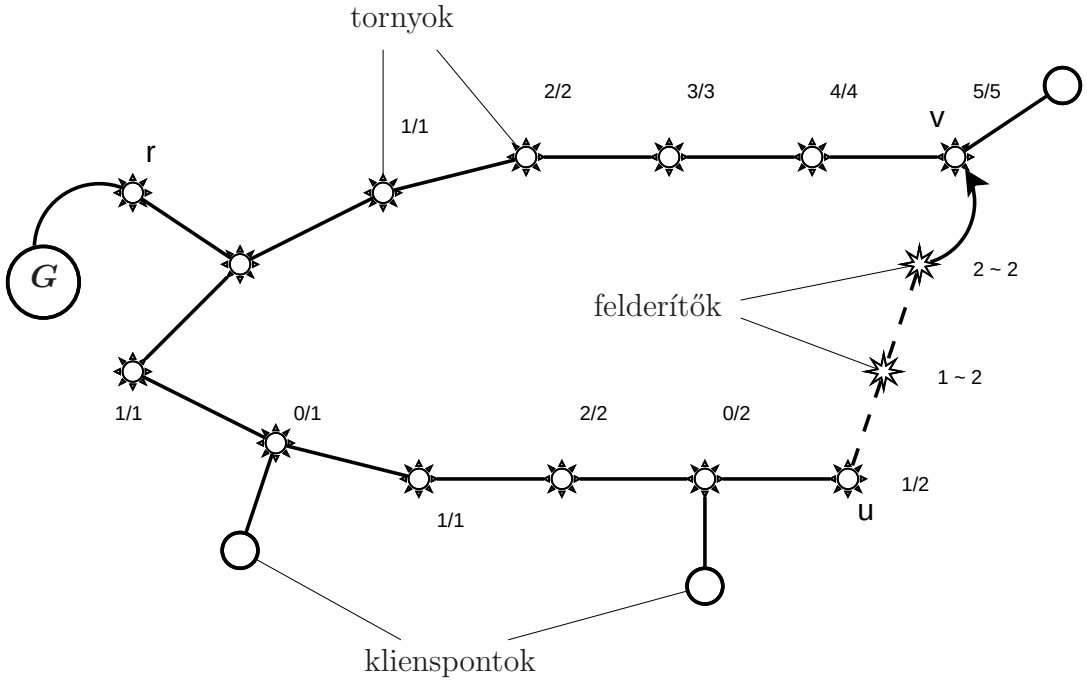
3.4.3. Több forrású mélységmérés

A gráf átmérőjének ismerete sokat segít, de sokszor előfordulhatnak feleslegesen kialakult bekötőutak, melyek helyett sokkal rövidebbekre is lehetőség lenne.

Az ilyen jellegű hibákon javítani lehet, ha létezik egy olyan rövidebb még fel nem használt él, mely bevételével egy másik él válik a legnehezebbé. De ehhez szükséges valamilyen módon súlyozni: a két szomszéddal rendelkező tornyok igazából nem tekinthetőek belső pontoknak, sőt leginkább súlyok, mivel ha kevesebbel is megoldható, akkor azt valamilyen módon figyelembe kell venni.

Amennyiben véletlenszerűen választanánk egy csúcsot, és attól mérnénk minden csúcs távolságát, akkor lehetséges javítani az ilyen jellegű hibákon - azonban ha a gráf már viszonylag nagy, akkor ezen csúcs kiválasztása csak a környezetében lévő hibákon tud segíteni a távolabbiakat nem fedi fel. Ezért egyszerre több csúcsot választok. A gráfot tulajdonképpen egyszerre több színnel színezem ki, ezáltal megnövelve az esélyét annak, hogy ezen hibák felfedhetőek legyenek.

Válasszunk néhány csúcsot véletlenszerűen, és mérjük meg minden csúcs távolságát ezektől a csúcsoktól, valamint tételezzük fel, hogy létezik egy olyan összekapcsolható javítóél, amivel könnyíthetünk a gráfon.



11. ábra. *Mélységi javítás*

A 11. ábrán látható egy olyan mélységmérés, mely esetében a távolságot az $\boldsymbol{r}$ csúcstól mértük és egy javításra van lehetőségünk, ugyanis a felső ág **5** hosszú,

melyet elérhetünk egy **3** hosszúval is. Az ábrán az egyes ágensek mellett feltüntetett értékpárok az $\boldsymbol{l_c/l_m}$

Minden $\boldsymbol{t_d}$ időnként egy előre megadott kiválasztó funkció meghatározza a robotok egy csoportját, akiktől távolságot mér a többi. A mérés elvégzése közben az alábbiakat közlik egymással:

- $\boldsymbol{l_c}$: jelenlegi hossz
  Láncok hosszát mérjük, és keressük az eddig előforduló leghosszabbat. Ezt az értéket minden **2** fokszámú csúcs növeli eggyel, a **3** fokszámúak pedig **0**-ra állítják

