Hibatűrő TDMA ütemezés tervezése ciklikus vezeték nélküli hálózatokban

Orosz Ákos, Róth Gergő, Simon Gyula

Pannon Egyetem

Rendszer- és Számítástudományi Tanszék

Email: {orosz, roth, simon}@dcs.uni-pannon.hu

Kivonat

Az időosztásos (TDMA-alapú) hálózatok előnye a verseny-alapú (pl. CSMA) közeghozzáférési technológiákkal szemben, hogy egyrészt képesek garanciákat nyújtani az üzenetek kézbesítési idejére, másrészt pedig az időosztásos rendszerek rendkívül energiatakarékos működést tudnak biztosítani azáltal, hogy az idő jelentős részében a hálózati elemek alhatnak. Az alkalmazások logikája természetszerűen megjelenhet a hálózati topológiájában, így pl. egy adatgyűjtő alkalmazás gyakran fa-topológiájú, míg más alkalmazások (pl. riasztó rendszerek) kör topológiájúak lehetnek. Ez a cikk hibatűrő ciklikus hálózatok tervezési kérdéseivel foglalkozik: először definiálunk egy hibatűrő ciklikus architektúrát, majd hatékony kereső módszereket adunk ilyen architektúrák keresésének (NP-teljes komplexitású) problémájára.

I. Bevezetés

Vezetéknélküli szenzorhálózatok esetében a TDMA (Time Division Multiple Access) alapú ütemezést használva valósidejű működés és energiamegtakarítás érhető el. Ennek oka, hogy az ütemezéssel elkerülhetőek a csomagok közötti ütközések, illetve az egyes szenzorok sem figyelnek üres járatban, helyette inkább energiatakarékos üzemmódra váltanak (alszanak) [1]. Csak akkor ébrednek fel kommunikáció céljából, amikor üzenetet adnak, vagy fogadnak.

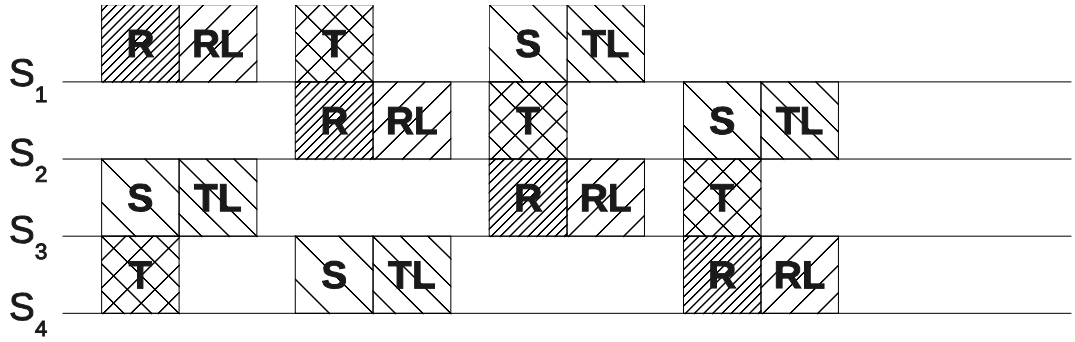
A hálózat felépítése az alkalmazástól függ. Egy adatgyűjtő hálózatot általában fa-felépítésűre terveznek. Azonban a legtöbb alkalmazás nem csak gyűjti az adatot, hanem folyamatosan áramoltatja is azzal a céllal, hogy bármelyik szenzor küldhessen üzenetet bármelyik másiknak (pl. riasztó rendszerek) [2]. Ezen hálózatokban sokkal inkább a gyűrű topológiát szokták alkalmazni. A mi kutatásunk célja gyűrű topológián hibatűrő hálózatot tervezni, ütemezni.

II. Hibatűrő TDMA ütemezés

Gyűrű topológiákban a szenzorok körkörösen 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 [3], de alkalmazhatók heurisztikák, melyek csökkenteni tudják a keresésre szánt időt.

A hálózatnak továbbá egy-hiba-tűrőnek is kell lennie, ami azt jelenti, hogy egy szenzor kiesése (meghibásodás, vagy lemerült elem miatt) esetén is tovább kell működnie. Ennek megoldására vezettük be a későbbi 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, azt a hálózat automatikusan át tudja ugrani.

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), 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 elmaradt, tehát a következő szenzor nem üzemel), akkor elküldi az üzenetet a következő utáni szenzornak (TL - Transmit Long). Minden üzenetet nyugtáz a hálózat, amennyiben az megérkezik. Az 1. ábrán látható egy példa ütemezés egy 4 szenzort tartalmazó hálózatban.



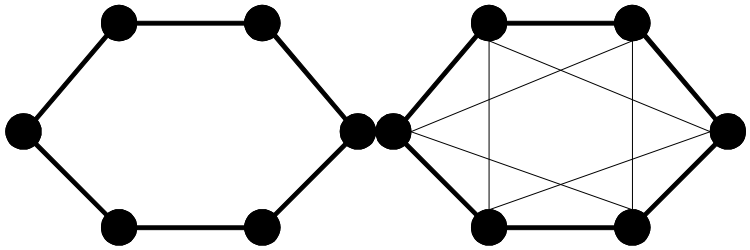
1. ábra. Ütemezési műveletek (R, RL, T, S, TL) egy 4 méretű hálózatban

Ideális esetben, ha nem vesznek el üzenetek, az RL és TL funkciókra nincsen szükség: ha egy T üzenetre nyugta érkezik, a küldő szenzor azonnal alvó állapotba vált. Ha egy nyugta elveszik (pl. S_2 nem kap nyugtát S_3-tól), a küldő belehallgat a következő kommunikációba (S_2 meghallgatja, hogy S_3 milyen üzenetet küld, ha küld). Ha a hallgatózás sikeres (S_2 hallja S_3-at), akkor a küldő befejezte 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.

