

TDK-dolgozat

Orosz Ákos, Róth Gergő

Algoritmusok tervezése TDMA alapú, gyűrű topológiájú szenzorhálózatok ütemezéséhez

Szerzők: Orosz Ákos, Róth Gergő (Mérnök Informatikus BSc Szak 3. évfolyam)

Témavezető: dr. Simon Gyula, egyetemi docens

Pannon Egyetem

Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

Absztrakt

A vezeték nélküli szenzorhálózatok érzékelők tucatjainak, esetleg százainak együttműködésével hajtanak végre feladatokat. A legtöbb alkalmazásban több urgásos, ad-hoc hálózatokat használunk, melyeknek alapvető kihívása a korlátos (pl. telepes) áramellátás, de ezzel egyidőben kívánalom a hosszú működési idő is. A szenzorok ezért nagyon kis kitöltési tényezővel működnek, pl. TDMA (Time Division Multiple Access) alapon, amikor is az idő nagy részében alszanak, kis részében pedig aktívak. A hálózattal szemben támasztott további követelmény a korlátos késleltetés (például egy riasztási esemény bekövetkezése után az információ érkezzon meg a központba legfeljebb 5 másodpercen belül).

A hálózatok kommunikációs szempontból többféle topológiát alkothatnak (mesh, fa, gyűrű), melyek kiválasztása alkalmazásfüggő. A dolgozatban a szenzorhálózatos biztonsági rendszerekben is sikerrel alkalmazott gyűrű topológiát vizsgáljuk.

A TDMA ütemezés problémája visszavezethető gráfokon végzett műveletekre. Gyűrű topológiájú hálózatokban a cél többnyire valamilyen speciális kör keresése (Hamilton-kör, b-Hamilton-kör) a kapcsolati gráfon. Ezek a műveletek NP-teljes problémák, így a munka célja automatikus vagy félautomatikus, hatékony heurisztikákat tartalmazó algoritmusok tervezése.

A dolgozat a szakirodalom áttekintése után javaslatot tesz gyűrű topológiák hibátűrő ütemezésének megvalósítására és definiálja a b-Hamilton-kör fogalmát. Az ütemezéshez szükséges b-Hamilton-körök kereséséhez a hálózat földrajzi topológiáját is felhasználó, új heurisztikákat tartalmazó algoritmusokat javasol. A dolgozat a javasolt algoritmusokat kiterjedt tesztek segítségével értékeli és hasonlítja össze.

A dolgozat bemutatja a kidolgozott algoritmusok valós felhasználási lehetőségeit is.

Tartalomjegyzék

1. Bevezetés	4
2. Hibatűrő TDMA ütemezés	7
3. Körkeresés gráfokban	10
3.1. Hamilton kör létezésének feltételei	10
3.2. Heurisztikus Hamilton kör-kereső algoritmusok	11
4. Hibatűrő Hamilton-körök keresése	13
4.1. Brute-force algoritmus	14
4.2. Klaszter alapú keresés	15
4.3. Klaszterező algoritmus	18
4.3.1. Szenzorok teljes részgráfokra osztása	18
4.3.2. Teljes részgráfok diszjunktta tétele	20
4.3.3. Klaszterek egyesítése	21
4.4. Buborékos keresés	22
5. Összehasonlító értékelés	29
5.1. Tesztelési módszerek	29
5.1.1. Teszthálózatok	29
5.1.2. Tesztelt algoritmusok	31
5.1.3. Mérési módszerek	32
5.2. Tesztelési eredmények	32

5.2.1.	"12_kör" teszteredmények	33
5.2.2.	"12_teljes" teszteredmények	34
5.2.3.	"100_kör" teszteredmények	34
5.2.4.	"100_teljes" teszteredmények	35
5.2.5.	"3k10" teszteredmények	36
5.2.6.	"10k10" teszteredmények	37
5.2.7.	"10c10" teszteredmények	38
5.2.8.	"random33" teszteredmények	38
5.2.9.	"random100" teszteredmények	40
5.2.10.	"I9" teszteredmények	40
5.3.	Értékelés	41
6.	Gyakorlati felhasználás	44
6.1.	GraphGenerator fül	44
6.2.	SensorInteract fül	46
7.	Összefoglalás	51

1. fejezet

Bevezetés

A szenzorhálózatok apró, intelligens érzékelő tucatjainak, százainak, esetleg ezereinek együttműködésével megvalósított, egyetlen cél érdekében működő autonóm elosztott alkalmazások. A tér különböző pontjaira elhelyezett egységek általában érzékelő-, de esetleg beavatkozó tevékenységeket is végezhetnek, egymással pedig - tipikusan vezeték nélküli csatornákon - kommunikálnak. Természetesen a mérésen, adatgyűjtésen kívül az érzékelő hálózat képes adatfeldolgozásra, elosztott számítások végzésére is. Mivel a szenzorhálózatok egyszerű, olcsó csomópontokból állnak, ezek száma igen nagy lehet, az érzékelőket nagy sűrűségben el lehet helyezni. Ez új alkalmazási lehetőségeket, irányokat nyitott meg. A szenzorhálózatok manapság már igen elterjedtek, számos alkalmazásban bizonyították hatékonyságukat az adatgyűjtés, környezetvédelem, intelligens mezőgazdaság, épületek, műtárgyak állapotmonitorozása, lokalizáció és követés területén [1].

A szenzorhálózatokban jellegüknél fogva kritikus kérdés az energiafelhasználás: a rendelkezésre álló szűkös energiaforrásból (tipikusan telep) minél hosszabb működési időt kell biztosítani. Vezetéknélküli szenzorhálózatok esetében a TDMA (Time Division Multiple Access) alapú ütemezést használva (pl. a CSMA alapú versengő közeghozzáféréssel ellentétben) valós idejű működés és jelentős energiamegtakarítás érhető el. Ennek oka, hogy az ütemezéssel elkerülhetők egyrészt a csomagok közötti ütközések, másrészt feleslegessé válik a szenzorok üres járatí figyelése, helyette

inkább energiatakarékos üzemmódra váltanak (alszanak) az eszközök, csak akkor ébrednek fel rövid időre kommunikáció céljából, amikor üzenetet adnak vagy fogadnak. Természetesen a TDMA alapú megoldásoknak hátrányai is vannak, nevezetesen a lefoglalt sáv szélességet ritkán használják ki az eszközök, így az átvitel szempontjából ritkán optimálisak [2].

A hálózat felépítése az alkalmazástól függ. Egy adatgyűjtő hálózatot általában fa-felépítésűre vagy mesh-topológiájúra terveznek. Azonban számos alkalmazás nem csak gyűjti az adatot, hanem folyamatosan áramoltatja is azzal a céllal, hogy bármelyik szenzor küldhessen üzenetet bármelyik másiknak (pl. riasztó rendszerek). Ezen hálózatokban sokkal inkább a gyűrű topológiát alkalmazzák [3]. A mi kutatásunk témája hibatűrő hálózatok tervezése és ütemezése gyűrű topológiájú vezeték nélküli hálózatokon.

Hamilton köröket gyakran alkalmaznak hibatűrő hálózatok kialakítására. A "Hamilton kör védelem" pl. tetszőleges topológiájú hálózatban képes linkeket, illetve csúcsok kiesése esetén a Hamilton kör segítségével elkerülő alternatív utat tervezni, ezáltal meghibásodás esetén is biztosítható a szolgáltatás. Ilyen megoldásokat gyakran alkalmaznak vezetékes hálózatok esetén is [4]. Az általunk vizsgált gyűrű topológiájú hálózatokban a Hamilton kör a kommunikáció feltétele: amennyiben a kör mentén megengedünk kétirányú kommunikációt is, akkor az egy-hibatűrő automatikusan megvalósul. Ez a megoldás azonban az energiaszegény szenzorhálózatos környezetben nem hatékony és ezért nem is célszerű. Így e helyett egy hatékony, egyirányú, de mégis hibatűrő ütemezési megoldást javaslunk.

A dolgozatban közölt saját eredmények a következők [5]:

- Hibatűrő ütemezési megoldást javaslunk gyűrű topológiájú, egyirányú kommunikációt megvalósító szenzorhálózatokban.
- A hibatűrő ütemezés támogatására definiáljuk a b-Hamilton kör fogalmát.
- Új klaszter-alapú heurisztikus kereső algoritmust javaslunk b-Hamilton körök keresésére. Az algoritmus a nagy keresési feladatot kisebb részproblémák

sorozatára bontja.

- Új "buborékos", félautomatikus, heurisztikus keresési módszert javasolunk b-Hamilton körök keresésére. Ezen algoritmus a vezeték nélküli hálózatok azon sajátosságát használja ki, hogy a kommunikációs linkek minősége és az eszközök távolsága között erős a korreláció. Ezen az alapon egy "lokálisan kimerítő" heurisztikát adunk.
- A javasolt algoritmusok hatékonyságát széleskörű tesztelés alapján értékeljük.
- Bemutatunk egy szoftver keretrendszert, amelyben implementáltuk a javasolt algoritmusokat oly módon, hogy azok valós hálózatok kezelésre is alkalmasak legyenek. Valós szenzorhálózatokban mérések segítségével bemutatjuk a tervezett TDMA ütemezések működését.

A 2. fejezetben a dolgozat bemutatja a gyűrű topológiájú szenzorhálózatok TDMA alapú ütemezésének alapjait. A 3. fejezet ismerteti a Hamilton-kör keresésének problémáját, ismerteti a probléma megoldására született eredményeket. A 4. fejezetben bevezetjük a b-Hamilton-kör fogalmát és részletesen ismertetjük az általunk adott algoritmusokat a probléma megoldására. Az 5. fejezetben összesen öt kiválasztott algoritmus futási eredményeit tárgyaljuk 10 különböző méretű és tulajdonságú tesztgráfra és azok csúcsindexeinek permutációira. A 6. fejezet bemutatja az elkészült szoftvert, amelynek segítségével (1) különböző algoritmusokkal lehet (b-)Hamilton-kört keresni a felhasználó interakciójával, valamint (2) lehetőséget a fizikai szenzorhálózat felderítésére és az így kapott kapcsolati gráf alkalmazására a keresések soráni. A 7. fejezet összefoglalja a munka eredményeit és a lehetséges további kutatási irányokat.

2. fejezet

Hibatűrő TDMA ütemezés

Az ütemezési feladat megoldásához a vizsgált szenzor-hálózatokat kapcsolati gráfokkal ábrázoljuk. Ebben az egyes szenzorok felelnek meg a gráf csúcsainak, míg a gráf élei a szenzorok közötti kommunikációs kapcsolatokat jellemzik. A csúcsok (szenzorok) térbeli elhelyezkedését (koordinátáit) eltárolva a gráf a valóságnak közel megfelelően, pl. térképen megjeleníthető. A földrajzi elhelyezkedést a javasolt algoritmusainkban használt heurisztikák hatékonyan ki fogják használni.

A gráf élei lehetnek irányítottak vagy irányítatlanok, továbbá szokás a gráf éleinek súlyozásával a kommunikációs link minőségét is jellemezni. Amennyiben a gráf irányított, úgy az A és B csúcs között két él halad ($\overrightarrow{AB}$ és $\overrightarrow{BA}$), potenciálisan különböző súlyokkal ($w_{\overrightarrow{AB}}$ és $w_{\overrightarrow{BA}}$). Az irányított gráf átalakítható az egyszerűbb, irányítatlan gráffá, mely a gyakorlatban jól alkalmazható. Ebben az A és B csúcs között egyetlen irányítatlan AB él halad, melynek súlya a gyengébb minőségű irány súlya ($w_{AB} = \min(w_{\overrightarrow{AB}}, w_{\overrightarrow{BA}})$). Amennyiben $w_{AB} = 0$, úgy az AB élet nem szokás megjeleníteni.

A kapcsolati gráf él-súlyainak meghatározása fontos gyakorlati probléma: ez történhet a vett jel térerőssége (RSSI) alapján, a vett jel LQI minőségi jellemzője alapján, vagy egyéb statisztikai alapú mérés segítségével. Ennek részleteire a dolgozatban nem térünk ki, de megjegyezzük, hogy a 6. fejezetben ismertetett szoftverünk a fenti jellemzők kombinációját használja a hálózat felderítése során.

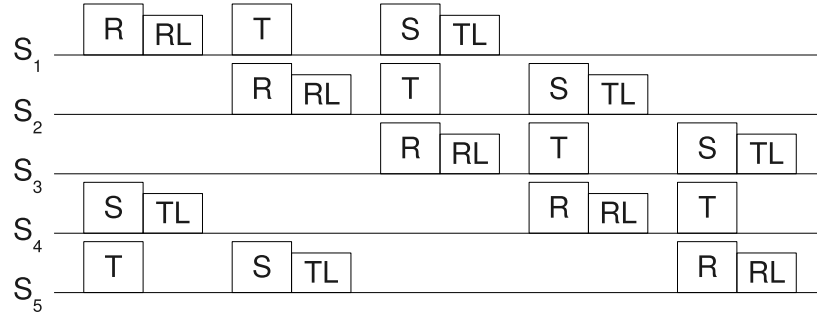
A kapcsolati gráfot további feldolgozás céljából bináris kapcsolati gráffá alakítjuk: amennyiben $w_{AB} > w_{limit}$, ahol w_{limit} egy adott minimális minőségi index, úgy az AB él szerepel a bináris kapcsolati gráfban, ellenkező esetben pedig nem. Ennek a bináris kapcsolati gráfnak a szemléletes jelentése a következő: él vezet két csúcs között, ha a két csúcs által reprezentált szenzor között (a w_{limit} -tel jellemzett) "elég jó" minőségű kommunikáció lehetséges. A w_{limit} tervezési paraméter.

A dolgozatban vizsgált gyűrű topológiákban a szenzorok körkörösén küldik egymásnak az üzeneteket. A szenzorok közti kapcsolati gráfot tekintve az üzenetek egy Hamilton-körön közlekednek. A hálózat ütemezéséhez ezt a Hamilton-kört kell megtalálni, ami egy NP-teljes probléma [6], de alkalmazhatók heurisztikák, melyek csökkenteni tudják a keresésre szánt időt.

A gyakorlatban használt hálózatoknak hibatűrőnek is kell lennie, hogy meghibásodás esetén a szolgáltatást továbbra is fenn lehessen tartani. A dolgozatban egy-hibatűrést vizsgálunk, ami azt jelenti, hogy a hálózatnak egy szenzor kiesése vagy egy link meghibásodása esetén is tovább kell működnie. Ennek megoldására vezettük be a később részletesebben is tárgyalt b-Hamilton-kör fogalmát, mellyel ütemezve egy szenzor nem csak a körben utána jövőnek tud üzenetet küldeni, hanem az azt követőnek is, így ha egy szenzor vagy összeköttetés meghibásodik, a hálózat automatikusan át tudja a hibás helyet hidalni, ezáltal fenntartva a működést.