- $\boldsymbol{l_m}$: eddigi leghosszabb lánc, amit találtak
  (a leghosszabb lánc legutolsó elemein $\boldsymbol{l_c = l_m}$)

- $\boldsymbol{l_r}$: mélység a kezdeményező csúcstól
  szükséges megszabni az irányát a javításnak, a forráscsúcstól közelebbivel történhet csak javítás

- $\boldsymbol{i_r}$: a mélységmérést kezdeményező pont azonosítója
  erre azért van szükség, hogy a különböző helyekről indított mérések ne keveredhessenek össze. Tulajdonképpen ez a színe a mérésnek

Folyamatában ez a következőképpen zajlik, ha ilyen mérést kap egy csúcs, akkor az alábbiak szerint cselekszik:

- ha már kapott korábban távolságmérő csomagot, akkor figyelmen kívül hagyja

- ha a jelenlegi csúcs fokszáma **2** - vagyis egy egyszerű összekötő csúcs -, akkor: $\boldsymbol{l_c = l_c + 1}$, s ha ez meghaladja az $\boldsymbol{l_m}$-et is, akkor azt is növeli

- beállítja mélységi szülőnek a küldőt, és továbbküldi minden szomszédjának, kivéve a szülőt

Tehát, ha létezik olyan él, amit a gráfba behúzva és törölve a kialakult kör legnehezebb élét egy könnyebbet kapunk, akkor a 11. ábrán látható helyzetben:

$$\boldsymbol{l_m^u + l_{chain} < l_m^v} \tag{3.16}$$

feltétel teljesül, így javítás végezhető.

Definíció

Steiner-fa: Adott egy $\boldsymbol{G = (V, E)}$ irányítatlan, összefüggő gráf, és a terminál pontjainak egy $\boldsymbol{T}$ halmaza ($\boldsymbol{V}$ része), továbbá minden élhez hozzárendelünk egy költséget. Keressünk egy minimális költségű fát, amely az összes $\boldsymbol{T}$-beli pontot tartalmazza.

Definíció

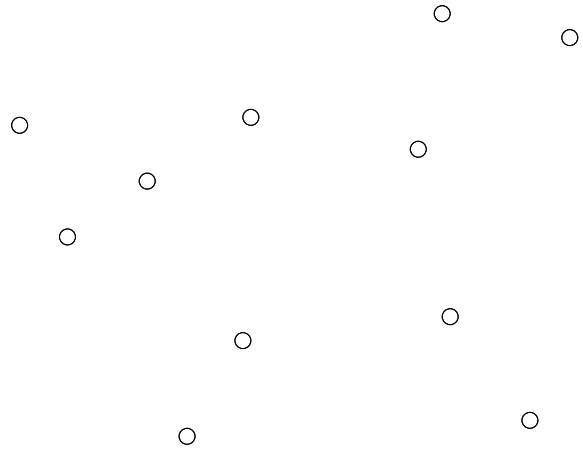
euklideszi Steiner-fa: $\boldsymbol{V}$ a sík pontjait tartalmazza

Definíció

maximum $\boldsymbol{n - 2}$ darab Steiner-pontot tartalmaz egy $\boldsymbol{n}$ terminálal rendelkező fa

Definíció

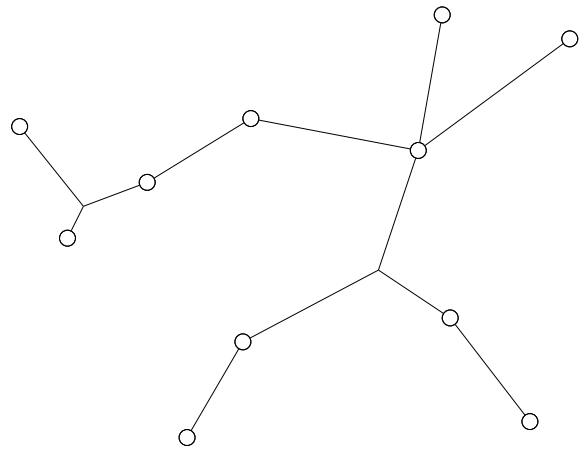
minden Steiner-pont foka **3** és az egyes élek **120** fokban indulnak ki belőle.



12. ábra. *Terminálisok (**T**)*

Legyen $\boldsymbol{T}$ a terminális pontok halmaza(12. ábra), melyeket szeretnénk lefedni egy euklideszi Steiner-fával.

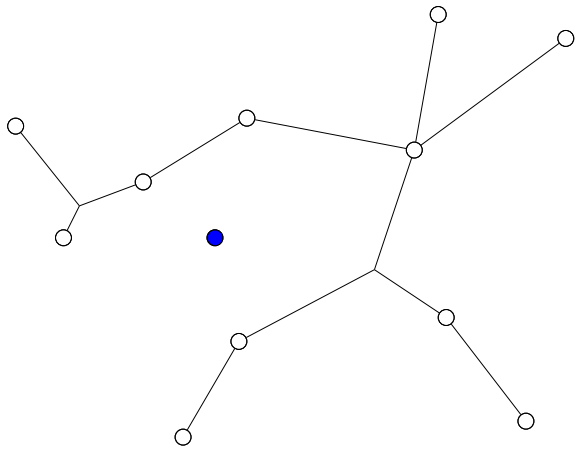
Ezen pontokra valamilyen módszer felhasználásával meghatározhatjuk a minimális Steiner-fát ($\boldsymbol{S_T}$), mely lefedi ezen pontokat (13. ábra).



13. ábra. *A $\boldsymbol{T}$-t lefedő $\boldsymbol{S_T}$ Steiner-fa*

Mivel a felvetett probléma esetében ez a $\boldsymbol{T}$ terminálishalmaz akár meg is változhat idővel, ezért érdemes megvizsgálni azt, hogy mi történik, ha már ismert egy $\boldsymbol{T}$ terminálishalmaz Steiner-fája, és ez esetlegesen hogyan viszonyul egy olyan $\boldsymbol{T'}$ terminálisokat lefedő $\boldsymbol{S_{T'}}$ fához, mely az eredeti $\boldsymbol{T}$ terminálishalmaznak egy ponttal való bővítése.

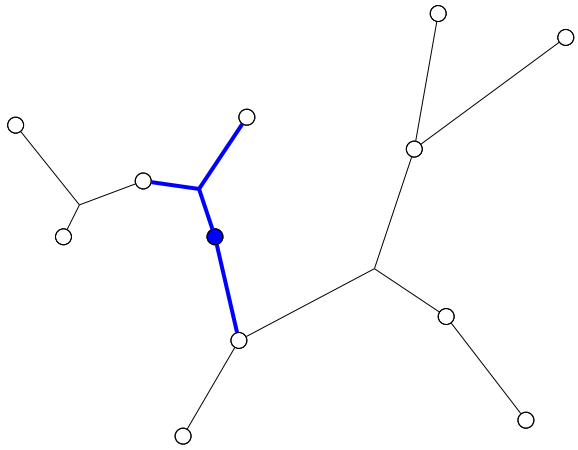
Ennek a szemléltetéséhez a korábbi fát bővítem egy új ponttal (14. ábra).



14. ábra. *Pont hozzáadása*

Amennyiben az így létrejött $\boldsymbol{T'}$ terminálishalmazra újra megkeressük az optimális Steiner-fát, abban az esetben a $\boldsymbol{T'}$-höz tartozó $\boldsymbol{S_{T'}}$ fa erősen eltérhet a $\boldsymbol{T}$ terminálisc soporthoz tartozó $\boldsymbol{S_T}$ fától.

Ezt illusztrálja a 15. ábrán látható kék vonalak, melyek az új fa élei.



15. ábra. *A $\boldsymbol{T'}$-t lefedő $\boldsymbol{S'_T}$ Steiner-fa*

Megjegyzés:

Az optimális Steiner-fa ilyen módon való keresése igényli a teljes $\boldsymbol{T}$ halmaz ismeretét. Valamint további hátránya, hogy a terminális halmaz esetleges változása esetén a fa erősen megváltozhat.

Egy olyan módszer segítségével lenne célszerű egy ilyen hálózatot felépíteni, mely megengedi az iteratív bővítést, és fokozatosan képes közelíteni a Steiner-fa megoldását, anélkül hogy a fa megszakadna.

A Steiner-fa probléma megoldása megközelíthető heurisztikus algoritmusok segítségével, ezek közül kiemelném a Steiner Insertion Heuristic nevű [3] algoritmust, mely egy picit módosítva egy Steiner-fa bővítővé alakítható át.

steiner_insertion(T,E)

1. legkönnyebb feszítőfa megkeresése a $\boldsymbol{T}$-n
2. minden élre, mely terminálisokat köt össze $(\boldsymbol{t_x}, \boldsymbol{t_y})$-ra:
   (a) egy olyan $\boldsymbol{q}$ csúcs keresése, mely vagy terminális vagy Steiner-pont, valamint létezik a $(\boldsymbol{q}, \boldsymbol{t_x})$ él, és ezen élek közül a legkisebb szöget zárja be a $(\boldsymbol{t_x}, \boldsymbol{t_y})$ éllel.
   (b) amennyiben a $(\boldsymbol{q}, \boldsymbol{t_x}), (\boldsymbol{t_x}, \boldsymbol{t_y})$ élek szöge kisebb mint **120** fok
      i. új steiner-pont felvétele ($\boldsymbol{s}$) a $\boldsymbol{t_x}$ pozíciójába
      ii. élek eltávolítása:
         - $(\boldsymbol{q}, \boldsymbol{t_x})$
         - $(\boldsymbol{t_x}, \boldsymbol{t_y})$
      iii. új élek felvétele:
         - $(\boldsymbol{s}, \boldsymbol{q})$
         - $(\boldsymbol{s}, \boldsymbol{t_x})$
         - $(\boldsymbol{s}, \boldsymbol{t_y})$
   (c) lokális optimalizáció

Alapesetben ez az algoritmus egy közelítést ad a $\boldsymbol{T}$ ponthalmaz Steiner-fával való lefedésére. Amennyiben úgy módosítjuk hogy csak bővítsen egy korábbi fát, akkor azzal már úgy közelíthetjük az új fát hogy a régit részben megtartjuk.