Ilyen ütemezést használva minden szenzor periodikusan ébred fel, és hajtja végre a következő műveleteket: üzenetet fogad az előző szenzortól (R - Receive), vagy hiba esetén az azt megelőzőtől (RL - Receive Long), illetve üzenetet küld a sorban utána következő szenzornak (T - Transmit). Majd belehallgat abba az üzenetbe, amit a következő szenzor küld az azt követőnek (S - Snoop), és ha ezt az üzenetet nem hallja (vagyis valószínűleg a következő szenzor nem üzemel), akkor elküldi az üzenetet a következő utáni szenzornak (TL - Transmit Long). Minden üzenetet nyugtáz a hálózat, amennyiben megérkezik. A 2.1. ábrán látható egy példa ütemezés 5 szenzort tartalmazó hálózatban.

Ideális esetben, mikor nem vesznek el üzenetek, az RL és TL funkciókra nincsen



2.1. ábra. Ütemezési műveletek (R, RL, T, S, TL) 5 méretű hálózatban

szükség: az egyszerre végrehajtott adás-vétel (T-R) után a küldő nyugtát kap a vevőtől, majd mind a vevő, amind az adó elelszik. Ha egy nyugta nem érkezik meg (pl. S_2 nem kap nyugtát S_3 -tól), a küldő ismételheti az adást (vagyis az R és T időszetek több csomag küldésére is elegendő hosszúak lehetnek; mi a 6. fejezetben bemutatott megvalósításunkban két adási kísérletet engedélyeztünk a T időszetben). Ha egyik adási kísérletére sem érkezik nyugta, akkor az adó belehallgat a következő kommunikációs időszetbe (S_2 meghallgatja, hogy S_3 milyen üzenetet küld, ha küld). Ha a hallgatózás sikeres (S_2 hallja S_3 üzenetét), akkor a küldő befejezheti a ciklust (S_2 alvó állapotra vált), míg ellenkező esetben elküldi az üzenetet a következő utáni szenzornak, aki a második (RL) fogadásának idejében megkapja a csomagot (S_2 TL-üzenetet küld S_4 -nek). Ez a módszer hibatűrő viselkedést eredményez, amennyiben egy szenzor vagy összeköttetés elromlik.

A fenti hibatűrő ütemezés megvalósításához ismerni kell a hálózat kapcsolati grájában egy Hamilton kört, amelynek mentén a hibamentes ütemezés végrehajtható, ráadásul a gráfnak tartalmaznia kell további éleket is, amelyek mentén a másodlagos üzenetek (TL - RL) továbbíthatók.

3. fejezet

Körkeresés gráfokban

3.1. Hamilton kör létezésének feltételei

Habár jelenleg nem ismerünk szükséges és elégséges feltétel arra, hogy egy gráfban létezzen Hamilton-kör, számos elégséges feltétel azonban ismert:

T1: Dirac tétele szerint egy n -ed fokú ($n \geq 3$) gráfban létezik Hamilton kör, ha minden csúcs fokszáma legalább $\frac{n}{2}$ [7].

T2: Ore tétele szerint egy n -ed fokú ($n \geq 3$) gráfban található Hamilton kör, ha bármely két nem szomszédos csúcspontra igaz, hogy a két csúcs fokszámainak összege legalább n [8].

T3: Ugyancsak Ore tétele szerint, ha az n -ed fokú ($n \geq 3$) gráfban az élek száma m és igaz, hogy $m \geq \frac{n^2-3n+6}{2}$, akkor a gráfban létezik Hamilton-kör [9].

T4: Pósa elégséges feltétele szerint, ha az n -ed fokú ($n \geq 3$) gráfban a csúcsok fokszámai rendre $d_1 \leq d_2 \leq \dots \leq d_n$, és $d_k > k, 1 \leq k < \frac{n}{2}$ esetén, akkor a gráfban létezik Hamilton-kör [10].

T5: Pósa Lajos másik eredménye szerint létezik olyan c konstans, melyre az n -ed fokú, $c n \log(n)$ csúcsú véletlen gráfokban a Hamilton kör létezésének valószínűsége tart 1-hez, ha n tart végtelenhez [11].

A Hamilton kör létezésére egy egyszerű szükséges feltétel adható a gráf adjacencia-mátrixának hatványai segítségével:

T6: Ha az n -ed fokú gráf adjacencia mátrixa A és ennek n -edik hatványa A^n , akkor

a Hamilton kör létezésének szükséges feltétele, hogy A^n átlójának valamennyi eleme pozitív legyen [12].

A T1-T4 tételek által érintett esetekre ismertek polinomrendű Hamilton kör-kereső algoritmusok, azonban a Hamilton kör keresés általános problémája NP-teljes [13]. Így a körkereső algoritmusok kutatása régóta arra irányul, hogy egyes speciális esetekre hatékony - általában heurisztikus - algoritmusokat találjanak.

3.2. Heurisztikus Hamilton kör-kereső algoritmusok

A korai Hamilton kör-kereső algoritmusok kimerítő keresést alkalmaztak. A szekvenciális kimerítő kereső algoritmusok alapelve a következő: az algoritmus egyesével növeli az útvonal méretét, mindaddig új csúcsokat hozzáadva a körhöz, amíg zsákutcába nem jut. Ezután visszalépést hajt végre egy olyan csúcsig, ahonnan lehetséges a továbblépés és ismételten próbálkozik újabb csúcsok hozzáadásával az elakadásig, vagy a Hamilton kör megtalálásáig. Ilyen elven működik pl. Kroft algoritmus [14].

A keresés hatékonyságának növeléséhez különféle szabályrendszereket hoztak létre, amelyek segítségével egy téves döntés korai szakaszban felismerhető volt, pl. Hakimi [15] és Rubin [16] algoritmusai.

Pósa Lajos 1976-ban publikált rotációs transzformáció-alapú algoritmus [11]. Pósa algoritmus [11] a következő ötleten alapul: amennyiben az egyszerű szekvenciális keresés zsákutcába jut a $p_1, p_2, \dots, p_k$ csúcssorozat esetén, de létezik egy $(p_1, p_j), 1 < j \leq k$ él a gráfban, akkor a csúcssorozat egy forgatási transzformációval (a $p_1, \dots, p_{j-1}$ szakasz megfordításával) átalakítható a $p_j, p_{j-1}, p_{j-2}, \dots, p_1, p_j, p_{j+1}, \dots, p_k$ sorozattá, amellyel a keresés tovább folytatható. Mivel Pósa cikkében az implementációs részletekkel nem foglalkozott, az algoritmusnak számos megvalósítása, továbbfejlesztése született, pl. Angulin és Valiant valószínűség-alapú keresése, mely legalább $O(n(\log(n))^2)$ csúcsot tartalmazó véletlen gráfokban közel 1 valószínűséggel talál Hamilton kört [17]. Shields is Pósa algoritmusát finomítja a transzformációs lépés előtti kiértékelési mechanizmus segítségével [18].

Bollobás Béla és társai 1987-ben készítették a Ham nevű algoritmust, ami irányított

véletlen gráfokban keres. Az algoritmus a kör-kiterjesztés elvét alkalmazza, szintén rotációs transzformációk segítségével [19].

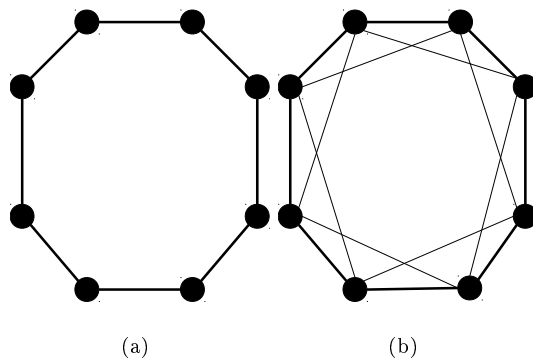
A transzformációs keresést segítő, a kutatók különféle heurisztikákkal bővítették az algoritmusokat. Brunacci pl. a csúcsok fokszáma alapján állított fel prioritási listákat, amelyek segítségével vezérlik a keresést [20].

Egy érdekes technikai megoldással Altschuler és társai a szimulált lehűtés módszerét alkalmazták heurisztikaként a Hamilton-kör probléma megoldására [21].

Egy érdekes heurisztikát publikált Ashay Dharwadker 2004-ben [22]. Ez az algoritmus polinom időben lefut, és ha egy gráfban található Hamilton-kör, akkor nagy valószínűséggel meg is találja benne, illetve ha a gráf rendelkezik bizonyos tulajdonságokkal, akkor garantáltan talál benne Hamilton-kört. Az algoritmus alapötlete, hogy nem visszalépéses alapon keres, mint a korábbi heurisztikák, hanem heurisztikus módon elkezd felépíteni a kört és soha nem lép vissza. Hasonlóan Pósa eredeti ötletéhez az algoritmus elakadás esetén a javításokhoz rotációs transzformációkat hajt végre. Dharwadker algoritmus első lépésként utat keres a gráfban, ezt addig csinálja, amíg talál újonnan bevezethető csúcsot. Ha az eddig felépített utat nem tudja folytatni, akkor nem kezd el keresni másik irányba, hanem javító heurisztikákkal próbálkozik: a következő lépésekben úgy próbálja meg kiegészíteni az utat, hogy a korábban felépített út végén, illetve közepén megfordít egy részutat, ha van rá lehetőség. Több ilyen próbálkozást változtatva polinom időben befejezi a futását. Ezek végén nagy valószínűséggel talál Hamilton-kört a gráfban, sőt a legtöbb esetben többet is.

Egy más típusú heurisztikát publikált 1987-ben Sudhansu B. Karmakar [23]. Ez az algoritmus egy gráf adjacencia mátrixán végez különféle transzformációs lépéseket, megpróbálva a diagonális elemeket egységnyire állítani. Ez az eljárás nem szűkíti le a keresési teret, így mindenképpen megtalálja a kört, amennyiben van a gráfban, viszont így futási ideje sem mérhető polinomiális időben.

Hamilton-kör keresésére tehát léteznek jól működő heurisztikák. Az általunk alkalmazott hibátűrő Hamilton körök keresésére azonban az irodalomban nem ismertek módszerek. A következő fejezetben ezen problémakör megoldásával foglalkozunk.



4.1. ábra. (a) Hamilton-kör, (b) b-Hamilton-kör

4. fejezet

Hibatűrő Hamilton-körök keresése

Ahhoz, hogy a 2. fejezetben bemutatott ütemezés működni tudjon, a hálózat kapcsolati gráfjának tartalmazni kell legalább egy hibatűrő (b-Hamilton) kört. Ha ez teljesül, akkor a gráfban egy csúcs vagy él kiesése esetén a megmaradt gráfban még mindig található olyan Hamilton-kör, mely az eredeti útvonalat követi, így a hálózat tud tovább működni. Ezt a hibatűrő Hamilton kört nevezzük b-Hamilton körnek.

Definíció: *b-Hamilton-körnek* nevezzük egy $G = (V, E)$ gráf olyan $H_b = (V, E_b)$ részgráfját, ahol H_b csúcsai $(v_1, v_2, \dots, v_N)$, és bizonyos élei $(v_1, v_2), (v_2, v_3), \dots, (v_{N-1}, v_N), (v_N, v_1)$ Hamilton-kört alkotnak G -ben, továbbá $(v_1, v_3), (v_2, v_4), \dots, (v_i, v_i + 2), \dots, (v_N - 2, v_N), (v_N - 1, v_1), (v_N, v_2) \in E_b$. A 4.1. ábrán látható egy példa Hamilton-, illetve b-Hamilton-körre.

A következő alfejezetekben arról lesz szó, miként lehet b-Hamilton-köröket keresni

gráfokban. A feladat egyszerű, ámde nem hatékony (brute force) megoldásának ismertetése után két új heurisztikus algoritmusra teszünk javaslatot: ismertetjük a klaszter alapú és a buborékos keresési algoritmusokat.

4.1. Brute-force algoritmus

Ez az algoritmus nagyon egyszerű, mentes mindenféle heurisztikától. Mélységi keresést alkalmazva addig keres a gráfban, amíg meg nem vizsgálta az összes lehetséges esetet, vagy nem talált egy kört. A keresés bármelyik csúctól kezdődhet, és minden lépésben, ha az utoljára kiválasztott két csúcs v_1 és v_2 , akkor minden csúcsra, melyre $v : v \in V, (v_1, v) \in E, (v_2, v) \in E$, elvégez egy szétválasztást, és onnan keres tovább. Az algoritmus pszeudo-kódja a következő:

```

select any  $v \in V$ 
for  $\forall v_1 \in V : (v, v_1) \in E$ 
     $C = \text{findvertexCycle}(G, [v|v_1])$ 
    if  $|C| = |G|$ ,  $C \text{ found}$ ,  $fi$ 
end
findvertexCycle()
for  $\forall v \in V : v \notin C \wedge (C[\text{end}], v) \in E \wedge (C[\text{end}-1], v) \in E$ 
    if  $|C| < |G| - 1$ 
         $C_1 = \text{findvertexCycle}(G, [C|v])$ 
        if  $|C_1| = |G|$ ,  $\text{return } C_1$ ,  $fi$ 
    else if  $(v, C[1]) \in E \wedge (v, C[2]) \in E \wedge (C[\text{end}], C[1]) \in E$ 
         $\text{return } [C|v]$ 
    fi
end
return  $\emptyset$ 
end.
```

4.2. Klaszter alapú keresés

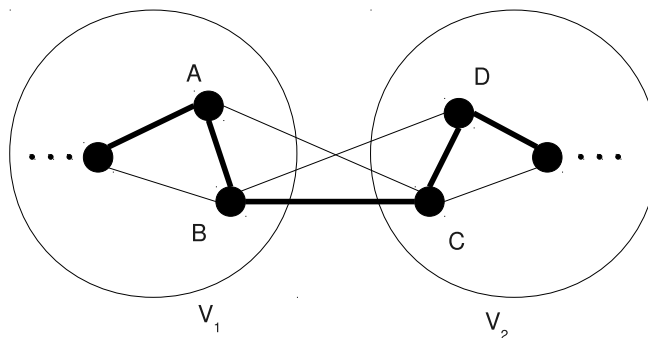
Amennyiben a kapcsolati gráfban lehet klasztereket (részgráfokat) úgy definiálni, hogy a klaszterben szereplő csúcsok egymáshoz közel helyezkedjenek el - és így egymással jó összeköttetésekkel rendelkezzenek -, akkor ezen klaszterek jól felhasználhatók b-Hamilton körök keresésére is.