Egy nagyon nagy előnye ennek a módszernek hogy amennyiben úgy tekintünk az egyes csúcsokra mint entitások, s az éleket mint szomszédsági kapcsolatokra, akkor a fenti algoritmus bizonyos formában lokális kommunikációkra alapozva kivitelezhető az élek mentén.

Definíció

Ha egy $\boldsymbol{G = (V, E)}$ euklideszi Steiner-fa egy $\boldsymbol{T \subset V}$-ra tekintve $\boldsymbol{\iff \nexists u, v \in V}$ csúcsok, melyek közé élet behúzva a keletkező körnek az $\boldsymbol{(u, v)}$ éle könnyebb lenne a kör bármely élénél.

Megjegyzés:

A többforrású mélységmérés segítségével a networkers algoritmus ilyen élek ellenőrzésével igyekszik csökkenteni a kialakított gráf súlyát.

3.4.4. Prioritás

Az ágenseknek szüksége van egyfajta időről időre történő kitüntetésre, például egy kapcsolat megszüntetése és annak helyettesítése egy másikkal egy ágens számára megengedett kell legyen, de amennyiben több ágens is egyszerre „gondolja" ezt, akkor az zűrzavart okozhat.

A prioritás bevezetésére csak lokálisan van szükség, ugyanis egy torony esetében elégséges, ha a szomszédaihoz mérten van prioritása.

Előnyös lenne, ha nagyon gyakran kerülne újra és újra prioritásba minden csúcs és a legjobb lenne, ha nem kellene hozzá kommunikáció. Ezért a sorozatszámok, valamint az eltelt idő segítségével kell kiszámolni egy logikai értéket.

A prioritás függvénnyel szemben azonban támasztanunk kell egy megkötést - nem lehet két szomszéd egyszerre prioritásban:

Definíció

Adott egy $\boldsymbol{G = (V, E)}$ gráf, és egy $\boldsymbol{\varphi : V \to \mathbb{L}}$. A $\boldsymbol{\varphi}$ egy megengedett priorizálása a csúcsoknak amennyiben:

$$\nexists \boldsymbol{(u, v) \in E : \varphi(v) \wedge \varphi(u)} \tag{3.17}$$

A feladat a következőként fogalmazható meg:

Feladat:

Adott egy tetszőleges gráf, melynek fokszámára ismerünk felső korlátot. Valamint adott $\boldsymbol{2^s}$ felső korlát a csúcsok számára és adottak a $\boldsymbol{\tau = \{\varphi_1, ..., \varphi_n\}}$ priorizálások, amennyiben $\boldsymbol{\forall u \in V : \exists i \in [1 : n] : \varphi_i(u)}$ teljesül tetszőleges $\boldsymbol{k}$ fokszám korlátos gráf esetén, melynek legfeljebb $\boldsymbol{2^s}$ csúcsa van, akkor $\boldsymbol{\tau}$ megoldása a gráf szabad prioritásának.

Címkézés

Vegyünk egy tetszőleges $\boldsymbol{k}$ fokszám korlátos egyszerű gráfot($\boldsymbol{G(V, E)}$), és egy olyan címkézését a csúcsoknak, mely minden csúcshoz különböző számokat rendel a $\mathbb{N}$ halmazból.

Irányítsuk meg a gráf éleit a címkézés szerint értelmezet '$<$' reláció segítségével.

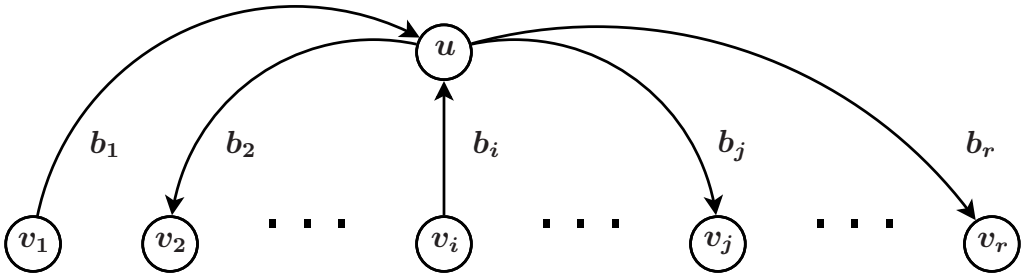
Értelmezzük a gráf $\boldsymbol{x}, \boldsymbol{y}$ csúcsaira a következő függvényt:

$$\boldsymbol{t(x,y) = \lfloor \log_2(s(x) \otimes s(y)) \rfloor} \tag{3.18}$$

Címkézzük fel az éleket a következő módon: az $\boldsymbol{(x,y) \in E}$ élhez rendeljük a $\boldsymbol{t(x,y)}$ függvény értékét: $\boldsymbol{b_{x,y} = t(x,y)}$.

A $\boldsymbol{t(x,y)}$ függvény megmondja hogy a címkézés hányadik bitjének negálása fordítaná meg az $\boldsymbol{x}$ és $\boldsymbol{y}$ csúcsok közötti '$<$' relációt.

Vegyük az így elkészített gráf egy $\boldsymbol{u}$ csúcsát és vizsgáljuk meg a szomszédait ($\boldsymbol{N(u) = \{v_1, v_2, .., v_r\}}$).



16. ábra. *Az $\boldsymbol{u}$ csúcs és szomszédai*

Az elkészített gráf és az élek címkézése látható a 16. ábrán.

Állítás:

Amennyiben $\boldsymbol{u}$ két különböző szomszédjával ($\boldsymbol{v_i, v_j}$) összekötő élekre ugyanazon ($\boldsymbol{b_i = b_j}$) címke került, akkor ezen két él nem alkothat utat a $\boldsymbol{v_i, v_j}$ csúcsok között.

Biz:

Az élek címkézése a legnagyobb ($\boldsymbol{z.}$) bitet jelöli, melynek negálása esetén a köztük fennálló '$<$' reláció megfordul. Az $\boldsymbol{u, v_i, v_j}$ csúcsokon kívül dobjuk el a gráf összes többi pontját. És a csúcsok címkézését szűkítsük le a $\boldsymbol{z.}$ bitre. Mivel az $\boldsymbol{u, v_i, v_j}$ csak ettől függött, ezért a köztük futó élek iránya és címkéje az eredetivel megegyező marad. A leszűkítés miatt viszont már csak két értéket tartalmaz a csúcsokhoz rendelhető címkék halmaza, de így biztosan nem alakulhat ki három hosszú irányított út.

Állítás:

Válasszunk egy (u, v_i) élet, és egy tetszőleges $r \in \mathbb{N}$ számot. Ekkor, az $s'(x) = s(x) \otimes 2^r$ újracímkézés esetén az (u, v_i)-él irányítása akkor és csak akkor fordul meg ha r megegyezik az él címkéjével z, vagyis ha $z = r$ teljesül.

Biz:

- $r > z$ esetében az $s(u)$ és $s(v_i)$ számok $r.$ bitje meg kellett hogy egyezzen, ezért a negálása nem változtat a köztük fennálló reláción.
- $r < z$ esetében mivel a $z.$ bit nagyobb helyiértéket foglal el, ezért a reláció erre érzéketlen marad.
- $r = z$ esetében megfordul a reláció.

Következmény

Készítsünk el egy számot az u-ból ki/befutó élek címkézése és iránya alapján. A kifutó élekre írt címkéket gyűjtsük a G halmazba, a befutókat pedig az L halmazba. Vegyünk egy olyan számot(w) melynek G halmazba tartozó bitjei **0**-ák, és minden L halmazba tartozó bitje **1**. Mivel $G \bigcap L = \emptyset$, ezért ilyen szám létezik.

Vegyük az $s'(x) = s(x) \otimes w$ címkézést, ebben az esetben az u csúcsnak csak kifutó élei lesznek.

Egy olyan átcímkézőre van szükség, mely képes véletlenszerűen csúcsokat kitüntetni. A fentiek alapján minden csúcs rendelkezik valahány bitre való előírással annak kitüntetéséhez - viszont a többi bitre nincs kikötése.

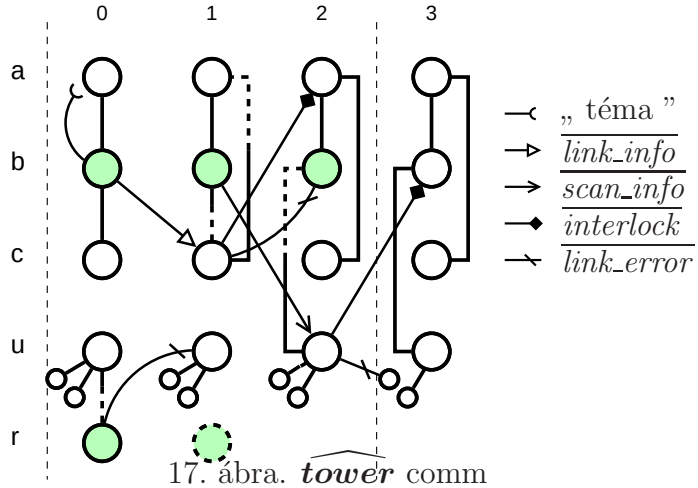
Megoldás: véletlen

A fenti feltételeket egy véletlenszámgenerátor teljesíti, ugyanis abban tetszőlegesen távol álló bitek mintavételezése esetén egyenletes eloszlást kapunk [11]. Így a prioritások megállapítása történhet egy olyan pszeudo véletlenszámgenerátorral, mellyel minden körben az összes ágensen ugyanazt a véletlen számot állítja elő, és ezzel a véletlenszámmal bitenkénti kizáró vagyot használva újracímkézzük az összes csúcsot, melyek az így átcímkézett gráfban minden szomszédjuknál kisebbek lesznek prioritásban.