A klaszter alapú algoritmus a következő alapötletre épül: definiáljunk jól összefüggő klasztereket a gráfon. Ezen klaszterek között keressünk a klasztereket összekötő b-Hamilton kört. Erre a célra definiálni fogjuk a boríték-él fogalmát és egy segédgráf segítségével visszavezetjük a feladatot az egyszerű Hamilton kör keresési problémára. A klaszterek közötti b-Hamilton kört ezután a klasztereken belüli b-Hamilton utak segítségével kiterjesztjük a kapcsolati gráf b-Hamilton körévé. A megoldás során mind a segédgráfon való Hamilton-kör keresés, mind a klasztereken belüli b-Hamilton út keresése kisebb feladat, így az eredeti feladatot több kisebb, hatékonyabban megoldható részfeladatra dekomponáljuk.

A keresés elkezdéséhez szükségünk van diszjunkt klaszterekre, melyek olyan $V_1, V_2, \dots, V_k$ diszjunkt részgráfok, melyekre $V_1 \cup V_2 \cup \dots \cup V_k = V$. A klaszterek részgráfokat definiálnak $(G_1, G_2, \dots, G_k)$, ahol $G_i = (V_i, E_i)$, $E_i = (\{v_u, v_w\} : v_u, v_w \in V_i, (v_u, v_w) \in E)$.

Ez a keresés feltételt is jelent a b-Hamilton-körre vonatkozóan, ugyanis a körnek minden klaszterbe pontosan egyszer szabad belépni, és egyszer kilépnie. Tehát a b-Hamilton-körnek egy klaszter összes csúcsát egymás után kell tartalmaznia, majd továbblépni a következőre, és ott is bejárni minden csúcsot. Ez ugyan szigorítás a körre vonatkozóan, de a klaszterek tulajdonságai (erős összefüggés) miatt az esetek nagy részében igaz, hogy ha van b-Hamilton kör a gráfban, akkor van olyan is, ami ezen feltételeknek megfelel. A korlátozás nagy mértékben csökkenti a keresési teret, így rövidebb idő alatt lehet kört találni a gráfban, feltéve, hogy a klaszterek megfelelően vannak megválasztva.

Mivel b-Hamilton-kört keressünk a gráfba, az ilyen típusú köröknek egy jellegzetes formája van, mikor egyik klasztert elhagyva a másikba lép be. A két klaszter között



4.2. ábra. Boríték-él két klaszter között

három élnek kell mennie a 4.2. ábrának megfelelően. A 4 csúcsnak ez a kapcsolati gráfja a boríték-él nevet kapta, mivel erre hasonlít az alakzat.

Definíció: A *boríték* egy olyan részgráf, mely 4 csúcsból (A, B, C, D) áll, 5 éle van (AB, AC, BC, BD és CD), ahogy a 4.2. ábrán is látható.

Definíció: Egy részgráf G_e *boríték-élet* alkot a C_1 és C_2 klaszterek között akkor és csak akkor, ha G_e egy boríték, és $A, B \in C_1, C, D \in C_2$.

A klaszter alapú kereső algoritmus négy lépésből áll:

1. lépés Generáljuk a boríték-gráfot.

A Q boríték-gráf csúcsai a gráf klasztereit jelölik, az élei pedig azt jelölik, hogy az adott két klaszter között van-e legalább egy boríték-él.

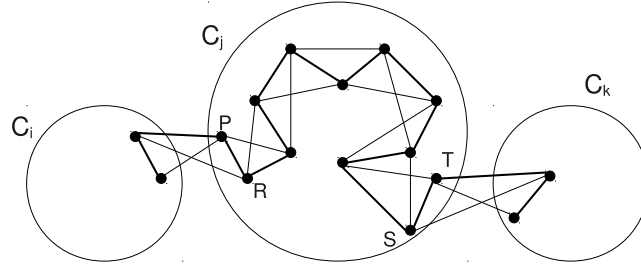
2. lépés Keressünk Hamilton-kört a boríték-gráfon.

Mivel a klaszterek száma általában kevés a gráfban, ezért a boríték-gráfban történő Hamilton-kör megkeresését a brute-force algoritmus is gyorsan elvégzi, vagy erre a feladatra bármely jól ismert heurisztikus módszer alkalmazható. Mi az implementációnkban itt brute-force algoritmust alkalmaztunk.

3. lépés terjesszük ki a kört a klasztereken belülre.

Miután az algoritmus talált kört a boríték-gráfban, megpróbál minden klaszterben utat keresni. Itt fontos megjegyezni, hogy a boríték-gráf csak azt jelzi, hogy van-e boríték-él a két klaszter között, de azt nem, hogy mennyi, és melyek azok.

Tegyük fel, hogy a Q -beli Hamilton-kör keresztül megy a C_i, C_j, C_k klasztereken ilyen sorrendben. Ez esetben nevezzük a C_j -beli (v_a, v_b) csúcspárokat kilépő-pároknak,



4.3. ábra. Klaszteren belüli b-Hamilton-kör

ha létezik C_j és C_k között olyan boríték-él, melyben a 4.2. ábrát tekintve v_a felel meg az A csúcsnak, és v_b felel meg a B csúcsnak. Hasonlóan a (v_c, v_d) csúcspárokat nevezzük belépő-pároknak, ha létezik C_i és C_j között boríték-él, melyre v_c felel meg a C csúcsnak, és v_d felel meg a D csúcsnak. Ezek után olyan b-Hamilton-utat keresünk a C_j klaszterben, mely egy belépő-párral kezdődik és egy kilépő-párral végződik.

Jelöljük a C_j klaszteren belüli belépő-, illetve kilépő-párok halmazát EN_j -vel és EX_j -vel. Az algoritmus megkeres és eltárol minden lehetséges (p_v, p_w) párt ($p_v \in EN_j, p_w \in EX_j$), melyek között van a klaszter összes csúcsát tartalmazó b-Hamilton-út. A 4.3. ábrán látható erre egy példa, ahol $p_v = (P, R)$ és $p_w = (S, T)$. Ekkor a b-Hamilton-út a klaszteren belül a következőképpen néz ki: $(P, R, \dots \{A \text{ } C_j \text{ klaszter összes többi csúcsa}\} \dots, S, T)$.

Ha valamelyik klaszterben az összes lehetséges belépő-, és kilépő-párt vizsgáltuk, és nem találtunk utat, akkor az algoritmus visszatér a kettes lépésre, és keres egy új kört a Q boríték-gráfban.

Két fontos megjegyzés:

- A már megtalált utakat az algoritmus eltárolja, így később, ha ugyanazt a klasztert ugyanazokkal a szomszédokkal vizsgálja, akkor már nincs szükség a keresés újbóli lefuttatására, ezzel jelentős időt spórolhatunk meg.
- Minden lehetséges belépő-, és kilépő-párt tárol az algoritmus, de nem tárolja az összes lehetséges útvonalat. Ha egy p_v belépő-, és egy p_w kilépő-pár között van b-Hamilton-út, akkor elég egyet eltárolni, mert a közbülső csúcsok sorrendje a keresés további menetét nem befolyásolja.

4. lépés Végső összeillesztés.

Az algoritmus eddig talált kört a Q boríték-gráfon, és talált ennek megfelelő b-Hamilton-utakat a klasztereken belül. Ez viszont még nem jelenti azt, hogy megtaláltuk a b-Hamilton-kört az egész gráfra vonatkozóan. Ugyanis lehet, hogy egy C_i klaszterből ki tudunk lépni a következő C_j klaszterbe a kettő között vezető boríték-élek egy H halmazán (ui. ezekbe tudtunk eljutni a C_i klaszter belépő-párjaiból minden csúcsot bejárva), de nem biztos, hogy a C_j klaszteren belül a H halmaz akár egy eleméből is el tudunk jutni a C_j és C_k klaszterek között vezető boríték-élek bármelyikébe is. A negyedik lépés feladat, hogy párosítsa a klasztereken belüli utakat a boríték-élekkel, és megtalálja a tényleges b-Hamilton-kört a teljes gráfban. Amennyiben az aktuális állapotból nem lehet ilyen kört találni, az algoritmus visszatér a kettes lépéshez, és új kört keres Q -n.

4.3. Klaszterező algoritmus

A klaszterező algoritmus célja az, hogy a gráfot olyan részgráfokra ossza, amik között relatív sok él fut a fürt méretéhez képest. Tehát olyan közel összefüggő részgráfokat hivatott készíteni, amikből meghatározott számú van (ami az algoritmus bemenő paramétere) és lehetőleg az egy fürtbe tartozó csúcsok száma nagyjából azonos. Az algoritmus 3 fő lépésből áll. Kezdetben teljes részgráfokat keresünk, majd ezek diszjunktá tételével kezdeti klasztereket alakítunk ki. Végül a kis klasztereket egyesítjük.

4.3.1. Szenzorok teljes részgráfokra osztása

Az eljárás első lépésében az a cél, hogy a gráf minden csúcsához megkeressük a lehető legnagyobb teljes részgráfot. Maximális méretű teljes részgráf keresése NP-teljes probléma. Ez pár csúcsnál nem olyan jelentős időbefektetés, de nagyobb gráfok esetében (akár már 30 csúcs) nem valósítható meg.

Mivel az optimális megoldás, azaz az egy csúcsához tartozó legnagyobb teljes részgráf megkeresése nehéz - és nem is feltétlenül szükséges feladat -, ezért ennek áthidalására egy

nem optimális megoldást adó teljes részgráf kereső algoritmussal dolgozik az eljárás első része. Az alkalmazott algoritmus egy mohó algoritmus, aminek a futási ideje csupán $O(n^2)$.

Az algoritmus egyetlen bemeneti paramétere maga a klaszterezni kívánt gráf, kimeneti paramétere pedig egy lista a teljes részgráfokról. Ennek a listának az elemei egy-egy halmaz, amelyben csúcsok vannak. A lista a bemenő gráf minden csúcspontjához tartalmaz egy teljes részgráfot.

Inicializáláskor minden részgráfot reprezentáló halmazban pontosan egy csúcs van, nevezetesen a lista k . részgráfja a gráf k . csúcsát tartalmazza.

Az inicializálás után az algoritmus végigmegy az összes eddigi részgráfon, majd egy ciklusban mindig kiválasztja azon csúcsok közül a legnagyobb fokszámút, amivel a jelenlegi részgráf teljes marad. Miután ezt megtalálta, hozzáadja azt a részgráfhoz. Ezt az eljárást addig folytatja, amíg létezik olyan csúcs, ami a kritériumoknak megfelel. Látható, hogy ezzel az eljárással n csúcsú gráf esetén n darab teljes részgráfot kapunk.

Az algoritmus pszeudo kódja a következő:

`complexSubgraphSearchAlgorithm(G)`

Input: $G = (V, E)$ – bemenő tartalmazó gráf

Output: $complete_subgraph = \{V_i \subseteq V\}$ – teljes részgráfok halmaza

$\forall v \in V : complete_subgraph_v = \{v\}$

$\forall v \in V$

$max = 1$

WHILE $max \neq 0$

$max = 0$

$p = -1$

$\forall w \in V \setminus complete_subgraph_v$

IF $connectionsToSubgraph(w, complete_subgraph_v, G) =$

$= |complete_subgraph_v|$

IF $degree(w, G) > max$

```

     $max = degree(w, G)$ 
     $p = w$ 
  FI
FI
END
IF  $max \neq 0$ 
     $complete\_subgraph_v = complete\_subgraph_v \cup \{w\}$ 
  FI
END
END

```

4.3.2. Teljes részgráfok diszjunktá tétele

Az előzőleg bemutatott algoritmus által generált részgráfokban egyértelműen lesznek átfedések. Egyetlen kivételt képez az az eset, amikor a gráfban egyetlen él sincs. Tehát a második lépés a klaszterezés során a teljes részgráfok diszjunktá tétele. Az algoritmus kimenete tehát teljes részgráfoknak olyan halmaza, ahol a halmaz elemei között nincs átfedés (közös csúcs).

Az algoritmus működése a következő: A legnagyobb klaszter csúcsait eltávolítja az összes többi klaszterből, ezen (legnagyobb) klasztert innentől kezdve ellenőrzöttnek tekintjük. Ezt az eljárást, kezdve a nem ellenőrzött klaszterek közötti legnagyobbal folytatja, amíg van nem ellenőrzött klaszter.

Az algoritmus pszeudo kódja:

$makeDisjointClusters(complete_subgraph, G)$

Input: $G = (V, E)$ – bemenő tartalmazó gráf

Input: $complete_subgraph = \{V_i \subseteq V\}$ – teljes részgráfok halmaza

Output: $disjoint_subgraph = \{V_i \subseteq V\}$ – diszjunkt teljes részgráfok halmaza

$disjoint_subgraph = \emptyset$

```

WHILE complete_subgraph  $\neq \emptyset$ 
    max = argmax(complete_subgraph)
    complete_subgraph = complete_subgraph  $\setminus \{max\}$ 
    disjoint_subgraph = disjoint_subgraph  $\cup \{max\}$ 
     $\forall s \in complete\_subgraph$ 
        s = s  $\setminus max$ 

```

A kimeneti érték ebben az esetben nem halmazban, hanem listában tárolódik, mivel a további feldolgozás során számít a sorrend. A makeDisjointClusters() eljárás után a disjoint_subgraph lista csökkenő sorrendbe rendezett lesz a tartalmazott fürtök méreteinek alapján.

4.3.3. Klaszterek egyesítése

A továbbiakban jelöljük x -el ($x \in N^+$) azt a számot, ahány fürtöt szeretnénk. A makeDisjointClusters() algoritmus által generált listában elképzelhető, hogy az első néhány (alacsony sorszámú) fürt sok csúcsot tartalmaz, míg a lista végén állók nagyon keveset. Ekkor az x -nél nagyobb sorszámú fürtöket célszerű megszüntetni oly módon, hogy az ezekbe tartozó szenzorokat megpróbáljuk hozzáfűzni az x -nél kisebb sorszámú klaszterekhez valamilyen fürt-tulajdonság optimalizálása alapján. Ez a tulajdonság lehet az átmérő, vagy a konnektivitás index, ami annak mértéke, hogy a kiegészített fürt mennyire marad teljes gráf.

Átmérő alatt annak az útnak a minimális hosszát értjük, amellyel bármely csúcsból bármely másikba el tudunk jutni. A konnektivitás index számítása a következő: ha a klaszter csúcsainak száma n , és a csatlakoztatott csúcsnak ezek közül k csúccsal van kapcsolata, akkor konnektivitás index $\frac{k}{n}$.