3.4.5. Kommunikációs minta

Amennyiben a torony szerepet betöltő robotok üzeneteire nem teszek megkötést, akkor megtörhet a fa jellege a kiépített hálózatnak, ezért a networkerek a kommunikációjukat az alábbi szabályok alapján végzik:

- minden robot **3** időegységre kap prioritást
- ha egy csúcs fokszáma **1**-re csökken - kiválhat a gráfból, ha van prioritása a lépés **0.** fázisában.
- $\overline{link\_info}$ -t csak akkor küldhet egy torony, ha van prioritása a lépés **0.** fázisában
- $\overline{scan\_info}$ -t csak akkor küldhet, ha van prioritása a lépés **1.** fázisában



17. ábra. $\widehat{\textbf{\textit{tower}}}$ comm

Az így kialakuló kommunikációs minta 17. ábrán látható, melyen a következők történnek:

- a $\boldsymbol{r}$ csúcs kiválik a gráfból, s az **1**-es fázisban már keresővé is válik.
- a $\boldsymbol{b}$ csúcsnak megvan a prioritása a körre, ezért tájékoztatja $\boldsymbol{c}$-t az $\boldsymbol{a}$ jelű szomszédjáról $|\boldsymbol{c} - \boldsymbol{a}| < \boldsymbol{d_r}$
- az **1.** fázis kezdetén $\boldsymbol{c}$ eldöntötte, hogy kiván élni a lehetőséggel és átveszi $\boldsymbol{b}$-től az $\boldsymbol{a}$ csúcsot, ezért bontja a kapcsolatát $\boldsymbol{b}$-vel és jelzi $\boldsymbol{a}$-nak, hogy van egy új szomszédja
- eközben $\boldsymbol{b}$ elküldi $\boldsymbol{u}$-nak, akit lát ugyan, de mivel nem ismeri, ezért tájékoztatja arról, hogy a legutolsó távolság mérésben mennyire is volt ő távol.

- az $\boldsymbol{u}$ csúcsnak tetszik az ajánlat - könnyebbé válik a gráf ha ő $\boldsymbol{b}$-vel van összekötve, ezért bontja a kapcsolatát a mélységi szülőjével, és visszajelez $\boldsymbol{b}$-nek, hogy helyreálljon a szimmetria

Biz:

Ha ez a kommunikációs minta hibához vezetne, az kétféleképp képzelhető el:

- két komponensre szakad a gráf
- kialakul egy kör

mivel mindkét esetben a $|\boldsymbol{E}| = |\boldsymbol{V}| - \mathbf{1}$ feltétel sérül, ezért csak a következőt kell belátni: nem fordulhat elő az, hogy két csúcs ugyanarról az élről nyilatkozik egymásnak különböző döntéseket:

- a **0.** körben csak $\overline{\textit{link\_error}}$ elküldésére kerülhet sor, mely ha a másik oldalról is $\overline{\textit{link\_error}}$, akkor éppen megszünik a gráfunk, ami a vezető megléte miatt elképzelhetetlen. Valamint, mivel jelenleg $\boldsymbol{r}$ prioritásban van, ezért $\boldsymbol{r}$ egyetlen éle sem szünhet meg, ugyanis a szomszédai biztosan nincsenek prioritásban.
- az **1.** körben elhangzó üzenetek kizárólag a $\boldsymbol{b}$ csúcs kommunikáció környezetében levő élekre vonatkozik, ezért biztosan csak maximum egy csúcs fog nyilatkozni két élről
- a **2.** körben: ha az **1.** körben történt események miatt változott a felállás az $\boldsymbol{u}$ csúcs számára és már nincs összekötve a mélységi szülőjével, akkor figyelmen kívül kell hagynia az üzenetet (ha többet kap, akkor is csak eggyel foglalkozik), és mivel a mélységi szülőjének ő nem lehet szülője, ezért ott nem lehet üzenet duplazás. Mivel $\boldsymbol{u}$ eddig nem volt összekötve $\boldsymbol{b}$-vel, ezért annak a kapcsolatnak a létrehozása semmilyen akadályba nem ütközhet.

3.4.6. Tornyok működése

Minden torony működése közben nyilvántarja a torony szomszédait egy $\boldsymbol{K}$ halmazban.