Az eljárást természetesen csak akkor kell lefuttatni, ha $x < |complete_subgraph|$

Az algoritmus pszeudo kódja, amely először konnektivitás index, majd átmérő alapján csatlakoztatja a csúcsokat, a következő:

```
clusterizeGBCAD(G,C,x,r)
```

Input: $G = (V, E)$ – bemenő tartalmazó gráf

Input: $C = \{V_i \subseteq V\}$ – klaszterek listája

Input: $x \in N^+$ – a kívánt klaszterek száma

Input: $r \in R$ – legkisebb konnektivitás index, amelynél még hozzáfűzünk egy csúcsot

Output: $Co = \{V_i \subseteq V\}$ – klaszterek listája az eljárás után

$$Co = \{\bigcup_{i=1}^x \{C_i\}\}$$

$$Ct = \{\bigcup_{i=x+1}^n \{C_i\}\}$$

$$\forall i \in N^+ : i = 1..|Ct| - x$$

$$\forall v \in Ct_i$$

$$p = \min \text{Diameter}(\{c \cup \{v\} : c \in Co, \text{connectionsToSubgraph}(v, c, G) \geq |c| * r\})$$

$$IF \ p \neq \emptyset$$

$$Co_p = Co_p \cup v$$

$$Ct_i = Ct_i \setminus v$$

$$FI$$

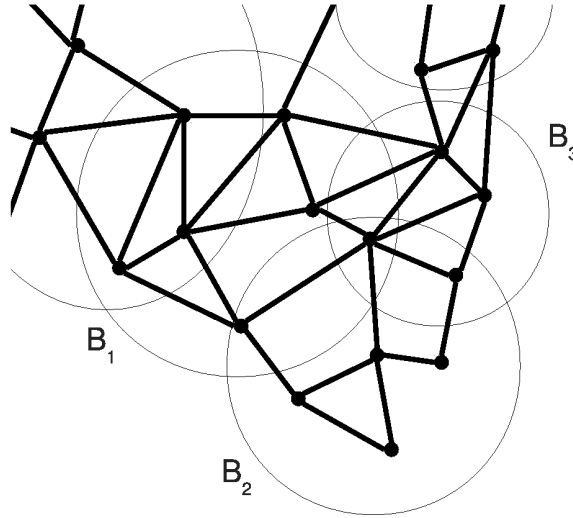
$$Co = Co \cup_{i=1}^{|Ct|} \{Ct_i\}$$

A $\min \text{Diameter}(C = \{V_i \subseteq V\})$ függvény visszatérési értéke annak legkisebb átmérőjű klaszternek a sorszáma, amit a paraméterben átadtunk

A valósághű szenzorhálózaton tesztelt eredményeink alapján ezek a klaszterek sok esetben megfelelnek az elvárásoknak, de jó alapul szolgálhatnak egy manuális klaszterezéshez is. A javasolt klaszterezési eljárás időigénye 200 csúcs esetén a másodperc tört része, 400 csúcs esetén kb. egy perc, míg 1000 csúcsnál már 20 perc is lehet.

4.4. Buborékos keresés

A buborékos keresés a klaszter alapú keresés egy továbbfejlesztett változata. Alapötlete a következő felismerésen alapul: valós hálózatokat leíró kapcsolati gráfokban a fizikailag



4.4. ábra. Csúcsok buborékokban

közel elhelyezkedő csomópontok között nagy valószínűséggel jó a kommunikációs link, míg távoli csúcsok esetén ez kisebb valószínűséggel van így (ellentétben pl. a véletlen-, vagy a vezetékes hálózatokat leíró gráfokkal). Amennyiben egy ilyen gráfban próbálunk Hamilton, vagy b-Hamilton kört keresni, akkor a keresés során célszerű nem "csapongani", hanem jobb bejárni a közeli csúcsokat és csak utána távolabbi területekre továbblépni. A keresés tehát egyrészt *lokális* tulajdonságokat mutat - hiszen a gráf aktuálisan bejárt környezetében kersünk csak, másrészt a keresés ezen a helyen *kimerítő* jellegű - hiszen az adott környezetet teljesen be akarjuk járni a továbblépés előtt. A javasolt keresés ezt a két heurisztikát alkalmazza. A lokális heurisztikája segíti a hatékony, gyors keresést, a kimerítés elve pedig a zsákutcák elkerülésében segít. Az algoritmus a fenti elvek mellett visszalépéses alapon működik, tehát amennyiben a buborékok által meghatározott keresési térben van megoldás, azt biztosan meg is találja.

Ebben a megoldásban klaszterek helyett a felhasználó definiál részgráfokat. Mivel a legegyszerűbb alakzat a kijelölésre a kör, ezért az implementáció köröket használ, innét a buborék megnevezés. A 4.4. ábra a koncepció alapötletét mutatja be. A buborékok sorozatát a következőképpen jelöljük: $B_1, B_2, B_3, \dots, B_k$. A klaszter alapú kereséssel ellentétben a részgráfok nem feltétlenül diszjunktak.

A keresés a buborékok segítségével történik. A b-Hamilton kört szekvenciálisan építjük, az algoritmus egy b-Hamilton utat bővít újabb csúcsok hozzáadásával. Az algoritmus a csúcsokat a megadott buborékokból sorban választja ki, ily módon a buborékok irányítják a keresést.

Definíció: egy buborék (B_i) *fedett*, ha minden olyan csúcs, amit tartalmaz a buborék ($v_j \in B_i$), de nem tartalmaz a soron következő buborék ($v_j \notin B_{i+1}$) már benne van az eddigi útban.

Alternatív módon egy engedékenyebb definíció a következő:

Definíció: egy buborék (B_i) *gyengén-fedett*, ha minden olyan csúcs, amit tartalmaz a buborék ($v_j \in B_i$), de nem tartalmaz a soron következő összes buborék ($v_j \notin B_{i+1} \cup B_{i+2} \cup B_{i+3} \cup \dots \cup B_{i+n}$) már benne van az eddigi útban.

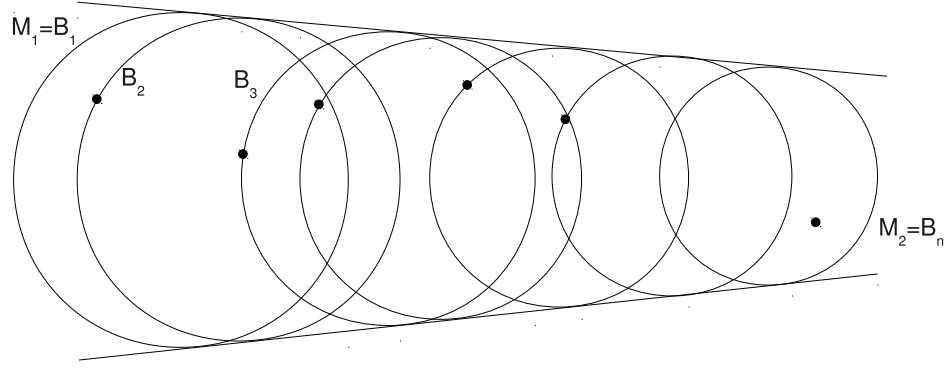
Definíció: az algoritmus *belépett egy buborékba* (B_i) , ha az előtte levő buborék (B_{i-1}) fedett (vagy alternatív módon gyengén-fedett) és legalább egy csúcs ($v_j \in B_i$) már benne van az eddigi útban.

Definíció: a *nyílt csúcsok halmaza* azon csúcsok, amik nincsenek benne az eddigi útban és abban a részgráfban vannak, amikbe az algoritmus be tud lépni, vagy már belépett.

A b-Hamilton kör kereső algoritmus egymás után megpróbálja lefedni az összes buborékot a következő egyszerű szabályt betartva: egy buborékba (B_i) nem lép be addig, míg az előtte levő buborék (B_{i-1}) nem fedett (vagy alternatív módon gyengén-fedett).

Az algoritmus a kör következő szegmensét abból a buborékból próbálja meg felépíteni, amibe belépett, így az algoritmus megfelelően kicsi buborékok esetén hatékony és gyors marad. Természetesen a kezdő buborék megválasztása is lényeges az algoritmus futása szempontjából.

A felhasználónak először ki kell jelölnie a buborékokat és meg kell határoznia ezek sorrendjét $(M_1, M_2, M_3, \dots, M_m)$. A buborékok a középpontjukkal és a sugarukkal definiáltak. A felhasználó kiválaszthatja az összes buborékot, amit fontosnak talál, vagy dönthet úgy, hogy a fő buborékokon kívül generáltat a rendszerrel interpolált



4.5. ábra. Interpolált buborékok ($B_1, B_2, \dots, B_n$) fő buborékok közt: M_1 és M_2

buborékokat is.

Ha a felhasználó az utóbbi lehetőséget választja, akkor az algoritmus automatikusan létrehozza azokat. Az interpolált buborékok középpontja a két fő buborék középpontját összekötő szakaszon lesz. Sugaruk pedig akkora lesz, hogy bármely kiválasztott interpolált buborékhoz és a két fő buborékhoz lehessen közös érintőt húzni. Minden interpolált buborék úgy jön létre, hogy pontosan egy olyan csúcs (v) létezik, amelyre igaz $v \in B_{i-1}, v \notin B_i$, ahogy azt a 4.5. ábra mutatja.

Amennyiben adott két fő buborék (M_1, M_2) és egy csúcs (P_p), ahogy azt a 4.6. ábra mutatja, akkor a P_p csúcsához tartozó buborék kiszámítható. Ha $|P_1P| = q * |P_1P_2|$ és $|PP_2| = (1 - q) * |P_1P_2|$ akkor a következő egyenletrendszer áll fenn:

$$R = (1 - q) * R_1 + q * R_2 \quad (4.1)$$

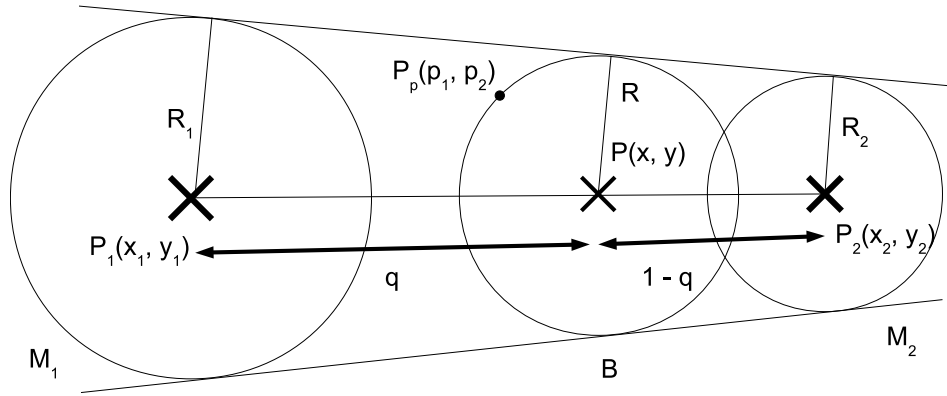
$$x = (1 - q) * x_1 + q * x_2 \quad (4.2)$$

$$y = (1 - q) * y_1 + q * y_2 \quad (4.3)$$

A kör egyenlete (interpolált buborék):

$$(x - p_1)^2 + (y - p_2)^2 = R^2 \quad (4.4)$$

A (4.1)-(4.4) egyenletrendszerből egy másodfokú egyenlet adódik q -ra, melynek nagyobbik megoldása adja q -t. Ezután a kör paraméterei kiszámíthatóak felhasználva a (4.1), (4.2), és (4.3) egyenleteket. A másodfokú egyenlet csak egy eredményt ad, ha P_p a közös érintőkön helyezkedik el.



4.6. ábra. Egy interpolált buborék kiszámítása

Miután a fő - és opcionálisan az interpolált buborékok - már adottak a gráfban, a keresési algoritmus elkezdődhet a nyílt csúcsok halmazán. Kezdetben a nyílt csúcsok halmazát a kezdő buborékban található csúcsok alkotják. Az algoritmus felépíti a b-Hamilton kört a nyílt csúcsok halmazából, amely tartalma folyamatosan változik a fent említett szabály szerint. A `getOpenVertices()` függvény a nyílt csúcsok halmazát nyújtja az algoritmus számára. A pseudo kód a következő:

`findBubbleCycle()` // b-Hamilton kör keresése

Input: $G = (V, E)$ // gráf csúcsokkal (V) és élekkel (E)

$(V = \{v_i : i = 1..k\})$

$(E \subseteq \{(v_1, v_2) : v_1, v_2 \in V\})$

solution // Találtunk már megoldást?

SET // a `branch()` állapotainak halmaza, globális változó

$n = |V|$ // csúcsok száma, globális változó

C // a körben résztvevő csúcsok listája

count // a körben résztvevő csúcsok száma

State : $S = (C, count)$

closed_vertices = $\emptyset$

open_vertices = `getOpenvertices(closed_vertices)`

solution = *false*

vertex = egy tetszőleges elem az *open_vertices* halmazból

for $\forall v \in V : (vertex, v) \in E$

$SET = SET \cup State([vertex|v], 2)$

done

while($(SET \neq \emptyset) \wedge (found = false)$)

S = egy tetszőleges elem a *SET* halmazból

$found = branch(G, S)$

done

Hogyha találtunk megoldást, használjuk.