A lépéseiket a kapott üzenetek feldolgozásával kezdik, a felderítőkhöz hasonlóan ugyanúgy használnak egy véletlen algoritmust az arra járó networkerek tájékoztatására a jelenlegi állapotukról és szerepükről egy $\overline{scan\_info}$ segítségével. Amennyiben egy torony kap $\overline{scan\_info}$ -t egy társától figyelmen kívül hagyja azt.
Amennyiben egy $\overline{interlock}$ üzenetet kap, abban az esetben valamelyik szomszédja felvette őt mint szomszédot, ebben az esetben ennek a csúcsnak is fel kell vennie a konzisztencia megőrzése céljából.
A szomszédos tornyok dönthetnek úgy, hogy megszüntetik a kapcsolatukat ezzel a toronnyal, ekkor egy $\overline{link\_error}$ üzenetet kap - ekkor törölnie kell a $\boldsymbol{K}$ halmazból az üzenet küldőjét a korábban említett okokból.
A tornyok által használt mélységmérő csomagoknak a már korábban tárgyalt módszer szerint kell tudniuk kezelni a csúcsokat, melyektől már kaptak
$\overline{measure\_depth}$ üzenetet, egy $\boldsymbol{O}$ halmazban nyilván kell tartaniuk és egészen addig nem küldhetnek tovább $\overline{measure\_depth}$ csomagot, míg az $|\boldsymbol{O}|+\mathbf{1} = |\boldsymbol{K}|$ feltétel nem teljesül. Amennyiben $|\boldsymbol{O}| = |\boldsymbol{K}|$ teljesül, a küldőt leszámítva minden szomszédjukat értesíteniük kell a mélység mérés befejeződésének érdekében.
A helyi kitüntetettséget biztosító prioritás segítségével lehetőség van arra, hogy más tornyokkal úgy kommunikáljon, hogy azok alárendeltek legyenek abban a körben a prioritásban lévő networker szemszögéből, ugyanis nem prioritásban lévő networkereknek tilos kiválnia a gráfból.
A kommunikációs minta szerint **3** fázist különböztet meg az algoritmus, és egy-egy ilyen **3** lépésből álló sorozatra szereznek a tornyok helyi prioritást. Az első fázisban egy torony nyilatkozhat egy szomszédjáról egy másik szomszédjának a $\overline{link\_info}$ üzenet segítségével, ezáltal lehetőséget adva arra, hogy őt kiegyszerűsíthessék, amennyiben látja a megemlített szomszédot.
A gráf véletlenszerű pontoktól mért távolsága az átmérőméréshez hasonló módon történik, annyi módosítással, hogy visszafele nem jön róla üzenet. Amennyiben egy csúcsnak már van mérés szerinti színe, akkor minden további mérőcsomagot eldob.
A tornyok végül a fent leírt módszerekkel megállapítják, hogy merre és mennyit

mozduljanak el.

Állítás:

A már összekapcsolódott tornyok által kifeszített gráf mindig fát alkot.

Biz:

Mindig $\boldsymbol{n-1}$ él van a gráfban:

1. az $\widehat{\textbf{\textit{init}}}$ állapotban megjelenik pontosan egy $\widehat{\textbf{\textit{tower}}}$, és mivel az élek száma **0**, ezért fennáll a tulajdonság

2. új $\boldsymbol{c} \in \mathbb{C}$ a meglévő gráfba való bekapcsolódásakor nem romlik el a tulajdonság, ugyanis csupa **2** fokszámú csúcs köti össze $\boldsymbol{c}$-t és a már meglévő fát, melyekből pontosan egynek megnövelte a fokszámát.

3. torony váltáskor nem romlik el a tulajdonság:
kizárt hogy egyszerre több torony is megpróbálja megszüntetni a kapcsolatot szomszédaival, míg csak egy új kapcsolatot hoznak létre, mert a $\boldsymbol{c}$ környezetében csak 1 csúcsnak küld $\boldsymbol{c}$ $\overline{\textit{link\_info}}$ üzenetet.

4. torony kiválásakor már vagy **0** a fokszáma a csúcsnak - ami esetén egyszerre kellett két oldalról $\overline{\textit{link\_error}}$ -t kapni - ezt a lehetőséget a kommunikációs minta zárja majd ki -, vagy egyre csökken a fokszáma, ami esetén a jelenléte a gráfban fölöslegessé válik.

4. Összefoglalás

Jelen munka bemutatott egy olyan módszert, mely képes egy infrastruktúramentes területen külső beavatkozás nélkül az esetlegesen felmerülő igényeket felderíteni, hálozatba kapcsolni, és esetleges megszűnésükkor újracsoportosítani az erőforrásait. A Networkers algoritmusban használt két szerepkör elégnek bizonyult egy szimulált környezetben való hálózat kiépíthetőségének vizsgálatára.

A kezdetben egy bolyból induló networkerek képesek viszonylag gyorsan megkezdeni az aktív hálózatkiépítést a kezdeti pásztázási fázis után. Ezután a felderítők egyre nagyobb láncokba szerveződve próbálnak meg egyre távolabbi pontokat a hálózatba bekapcsolni, míg a tornyok a hálózat fenntartásához szükséges egységek csökkentésére koncentrálnak.