Összes megmaradt állapot törlése a *SET* halmazból.

end.

branch() // az aktuális állapot vizsgálata

Input: $G = (V, E)$ // gráf csúcsokkal (V) és élekkel (E)

Input: $S = (C, count)$ // az állapot

Output: *solution* // találtunk megoldást ebből a pontból?

solution = *false*

open_vertices = *getOpenvertices(vertices)*

for $\forall i \in open_vertices$

if($((C[end - 1], i) \in E \wedge (C[end], i) \in E)$)

if($count = n - 1$)

if($((i, C[1]) \in E \wedge (i, C[2]) \in E \wedge (C[end], C[1]) \in E)$)

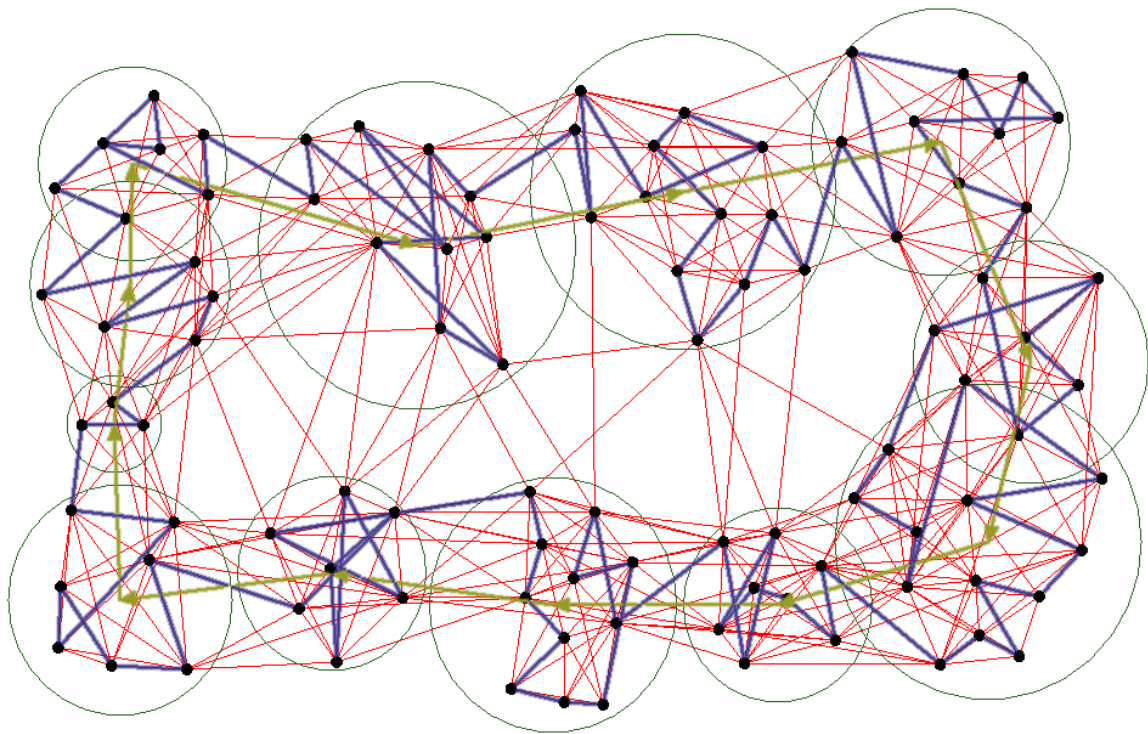
solution = *true*

return

fi

else

$SET = SET \cup \{State([C|i], count + 1)\}$



4.7. ábra. Teszthálózat 100 csúccsal (fekete pöttyök). Piros élek: kommunikációs linkek, zöld körök: fő buborékok, barna nyílak: fő buborékok sorrendje, kék élek: b-Hamilton kör

fi
fi
done
end.

A 4.7. ábra az algoritmus működését illusztrálja egy 100 csúcsot tartalmazó teszthálózatban. A csúcsokat fekete körök, a kapcsolatokat piros élek jelölik. A buborékokat zöld körök mutatják, melyek között barna nyílak mutatják a buborékok sorrendjét. Az algoritmus által megtalált b-Hamilton kör elsődleges éleit (ami egy egyszerű Hamilton-kör) vastag kék élek mutatják. Az jobb áttekinthetőség kedvéért a másodlagos éleket külön nem jelöltük.

5. fejezet

Összehasonlító értékelés

5.1. Tesztelési módszerek

Ebben a fejezetben először bemutatjuk a tesztelés során használt hálózatokat, majd áttekintjük a tesztelés során alkalmazott algoritmusokat, végül ismertetjük a tesztelés menetét.

5.1.1. Teszthálózatok

A mérések során 10 darab teszthálózatot használtunk, melyeken a bemutatott algoritmusokat teszteltük. A teszthálózatok az egyszerűbbektől a bonyolultakig széles skálán mozognak, igyekeztünk olyan típusú feladatokat generálni, melyek a valós körülményeket is jól modellezik. Ily módon a gráfokat mindig a fizikai topológiával együtt adjuk meg, és egyes algoritmusok így is használják ezeket. A teszthálózatok a következők:

- "12_kör": A gráf 12 csúcsot tartalmaz, melyeken keresztül egy b-Hamilton-kör fut az 5.1. ábra szerint. A probléma kis méretű, megoldása az emberi szemlélő számára triviális. Gyűrű struktúrát modellez.
- "12_teljes": 12 csúcsú teljes gráf, melyet az 5.2. ábra illusztrál. Kis méretű probléma, melyben várhatóan könnyű megoldást találni. Nagy sűrűségű hálózatot modellez.

- "100_kör": A gráf 100 csúcsot tartalmaz, melyeken keresztül egy b-Hamilton-kör fut az 5.3. ábra szerint. A probléma nagy méretű, de megoldása az emberi szemlélő számára triviális. Nagy méretű gyűrű struktúrát modellez.
- "100_teljes": 100 csúcsú teljes gráf, melyet az 5.4. ábra illusztrál. Nagy méretű probléma, melyben várhatóan könnyű megoldást találni. Nagy sűrűségű, nagy méretű hálózatot modellez.
- "3k10": A gráf 36 csúcsot tartalmaz az 5.5. ábra szerint. A 36 csúcsból háromszor 10 csúcs teljes gráfot alkot, melyeket 2-2 csúcsból álló útvonal köt össze. Modellezhet például egy három, egymástól viszonylag elszigetelt termet tartalmazó épületet.
- "10k10": A gráf 120 csúcsot tartalmaz az 5.6. ábra szerint. A 120 csúcsból tízszer 10 csúcs teljes gráfot alkot, melyeket 2-2 csúcsból álló útvonal köt össze. Modellezhet például egy tíz, egymástól viszonylag elszigetelt termet tartalmazó épületet. A "3k10" példa nagy méretű változata.
- "10c10": A gráf 120 csúcsot tartalmaz az 5.7. ábra szerint. A "10k10" példabeli teljes részgráfok helyett némileg él-ritkított csoportokat tartalmaz. A "10k10" példával ellentétben a csoportokon belüli keresés kevésbé triviális, bár az emberi szemlélő számára a feladat még mindig egyszerű.
- "random33": A gráf 33 darab, véletlen elhelyezkedésű csúcsot tartalmaz az 5.8. ábra szerint. A gráf éleit távolság alapon generáltuk: "közeli" csúcsok között vezet él, "távoliak" között nem.
- "random100": A gráf 100 darab, véletlen elhelyezkedésű csúcsot tartalmaz az 5.9. ábra szerint. A gráf éleit távolság alapon generáltuk: "közeli" csúcsok között vezet él, "távoliak" között nem.
- "I9": A Pannon Egyetem I épületének 9. emeletén végzett mérések alapján generált gráf. A gráf az 5.10. ábrán látható. Az eszközök RFA1 típusú rádióchipet használtak, amelyek 802.15.4 szabvány szerint működnek. Az alkalmazott

név	csúcsok száma	élek száma	minimum fokszám	maximum fokszám	átlagos fokszám	átmérő
12_kör	12	24	4	4	4	3
12_teljes	12	66	11	11	11	1
100_kör	100	200	4	4	4	25
100_teles	100	4950	99	99	99	1
3k10	36	156	4	11	8.7	5
10k10	120	522	4	11	8.7	16
10c10	120	474	4	11	7.9	16
random33	33	115	4	12	7.0	6
random100	100	828	7	29	16.6	6
I9	87	382	5	16	8.7	7

5.1. táblázat. A tesztelt hálózatok

frekvencia 2.4 GHz volt. A rádiókat maximális teljesítményen üzemeltettük. Az 5.10. ábrán látható gráf már csak a jó minőségű kommunikációs linkeket tartalmazza.

A tesztelt hálózatokat az 5.1. táblázat tartalmazza jellemző paramétereikkel együtt.

5.1.2. Tesztelt algoritmusok

A tesztek során a következő algoritmusokat teszteltük:

- Brute force. A tesztelt algoritmus leírását a 4.1. fejezet tartalmazza.
- Karmakar. A tesztelt algoritmus a [23] algoritmus általunk készített módosítása. Az eredeti algoritmust úgy módosítottuk, hogy az a Hamilton kör helyett b-Hamilton kört keressen. Az algoritmus heurisztikáin nem változtattunk.
- Dharwadker. A tesztelt algoritmus a [22] algoritmus általunk készített módosítása. Az eredeti algoritmust úgy módosítottuk, hogy az a Hamilton kör helyett b-Hamilton kört keressen. Az algoritmus heurisztikáin nem változtattunk.
- Klaszter. A tesztelt algoritmus leírását a 4.2. fejezet tartalmazza.

- Buborék. A tesztelt algoritmus leírását a 4.4. fejezet tartalmazza.

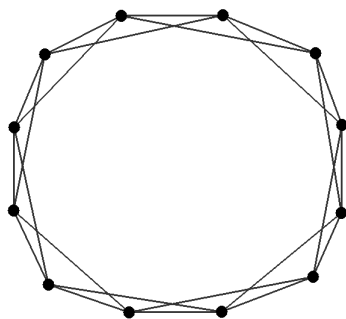
5.1.3. Mérési módszerek

A tesztek során minden algoritmust lefuttattunk minden teszt hálózatra. A vizsgált algoritmusok determinisztikusak, így statisztikai jellemzők mérésének egyazon gráfra nem lenne értelme. Az algoritmusok azonban érzékenyek a gráf adatszerkezetének megadására, pontosabban ebben a gráf csúcsainak sorrendjére. Ezért a tesztek során a teszthálózatokban véletlenszerűen permutáltuk a csúcsok sorszámait (minden hálózatra 51 különböző permutációt generáltunk), de természetesen a hálózat egyéb tulajdonságain nem változtattunk. Minden algoritmust minden teszthálózat minden permutációján lefuttatuk és a futási időket megmértük. Amennyiben az adott teszt 5 percen belül nem vezetett eredményre, a futást megszakítottuk. Ezen esetben a futást sikertelennek minősítettük. A közös teszthálózatra vonatkozó tesztekben a következő statisztikai jellemzőket számítottuk ki:

- találati arány (%): az 51 permutáció hány százalékában tudott az algoritmus 5 percen belül megoldást adni.
- minimum (ms): a sikeres futások közül a legrövidebb ideje
- maximum (ms): a sikeres futások közül a leghosszabb ideje
- átlag (ms): a sikeres futások átlagos ideje
- szórás (ms): a sikeres futások ideinek szórása

5.2. Tesztelési eredmények

Ebben a fejezetben ismertetjük a részletes tesztelési eredményeket minden teszthálózatra, majd algoritmusonként összegezzük a tesztelési eredményeket, tapasztalatokat.



5.1. ábra. A "12_kör" teszhálózat

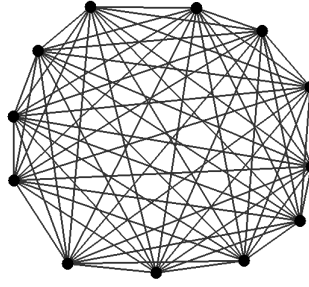
algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	100	1	13	4.4	3.1
Karmakar	0	-	-	-	-
Dharwadker	90	9	12	10	0.7
klaszter	100	1	31	13	8.4
buborék	100	0	13	4.7	3.4

5.2. táblázat. A "12_kör" teszt eredményei

5.2.1. "12_kör" teszteredmények

A teszhálózat az 5.1. ábrán látható, a mérési eredményeket az 5.2. táblázat tartalmazza.

A brute force algoritmus elvárásainknak megfelelően gyorsan futott le minden permutációra, mert a gráf kevés csúcsot és kevés élet tartalmaz. A Karmakar algoritmus egyik permutációra sem futott le 5 perc alatt, mivel a gráf a csúcsok számához képest arányaiban kevés élet tartalmaz. A Dharwadker algoritmus gyorsan lefutott minden permutációra, mivel a futási ideje polinom idejű, viszont némely esetben nem találta meg a létező Hamilton-kört. A klaszter alapú keresés lassabb volt a brute force algoritmusnál, de ezt a keresést nem kis méretű hálózatokra terveztük, ahhoz túl komplikált. A buborék alapú algoritmus gyorsan megtalálta a kört, de ezt is célszerűbb inkább nagyobb problémákra használni.



5.2. ábra. A "12_teljes" teszhálózat

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	100	1	36	7.2	6.6
Karmakar	100	3	152	7.2	20.4
Dharwadker	100	5	17	13.3	2
klaszter	100	11	48	27	9.8
buborék	100	0	10	3.3	2.6

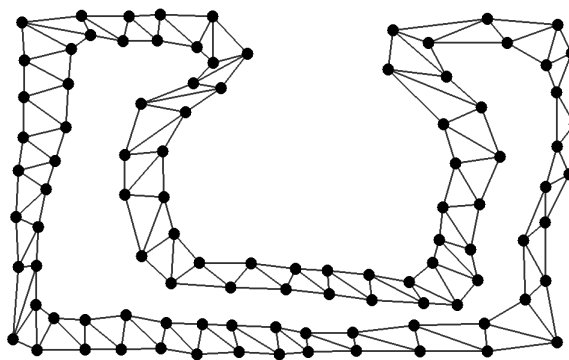
5.3. táblázat. A "12_teljes" teszt eredményei

5.2.2. "12_teljes" teszteredmények

A teszhálózat az 5.2. ábrán látható, a mérési eredményeket az 5.3. táblázat tartalmazza. A brute force, Karmakar és Dharwadker algoritmusnak nem volt más dolga, mint ellenőriznie a csúcsokat az adott sorrendben, így lineáris időben lefutottak. Ezért a mért idő nagyon kevés volt minden esetben. A klaszter alapú algoritmus is gyorsan lefutott, mivel az egyes klaszterek között kevés borítékél van a kevés csúcs és él miatt. A buborék alapú algoritmus ebben az esetben is gyorsan eredményre jutott, habár ilyen kicsi gráfok esetében inkább hátrány az emberi segítség, mint előny.

5.2.3. "100_kör" teszteredmények

A teszhálózat az 5.3. ábrán látható, a mérési eredményeket az 5.4. táblázat tartalmazza. A brute force algoritmus elvárásainknak megfelelően gyorsan futott le minden permutációra, mert bármely pontból egyértelműen jó döntés hozható a következő csúcsot illetően. A Karmakar algoritmus egyik permutációra sem futott



5.3. ábra. A "100_kör" teszhálózat

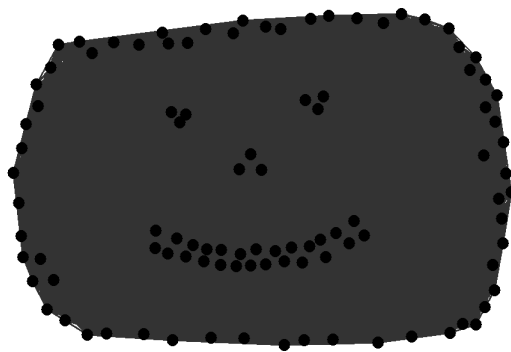
algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	100	12	44	27	8.1
Karmakar	0	-	-	-	-
Dharwadker	100	98	116	105	3.7
klaszter	100	16	46	30	7.8
buborék	100	6	35	14.1	4.9

5.4. táblázat. A "100_kör" teszt eredményei

le 5 perc alatt, mivel a gráf a csúcsok számához képest arányaiban kevés élet tartalmaz. A klaszter alapú keresés lassabb volt a brute force algoritmusnál. A buborék alapú algoritmus gyorsan megtalálta a kört, de ezt is célszerűbb kevésbé egyértelmű problémákra használni.

5.2.4. "100_teljes" teszteredmények

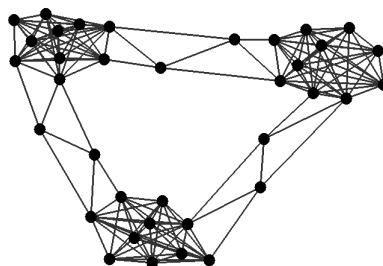
A teszhálózat az 5.4. ábrán látható, a mérési eredményeket az 5.5. táblázat tartalmazza. A brute force, Karmakar és Dharwadker által implementált algoritmusnak itt is egyszerű dolga volt. A klaszter alapú algoritmus egy esetben sem futott le időben, mivel az egyes klaszterek között rengeteg boríték-él van a sok csúcs és él miatt. A buborék alapú algoritmus ebben az esetben is gyorsan eredményre jutott, habár ilyen sűrű gráfok esetében is inkább hátrány az emberi segítség, mint előny.



5.4. ábra. A "100_teljes" teszhálózat

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	100	47	75	61.7	6.6
Karmakar	100	6	48	8.9	7.9
Dharwadker	100	2360	2533	2403.3	35.1
klaszter	0	-	-	-	-
buborék	100	6	35	14.1	4.9

5.5. táblázat. A "100_teljes" teszt eredményei



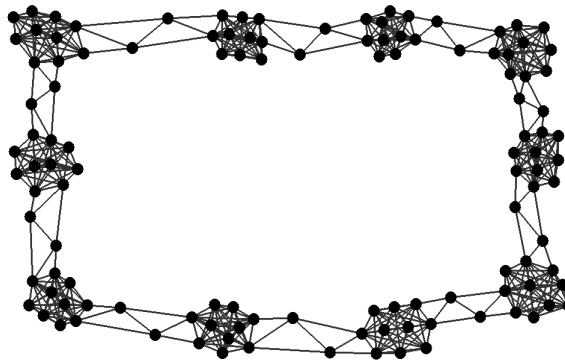
5.5. ábra. A "3k10" teszhálózat

5.2.5. "3k10" teszteredmények

A teszhálózat az 5.5. ábrán látható, a mérési eredményeket az 5.6. táblázat tartalmazza. Az eredmények azt mutatják, hogy a brute force algoritmusnak ilyen méretben már nem elég az 51 permutáció, ahhoz hogy biztosan legyen köztük legalább egy olyan sorrend, amiben a keresés jól teljesít. Karmakar algoritmusa nem jutott eredményre 5 percen belül. A Dharwadker-algoritmus minden esetben talált megoldást

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	0	-	-	-	-
Karmakar	0	-	-	-	-
Dharwadker	100	27	44	33	4.8
klaszter	100	1448	1515	1472.9	12.4
buborék	100	131	157	144.3	5.9

5.6. táblázat. A "3k10" teszt eredményei



5.6. ábra. A "10k10" teszhálózat

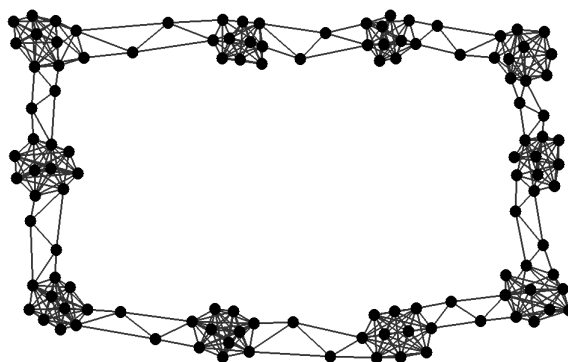
erre a gráfra. A klaszter alapú algoritmus is jól teljesített, minden esetben megtalálta a kört, de jóval lassabban, mint az ugyancsak sikeres buborék alapú keresés.

5.2.6. "10k10" teszteredmények

A teszhálózat az 5.6. ábrán látható, a mérési eredményeket az 5.7. táblázat tartalmazza. Az eredmények azt mutatják, hogy a brute force algoritmusnak ilyen méretben még inkább nem elég az 51 permutáció, ahhoz hogy biztosan legyen köztük legalább egy olyan sorrend, amiben a keresés jól teljesít. Karmakar algoritmusa itt sem jutott eredményre 5 percen belül. A Dharwadker algoritmus minden esetben talált megoldást erre a gráfra. A klaszter alapú algoritmus is jól teljesített, minden esetben megtalálta a kört, de itt is jóval lassabban, mint a buborék alapú keresés.

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	0	-	-	-	-
Karmakar	0	-	-	-	-
Dharwadker	100	206	244	220.6	8.6
klaszter	100	7297	7843	7454.9	100.4
buborék	100	1106	4340	2702.3	725.3

5.7. táblázat. A "10k10" teszt eredményei



5.7. ábra. A "10c10" teszhálózat

5.2.7. "10c10" teszteredmények

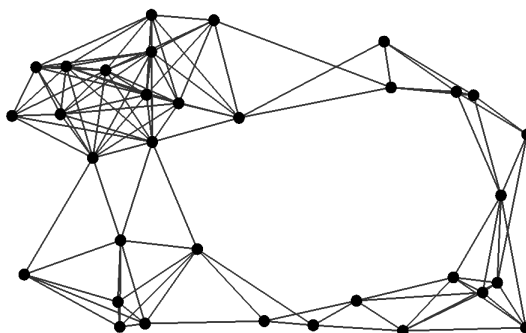
A teszhálózat az 5.7. ábrán látható, a mérési eredményeket az 5.8. táblázat tartalmazza. Ezt a méretű problémát sem a brute force, sem a Karmakar algoritmus nem tudta 5 percen belül megoldani. Ebben az esetben a Dharwadker algoritmus egyetlen permutációra sem talált megoldást, ami azzal magyarázható, hogy a ritkább csomósodási helyeken a csomósodás bejárása helyett a továbbhaladás mellett döntött; a rossz döntést a cserélő heurisztikák már nem tudtak helyrehozni. A klaszter alapú algoritmus jól teljesített, minden esetben megtalálta a kört, bár jóval lassabban, mint a buborék alapú keresés.

5.2.8. "random33" teszteredmények

A teszhálózat az 5.8. ábrán látható, a mérési eredményeket az 5.9. táblázat tartalmazza. Az eredmények azt mutatják, hogy a brute force algoritmusnak ilyen

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	0	-	-	-	-
Karmakar	0	-	-	-	-
Dharwadker	0	-	-	-	-
klaszter	100	3125	4823	3251.8	234.0
buborék	100	594	797	625.4	31.9

5.8. táblázat. A "10c10" teszt eredményei

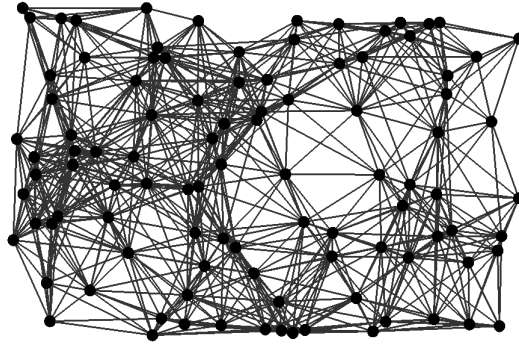


5.8. ábra. A "random33" teszhálózat

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	80	6	252315	55491.8	77159.9
Karmakar	0	-	-	-	-
Dharwadker	100	14	34	23.1	4.8
klaszter	100	180	235	210.3	13.6
buborék	100	42	61	48	4.1

5.9. táblázat. A "random33" teszt eredményei

méretben igencsak számít a csúcsok számozása: bizonyos esetekben 5 perc alatt sem találta meg, míg bizonyos esetekben szinte azonnal megtalálta a kört. A Karmakar algoritmus itt is elbukott. A Dharwadker algoritmus minden permutációban jól teljesített. A klaszteres és a buborékos algoritmus sikeresen lefutott a példa minden permutációjára.



5.9. ábra. A "random100" teszhálózat

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	0	-	-	-	-
Karmakar	0	-	-	-	-
Dharwadker	100	25	298	249.5	17.8
klaszter	0	-	-	-	-
buborék	100	2790	3104	2885.2	77.6

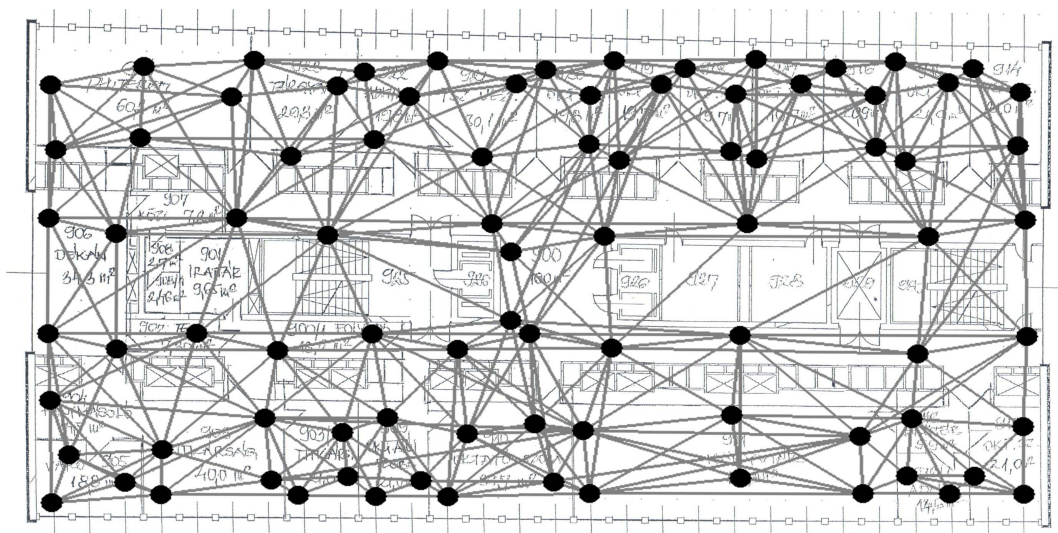
5.10. táblázat. A "random100" teszt eredményei

5.2.9. "random100" teszteredmények

A teszhálózat az 5.9. ábrán látható, a mérési eredményeket az 5.10. táblázat tartalmazza. A brute force algoritmus és a Karmakar algoritmus itt is elbukott. A Dharwadker algoritmus itt minden permutációban jól teljesített. A klaszter alapú algoritmus minden esetben túllépte az 5 percet a sok borítékél miatt. A buborékos algoritmus sikeresen lefutott a példán minden permutációra.

5.2.10. "I9" teszteredmények

A teszhálózat az 5.10. ábrán látható, a mérési eredményeket az 5.11. táblázat tartalmazza. A brute force és a Karmakar algoritmus az elvárásoknak megfelelően nem futott le az időkorláton belül erre a sok csúcsot tartalmazó gráfra. A klaszter alapú és a buborékos keresés gyorsan lefutott, viszont nem volt triviális a buborékok megadása, a többi teszthez képest több munkát igényelt a felhasználó részéről. A Dharwadker



5.10. ábra. A "I9" teszhálózat

algoritmus	találat (%)	minimum (ms)	maximum (ms)	átlag (ms)	szórás (ms)
brute force	0	-	-	-	-
Karmakar	0	-	-	-	-
Dharwadker	1	313	313	313	-
klaszter	100	150	199	174.5	12.2
buborék	100	5	20	10.9	3.4

5.11. táblázat. A "I9" teszt eredményei

algoritmus mindössze egy permutációra talált megoldást az 51 közül, ami jól mutatja, hogy a valós problémák sokszor nehezebbek az egyébként jól működő heurisztikus algoritmusok számára, mint a "tenyésztett" példák.

5.3. Értékelés

Az 5.2. fejezetben bemutatott tesztek alapján az egyes algoritmusok a különféle teszhálózatokon elért találati arányait az 5.13. táblázat tartalmazza. Az algoritmusok sikeres futtatásainak átlagos futási idejét és szórását az 5.12. táblázat tartalmazza.

A brute-force keresési algoritmus kis méretű (<15-20) gráfokra használható, nagyobb méretű gráfokra már nem képes kivárható időn belül megtalálni a

	brute force	Karmakar	Dharwadker	klaszter	buborék
12_kör	100	0	90	100	100
12_teljes	100	100	100	100	100
100_kör	100	0	100	100	100
100_teljes	100	100	100	0	100
3k10	0	0	100	100	100
10k10	0	0	100	100	100
10c10	0	0	0	100	100
random33	80	0	100	100	100
random100	0	0	100	0	100
I9	0	0	1	100	100

5.12. táblázat. találati arányok

	brute force	Karmakar	Dharwadker	klaszter	buborék
12_kör	4.4/3.1	-/-	10/0.7	13/8.4	4.7/3.4
12_teljes	7.2/6.6	7.2/20.4	13.3/2	27/9.8	3.3/2.6
100_kör	27/8.1	-/-	105/3.7	30/7.8	8.1/4.8
100_teljes	61.7/6.6	8.9/7.9	2403.3/35.1	-/-	14.1/4.9
3k10	-/-	-/-	33/4.8	1472.9/12.4	144.3/5.9
10k10	-/-	-/-	220.6/8.6	7454.9/100.4	2702.3/725.2
10c10	-/-	-/-	-/-	3251.8/234	625.4/31.9
random33	55491.8/77159.9	-/-	23.1/4.8	210.3/13.6	48/4.1
random100	-/-	-/-	249.5/17.8	-/-	2885.2/77.6
I9	-/-	-/-	313/-	174.5/12.2	10.9/3.4

5.13. táblázat. átlag/szórás

Hamilton-kört. A keresést nagy mértékben befolyásolja a csúcsok számozása a gráfban. Ezt jól szemlélteti a "random33" példa.

A Karmakar keresés b-Hamilton-kör keresésére kiegészített verziója a teljes gráfok kivételével a többi esetben nem futott le a hatáidőn (5 perc) belül. Ennek oka az lehet, hogy az algoritmus egyszerű Hamilton körök keresésére kidolgozott gráftranszformációs heurisztikái a jóval több kényszert tartalmazó b-Hamilton kör esetén már nem

hatékonyak.

A Dharwadker algoritmus minden esetben gyorsan lefutott, ennek oka az algoritmus polinom idejű futási ideje. Ellenben nem talált minden tesztesetben b-Hamilton-kört. Az egyik teszt, ahol néha nem talált, a "12_kör", ahol bizonyos permutációknál a csúcsok számozása miatt már a legelső lépéseknél eldőlt, hogy nem lesz találat. A másik teszt, ahol a heurisztika megbukott, a "10c10". Ebben a "10k10" esetből származtatott példában a nem szabályos gócpontokban az algoritmus heurisztikája már könnyen eltéved.