Az algoritmusnak készült egy implementációja, mely segítségével az elméleti megoldások helyességét egy szimulált környezetben teszteltem. A megvalósított programban jól követhető, ahogyan a sok száz ágens párhuzamosan döntéseket hoz és követik a változó igényeket a síkon, mely a felépített hálózatot folyamatos változásokra készteti.

A sikeres problémamegoldás mellett azonban van néhány hátránya is a jelenlegi változatnak, ugyanis az algoritmus kialakításakor egyáltalán nem vettem figyelembe annak energiafelhasználását. Így a felderítők fáradhatatlanul kutatnak új kliens-pont után. Amennyiben a klienspontok nem szűnnek meg és nem jelennek meg újak, tanácsos lenne ezt a tevékenységüket felfüggeszteni. Mindemellett a Steiner-fa kialakítását még nem sikerült teljesen elérni.

Ha még most azonnal nem is, de az algoritmus későbbi változata már képes lehet majd helytállni a valódi világban és hálózatot építeni külső beavatkozás nélkül. Ennek a legnagyobb hasznát olyan helyen vennénk, ahova nem tudunk eljutni, vagy a hálózat kiépítésének annyira gyorsan kell követnie a változásokat, hogy egyáltalán nem éri meg kiépíteni. Elképzelhetőnek tartom, hogy a klienspontok úgyszintén autonóm robotok, amik szabadon járják a területet, a networkerek feladata pedig, hogy folyamatosan kapcsolatban tartsák őket egymással.

Az itt alkalmazott technikák segítségével felismertem, hogy a formációbaállás és annak megtartása milyen komoly problémákat vethet fel. Jelenleg tervezés alatt áll egy általánosabb módszer a formáció felvételre, mely a dolgozatban tárgyaltnál

valószínűleg hatékonyabban oldja majd meg a felderítés problémáját.

A tornyok igyekeznek a kommunikációs távolságukat a lehető legjobban kihasználni, amennyiben egy ilyen hálózaton tényleges hálózati forgalom jelenne meg, abban az esetben ez az áteresztőképességet gyengítheti. Amennyiben a tornyoknak tudomása van a rajtuk keresztülfolyó forgalom mennyiségéről, akkor az algoritmus kiterjeszthető oly módon, hogy az áteresztőképességet maximalizálhassa. Elképzelésem szerint a jelenlegi geometriai metrika lecserélhető egy alternatívra, melynek segítségével az ilyen minőségi jellemzők kielégítése is lehetővé válhatna.

5. Irodalomjegyzék

[1] Iványi Antal. Párhuzamos algoritmusok, 2003.

[2] Tucker Balch, Ronald C. Arkin, and Senior Member. Behavior-based formation control for multi-robot teams. IEEE Transactions on Robotics and Automation, 14:926–939, 1999.

[3] D R Dreyer and M L Overton. Two heuristics for the euclidean steiner tree problem. Journal of Global Optimization, (13):95–106, 1998.

[4] Kevin Fall. A delay-tolerant network architecture for challenged internets. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[5] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. Softw. Pract. Exper., 21:1129–1164, November 1991.

[6] Andrew Howard, Maja J. Matarić, and Sukhat Gaurav S. An incremental self-deployment algorithm for mobile sensor networks. Auton. Robots, 13:113–126, September 2002.

[7] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. pages 299–308, 2002.

[8] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. 2002.

[9] M Labella, T H Dorigo, and J-L Deneubourg. Self-organised task allocation in a group of robots. In Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (DARS04, 2004.

[10] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. In Petre Dini, Pascal Lorenz, and José Neuman de Souza, editors, Service Assurance with Partial and Intermittent Resources, volume 3126 of Lecture Notes in Computer Science, pages 239–254. Springer Berlin / Heidelberg, 2004.

[11] Lovász László. Computation complexity, 1999.

[12] Lynne E. Parker and Andrew Howard. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. International Journal of Robotics Research, 25:431–447, 2006.

[13] Guiling Wang, Student Member, Guohong Cao, and Thomas F. La Porta. Movement-assisted sensor deployment. pages 2469–2479, 2004.

[14] Torben Weis, Helge Parzyjegla, Michael A. Jaeger, and Gero Mühl. Self-organizing and self-stabilizing role assignment in sensor/actuator networks. In Robert Meersman and Zahir Tari, editors, The 8th International Symposium on Distributed Objects and Applications (DOA 2006), volume 4276 of LNCS, pages 1807–1824, Montpellier, France, October 2006. Springer.

[15] Yansheng Zhang, Farokh Bastani, and I-Ling Yen. Self-stabilizing structure forming algorithms for distributed multi-robot systems. In Proceedings of the 2007 international conference on Embedded and ubiquitous computing, EUC'07, pages 754–766, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Y. Zou and Krishnendu Chakrabarty. Sensor deployment and target localization based on virtual forces. In INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, 2003.