A klaszter alapú keresést nagy méretű, de ritka gráfokra terveztük, amelyekre hatékonyan működik. Ennek megfelelően kis méretű gráfokra is működik, de lemarad a többi algoritmustól a felesleges számolások miatt. Nagy méretű, de sűrű gráfokra az keresés szintén sokáig dolgozik, mert a klaszterek között sok boríték-él megy, és a klaszteren belüli útkeresés is lassú a sok lehetőség miatt.

A buborékos keresés hatékonynak bizonyult minden problémára. Ennek oka, hogy a buborékok megadása a felhasználó feladata, aki ránézésre jó eséllyel el tudja dönteni, hogy a Hamilton-körnek nagyjából milyen irányba kell haladni. Minél jobb segítséget ad a felhasználó annál gyorsabban talál b-Hamilton-kört az algoritmus. Ellenben nem megfelelő segítség esetén könnyen elképzelhető, hogy nem találja meg a kört a gráfban. A tesztek során minden példára sikerült olyan beállításokat megadni, hogy találjon az algoritmus kört a futás során.

6. fejezet

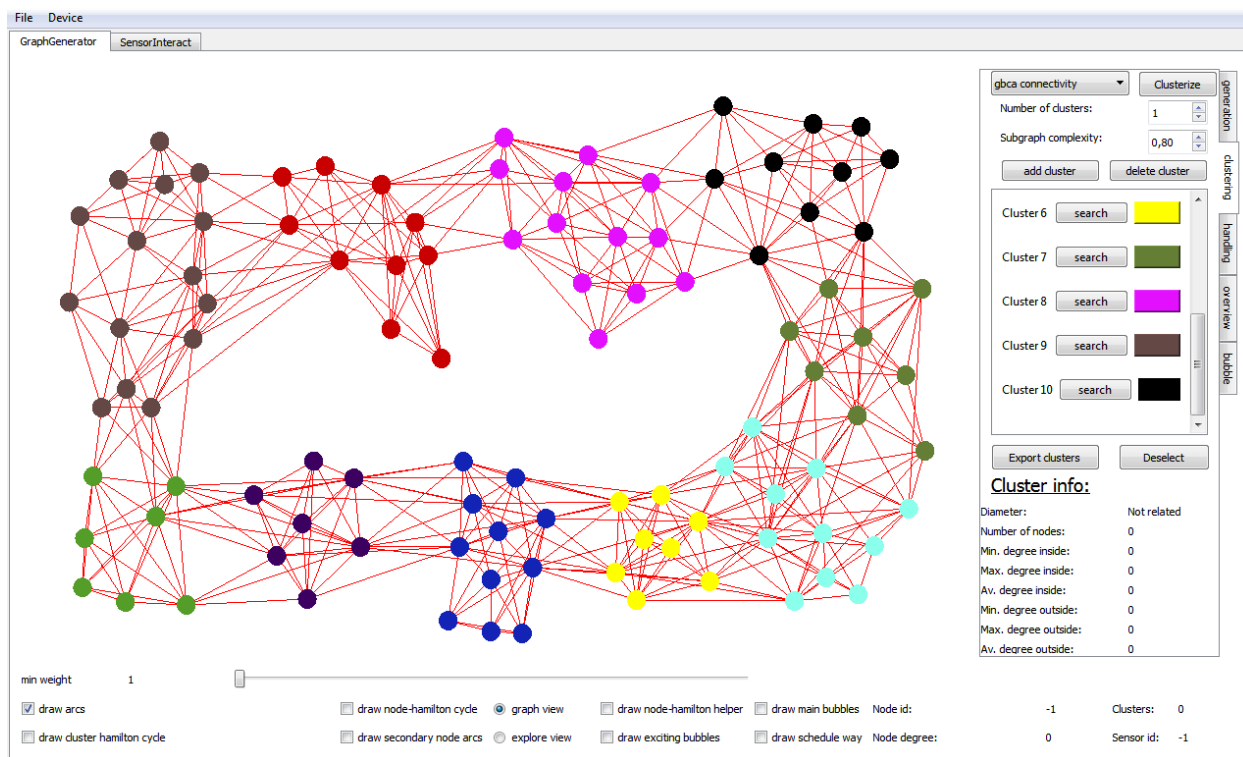
Gyakorlati felhasználás

Az algoritmusok tesztelésére, valamint ezek valós hálózatokban történő alkalmazásának támogatására elkészítettünk egy szoftvert, amelyben sikeresen implementálásra kerültek a szakmai munkában tárgyalt algoritmusok. A szoftvert két fő részre lehet osztani, melyek külön fülként jelennek meg az alkalmazásban (lásd a 6.1. ábrát): az egyik fül (GraphGenerator) a dolgozatban tárgyalt gráfkezelő algoritmusokat és azok kisegítő funkcióit tartalmazza (pl. gráfok generálása, szerkesztése), míg a másik fül (SensorInteract) a valós szenzorhálózatokkal való kapcsolattartásra, azok felmérésére, a valós kapcsolati gráfok meghatározására, illetve az ütemezés hálózatba töltésére használatos. A következő alfejezetekben a szoftver fő funkcióit ismertetjük.

6.1. GraphGenerator fül

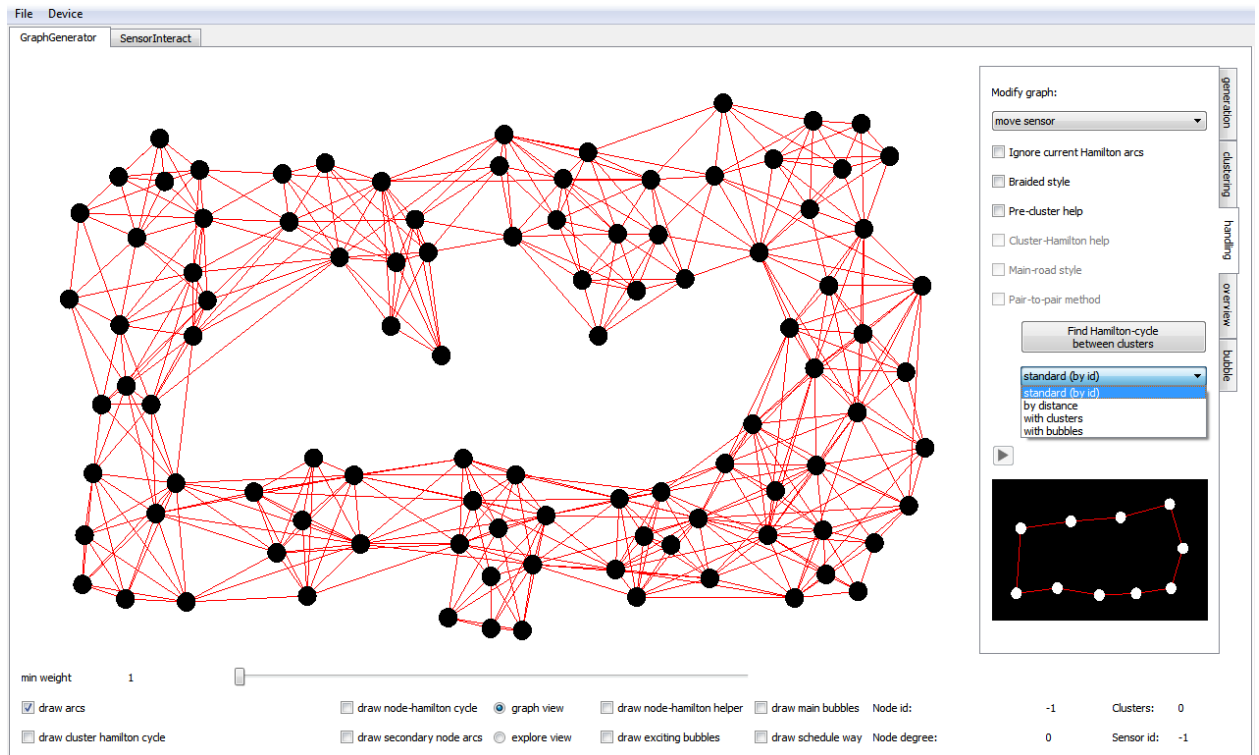
Ezen a fülön található meg minden gráffal kapcsolatos algoritmus, valamint segédfunkció.

- Gráfok generálása tesztek céljára. A foksám és az élek generálásának módja beállítható. Az élek vagy topológia alapon (távolság és véletlen faktorok segítségével) szimulálhatók, vagy a SensorInteract fülön megmért hálózat linkminőségi adataiból generálhatók. A linkminőségi adatokból a 6.1. ábra alsó részén látható csúszka segítségével választhatók ki az "elég jó" minőségű linkek.



6.1. ábra. GraphGenerator / clustering fül

- Klaszterezés. A klasztereket lehet automatikusan keresni, illetve ezeket a felhasználó módosíthatja, átrendezheti igény szerint. A 6.1. ábrán egy színekkel jelölt klaszterhalmaz látható.
- Keresések. A körkereső algoritmusok a "handling" alfülön találhatók. A 6.2. ábrán látható módon a felhasználó kiválasztja az alsó legördülő listából a keresési algoritmust, majd a "Find Hamilton-cycle between nodes" gombra kattintva elindul a keresés a kiválasztott keresési opciók figyelembe vételével. Alapvetően minden implementált algoritmus képes szimpla Hamilton-kör vagy b-Hamilton-kör keresésére. Az ábra jobb alsó sarkában a klaszterek által definiált segédgráf látható a boríték élekkel.

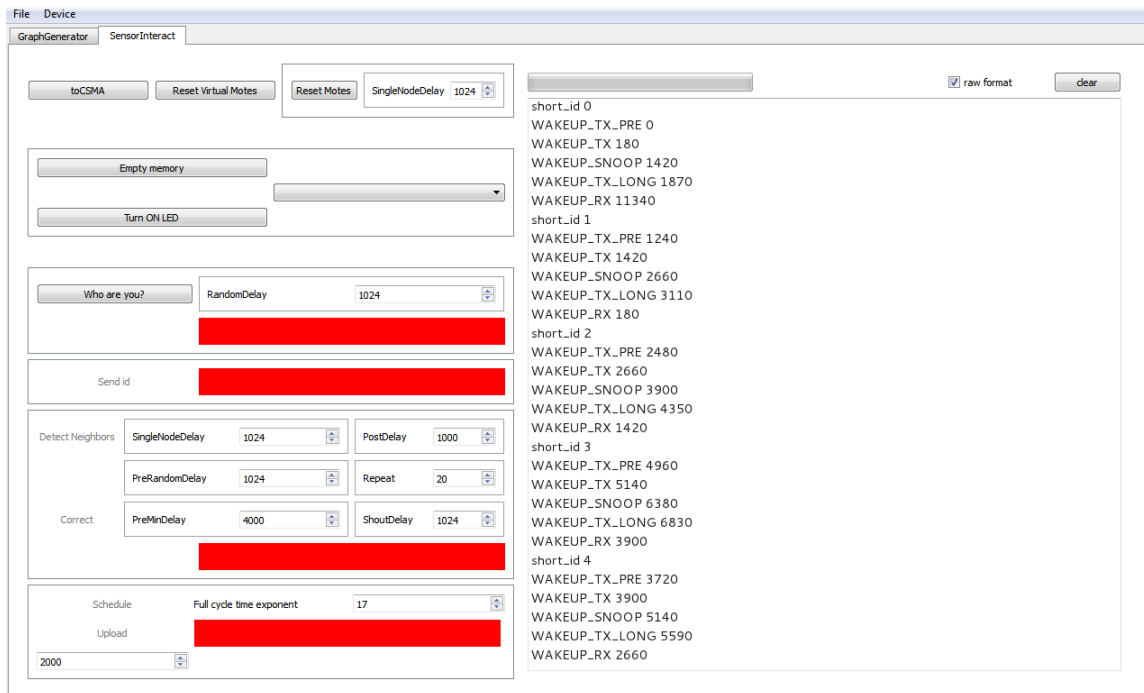


6.2. ábra. GraphGenerator / handling fül, körkeresési algoritmus kiválasztása

6.2. SensorInteract fül

A SensorInteract fül a szenzorhálózat kezelésére szolgál. A program soros vonalon egy gateway eszközhöz kapcsolódik, amely képes a szenzorhálózatba üzeneteket küldeni, illetve az onnan fogadott csomagokat programunk számára továbbítani. A SensorInteract fül fő funciója a hálózat felderítése, a kapcsolati gráf mérése (amelyen ezután a Graphgenerátor fülön segítségével a b-Hamilton kör megkereshető), majd a TDMA ütemezés előállítása és letöltése a hálózatba. A 6.3. ábra mutatja a SensorInteract felületet. Az ábra bal oldalán láthatók a kezelőszervek, a jobb oldalon pedig a küldött és kapott üzenetek listája található (debug céljára). A hálózat kezelése a bal oldali mezőben történik nyomógombok segítségével. Ezek a hálózat állapotának függvényében eltűnnek, illetve megjelennek. Az ábrán jelzett állapotban pl. a "Who are you?" nyomógomb aktív, míg pl. a "Send id" nyomógomb passzív.

A hálózat a felderítési szakaszban nem TDMA, hanem CSMA módban működik.



6.3. ábra. SensorInteract fül

Ekkor minden eszköz állandóan aktív és a hálózatban elárasztásos, illetve irányított elárasztásos üzenetszórési módokkal kommunikálnak [24].

A következőkben a főbb funkciókat foglaljuk össze.

- Általános hálózatkezelő és debug funkciók. A toCSMA gombbal a már esetleg TDMA módban működő hálózatot kényszeríthetjük arra, hogy visszalépjen CSMA módba és így a feltérképezés és újabb ütemezés elvégezhető. A “Reset Virtual Motes” gomb a PC oldali virtuális hálózatot indítja újra, míg a “Reset” gomb mind a virtuális hálózatot, mind a fizikai hálózatot reseteli. Az "Empty memory" és a "Turn ON LED" melletti legördülő listában jelenik meg az összes mote rövid címe. Kiválasztva egy címet, törölhetjük annak memóriáját vagy bekapcsolhatunk rajta egy ledet. Az utóbbi debug célból jelent meg a felületen. A “clear”-re kattintva az aktuális megjelenített kimenet (jobb oldal) tartalmát törölhetjük.
- Who are you? Erre a gombra kattintva elindul a hálózat felderítése. A gateway-en keresztül a hálózatot elárasztjuk a felderítés paranccsal, amire a hálózatban aktív

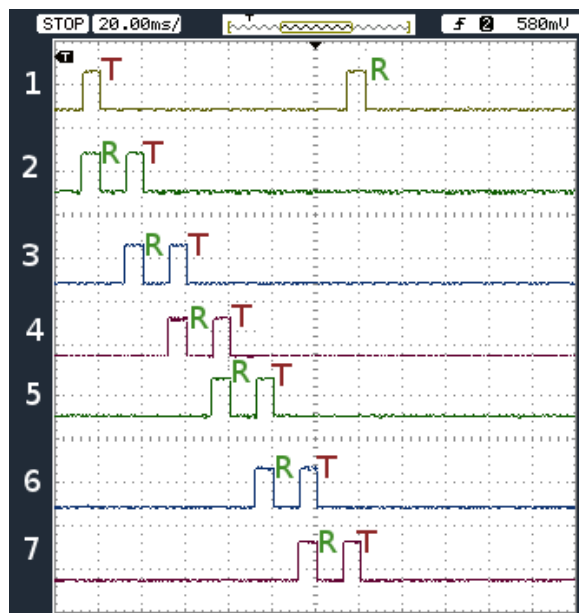
eszközök elküldik a 64 bites egyedi, ún. hosszú azonosítójukat. A SensorInteract ekkor a virtuális hálózatába új eszközöket vesz fel és mindegyiknek beállítja a hosszú és rövid azonosítóját.

- Send id. Minden hálózati csomópont a hosszú azonosítók helyett egy 16 bites rövid azonosítót kap. Az azonosítók küldése minden eszköznek nyugtázott, hibatűrő módon történik.
- Detect Neighbors. Ezen gomb indítja a hálózati kapcsolatok feltérképezését, a hálózat kapcsolati gráfjának mérését. A hálózat eszközei ezen parancs hatására - a program felületén beállított késleltetések figyelembe vételével - véletlenszerű időnként broadcast üzeneteket küldenek szomszédaiuknak, akik a kapott üzeneteket rögzítik. A hallott üzenetek statisztikáit az eszközök - a hatékonyság érdekében nem hibatűrő csatornán - visszaküldik. Megjegyzés: az üzenetek oly módon vannak jelölve, hogy az elveszett üzenetek a központban azonosíthatók.
- Correct. Az előbbi funkció végrehajtása során elveszett üzenetek pótlása, immár hibatűrő módon. A funkció a megfelelő mote-oknak neighbor request üzeneteket küld, míg a minden csomag meg nem érkezik.

Amennyiben a hálózat feltérképezése megtörtént, a b-Hamilton kör a GraphGenerator fülre kattintva megkereshető. A keresés végén a kör birtokában az ütemezés előállítható.

- Schedule. A b-Hamilton kör mentén a konkrét ütemezés megtörténik. A hálózat mérete a Full Cycle exponent paraméter segítségével állítható be.
- Upload. Az elkészült ütemezés a gateway-en keresztül feltöltésre kerül a hálózat minden csomópontjára. A feltöltés után a hálózat automatikusan átvált a megtervezett TDMA mód végrehajtására: az eszközök egymáshoz szinkronizálódnak és csak a beállított időpontokban ébrednek fel.

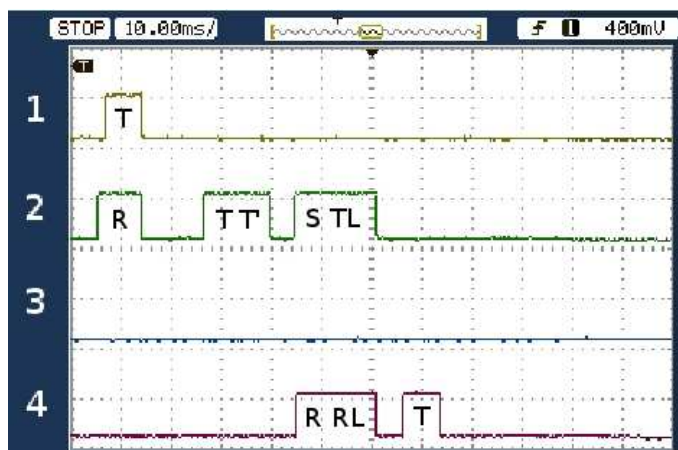
Egy teszhálózat TDMA módban való működését szemlélteti a 6.4. ábra. A mérés egy 7 csomópontot tartalmazó hálózatban készült, melyben az ütemezés szerint



6.4. ábra. Egy 7 szenorból álló hibamentes teszhálózat TDMA alapú működésének oszcilloszkópos felvétele

a szenzorok növekvő sorrendben egymás után adnak a következő sorszámú szenzornak, míg a 7. sorszámú szenzor az 1. számúnak ad a periódus végén. Az ábrán a rádiók ébrenléti ideje látható, adás (T) vagy vétel (R) idején. Az ábrát többcsatornás oszcilloszkóp segítségével készítettük, mellyel a rádió adóvevők ki és bekapcsolt állapotát mértük. Az ábrán jól láthatók a szinkron adás-, vétel ébredések (T-R), hasonlóan a 2.1. ábrán leírt (hibamentes) működéshez.

A 2. fejezetben javasolt, a 2.1. ábrán látható ütemezési módszer gyakorlati megvalósítását illusztrálja injektált hiba esetén a 6.5. ábra. Az ábrán négy szenzor aktivitása látható, melyek az 1, 2, 3, 4 sorrendben adnak egymás után. A teszhálózatban a 3. számú szenzort kikapcsoltuk. Jól látható, hogy az 1. szenzor adását a 2. sikeresen veszi. A 2. szenzor adására azonban a 3. szenzor nem küld nyugtát, így a 2. számú szenzor újra próbálkozik (T'). Mivel az ismételt adásra sem érkezik nyugta, a 2. szenzor ébren marad a 3. és 4. szenzor ütemezett kommunikációjának idejére, és hallgatja a 3. szenzor adását (S). Mivel a 2. szenzor nem hallja a 3. üzenetét, ezért a 2. szenzor az ismételt üzenetküldés időszelében egy



6.5. ábra. TDMA ütemezés működése a 3. számú szenzor kiesése esetén

áthidaló üzenetet küld a 4. szenzornak (TL). A 4. szenzor, mivel a 3. szenzortól nem kapott üzenetet (R), ébren marad és sikeresen veszi a 2. szenzor neki szóló üzenetét (RL), majd ő is továbbítja az üzenetét (T).

7. fejezet

Összefoglalás

A dolgozatban gyűrű topológiájú hálózatok hibatűrő ütemezésével foglalkoztunk. Ezen hálózati struktúrák egyes szenzorhálózati alkalmazások számára előnyös topológiájúak; ilyenek pl. az elosztott biztonsági rendszerek. Ezen hálózatokban potenciálisan minden eszköz érdeklődhet bármely más eszköztől származó információ iránt, így a riasztásokat, státuszadatokat és parancsokat minden eszközhöz el kell juttatni, ráadásul garantált időkereteken belül. Az egyes eszközök és kommunikációs linkek átmeneti vagy permanens meghibásodása elleni immunitás, hibatűrés ezen hálózatokban szintén alapvető követelmény.

A dolgozatban javasolt és vizsgált időosztás alapú (TDMA) ütemezés természetes megoldás a fenti követelmények esetén: a kommunikáció hatékony, nincs az ütközések elkerülése, javítása miatt felesleges energiafogyasztás, nagyon kis kitöltési tényező biztosítható, ráadásul az megtervezett ütemezés késleltetési ideje is ismert és garantált.

A dolgozatban közölt saját eredmények a következők:

- Hibatűrő ütemezési megoldást javasoltunk gyűrű topológiájú, egyirányú kommunikációt megvalósító szenzorhálózatokban. A javasolt megoldás egy Hamilton kör mentén szekvenciális TDMA alapú üzenettovábbítást valósít meg, amit a hibatűrés céljából kiegészítettünk egy áthidaló mechanizmussal, amelyhez Hamilton kör helyett annak hibatűrő variánsa (a b-Hamilton kör) használható. A javasolt megoldás energiahatékony, hiszen hibamentes esetben a

javító mechanizmus nem működik, így járulékos energiafogyasztás nem lép fel. Hiba esetén a javító mechanizmus segítségével a hálózat lokálisan detektálja a hibát és átugorja a hibás linket, illetve eszközt.

- A hibatűrő ütemezés megvalósítására definiáltuk a kapcsolati gráfon a b-Hamilton kör fogalmát.
- Klaszter-alapú heurisztikus kereső algoritmust javasoltunk b-Hamilton kör keresésére. Az algoritmus a hálózatot jól összefüggő részgráfokra (klaszterekre) bontja, majd először ezen klaszterek által definiált segédgráfon keres Hamilton kört, majd ezt a klasztereken belüli b-Hamilton utak keresésével a kapcsolati gráf b-Hamilton körévé egészíti ki.
- Félautomatikus, heurisztikus keresési módszert ("buborékos keresés") javasoltunk b-Hamilton kör keresésére, amely a vezeték nélküli hálózatok azon sajátosságát használja ki, hogy a kommunikációs linkek minősége és az eszközök távolsága között erős a korreláció. Ezen az alapon egy, a keresést lokálisan kimerítő részkeresésekre bontó heurisztikát terveztünk, amely a felhasználó által definiált régiókban irányított keresést hajt végre.
- A javasolt algoritmusokat széleskörű tesztelésnek vetettük alá, amelynek során egyszerűbb és bonyolultabb teszhálózatokon vizsgáltuk az algoritmusok hatékonyságát. A tesztek során saját algoritmusainkat összehasonlítottuk az irodalomból ismert - b-Hamilton kör keresésére kiegészített - Hamilton-kör kereső algoritmusok teljesítőképességével is.
- A javasolt algoritmusokat - sok más kiegészítő funkció kíséretében - egy egységes keretrendszerben megvalósítottuk. A rendszer a gráfkeresésen kívül alkalmas valós hálózatok feltérképezésére, a kapcsolati gráf mérésére, majd az ezen való b-Hamilton kör keresés után a képes a hálózatra a TDMA ütemezés elkészítésére és a hálózatra történő letöltésére.
- Algoritmusainkat valós hálózatok tervezésére alkalmazzuk. Bemutattuk az algoritmusaink által készített ütemezés felhasználásával működő hálózatok

ébrenlét-alvás mechanizmusát valós mérések segítségével. Ugyancsak bemutattuk a javasolt hibatűrő ütemezés működését valós körülmények között, mérések segítségével.

A dolgozatban közölt eredmények korántsem jelentik a kutatás végét, számos új és érdekes probléma vetődött fel a feladat megoldása közben:

- A gyűrű topológia kiegészíthető ennek hierarchikus változatával: a gyűrűn "fülek" foglalhatnak helyet, azokon újabb fülek, stb. Újabb feladat lehet ilyen topológiák meghatározása a kapcsolati gráfon, esetleg valamilyen előzetes felhasználói információ figyelembe vételével (ilyen lehet pl. a hálózatban alkalmazott szenzorok típusa). Ezen a topológián a hibatűrő ütemezés megvalósítása újabb feladat.
- Nagyobb kiterjedésű hálózatokban lehetséges több szenzor együttes adása oly módon, hogy azok nem zavarják egymás kommunikációját. Ilyen ütemezések készítéséhez az eddig használt kapcsolati gráfon kívül egy másik, ún. zavarási gráf felhasználásával lehetséges. Amennyiben több szenzor is adhat egyszerre, a hálózat sebessége és áteresztő képessége növelhető.
- Elképzelhető, hogy a hierarchikus felépítésű ("füles") hálózatban az egyes fülekben más és más sebességi követelmények vannak. Ezen többsebességű hálózatok ütemezése újabb érdekes problémákat vet fel.

Irodalomjegyzék

- [1] W. Dargie and C. Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley and Sons, 2010.
- [2] S. C. Ergen and P. Varaiya. TDMA scheduling algorithms for wireless sensor networks. *Wirel. Netw.*, 16:985–997, May 2010.
- [3] G. Vakulya and Gy. Simon. Design of a sensor network based security system. In *Proc. of IEEE International Symposium on Intelligent Signal Processing*, Floriana, Malta, September 19-21 2011.
- [4] Lei Guo, Xingwei Wang, Juan Du, and Tengfei Wu. A new heuristic routing algorithm with hamiltonian cycle protection in survivable networks. *Comput. Commun.*, 31(9):1672–1678, June 2008.
- [5] Á. Orosz, G. Róth, and Simon G. TDMA scheduling in fault tolerant wireless sensor networks. In *Proc. 2012 IEEE International Instrumentation and Measurement Technology Conference, Graz, Austria*, pages 1–5, 2012.
- [6] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. New York: Plenum Press, 1972.
- [7] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc. Vol.*, s3-2:69–81, 1952.
- [8] Oystein Ore. Arc coverings of graphs. *Ann. Mat. Pura Appl.*, 55:315–321, 1961.

- [9] K. Thulasiraman and M. N. S. Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, 1992.
- [10] L. Pósa. A theorem concerning hamilton lines. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 7:225–226, 1962.
- [11] L. Pósa. Hamiltonian circuits in random graphs. *Discrete Math.*, 14:359–364, 1976.
- [12] S. B. Karmakar. Detection of hamiltonian circuits in a directed graph. *Journal of the Australian Mathematical Society, Series B.*, 24:234–242, 1982.
- [13] M. R. Gary, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is np complete. In *SIAMJ. Comput.* 5, pages 704–714, 1976.
- [14] D. Kroft. All paths through a maze. In *Proc. IEEE*, volume 55, pages 88–90, 1967.
- [15] S. Hakimi. Recent progress and new problems in applied graph theory. In *IEEE Region Six Conference Record*, pages 635–643, 1966.
- [16] F. Rubin. A search procedure for hamilton paths and circuits. *Journal of the ACM (JACM)*, 21, no 4:576–580, 1974.
- [17] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 30–41, 1977.
- [18] I. Shields. *Hamilton Cycle Heuristics in Hard Graphs*. PhD thesis, North Carolina State University, 2004.
- [19] B. Bollabas, A. M. Frieze, and T. I. Fenner. An algorithm for finding hamilton paths and cycles in random graphs. *Combinatorica.*, 7:327–341, 1987.
- [20] F. A. Brunacci. DB2 and DB2A - 2 useful tools for constructing hamiltonian circuits. *European Journal of Operational Research*, 34 (2):231–236, 1988.
- [21] Eric Lewin Altschuler, Martin Lades, and Richard Stong. Finding hamiltonian cycles. *Science*, 273:413, 1996.

- [22] Ashay Dharwadker. A new algorithm for finding hamiltonian circuits. In *Proceedings of the Institute of Mathematics*, 2004.
- [23] Sudhangshu B. Karmakar. A heuristic method for the determination of a Hamiltonian circuit in a graph. *Journal of the Australian Mathematical Society, Series B.*, 28:328–339, 1987.
- [24] M. Maróti. Directed flood-routing framework for wireless sensor networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 99–114, 2004.