III. Hibatűrő Hamilton-körök keresése

Azért, hogy az előző fejezetben leírt ütemezés működni tudjon, a hálózat kapcsolati gráfjának tartalmazni kell legalább egy 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.

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



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

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

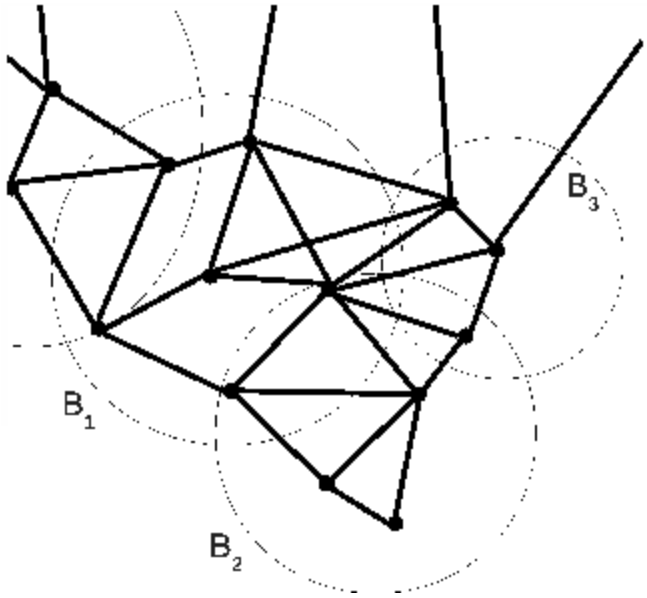
III-A. Brute-force algoritmus

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 csúcstól 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 szétválasztás, és onnan keres tovább.

III-B. Buborékos keresés

A buborékos keresés egy speciális klaszter alapú keresés, ahol a felhasználó definiál klasztereket (részgráfokat). Mivel a legegyszerűbb alakzat a kijelölésre a kör, ezért az implementáció köröket használ, innét a buborék megnevezés. A 3. ábra a koncepció alapötletét mutatja be. A buborékok (nem feltétlenül diszjunkt) sorozatát a következőképpen jelöljük: B_1, B_2, B_3, ... B_k.



3. ábra. Csúcsok buborékokban

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.

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

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

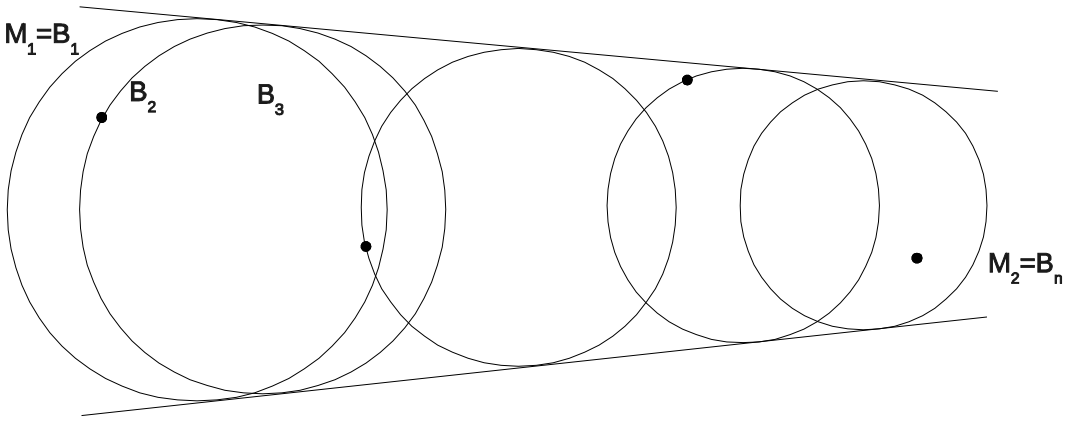
A b-Hamilton kör kereső algoritmus egymás után megpróbálja lefedni az összes buborékot a következő egyszerű szabályt betartva: egy buborékba (B_i) nem lép be addig, míg az előtte levő buborék (B_{i-1}) nem fedett.

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 lényeges az algoritmus futási szempontjából.

Ahhoz, hogy a felhasználó el tudja indítani az algoritmust, ki kell választania a fő buborékokat és kézzel meg kell határoznia ezek sorrendjét (M_1, M_2, M_3, ... 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 buborékokat is.

Ha a felhasználó az utóbbit választja, akkor az algoritmus automatikusan létrehozza azokat. Az interpolált buborékok középpontja a két fő buborék középpontját összekötő szakaszon lesz. Sugaruk pedig akkora, hogy bármely kiválasztott interpolált buborékhoz és a két fő buborékhoz lehet

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. ábra mutatja.



4. ábra. Interpolált buborékok (B_1, B_2, .. B_n) az M_1 és M_2 fő buborékok közt: M_1 és M_2

Miután a fő (és opcionálisan az interpolált) buborékok már adottak a gráfban, a keresési algoritmus elkezdődhet a nyílt csúcsok halmazán. Kezdetben a nyílt csúcsok halmaza a kezdő buborékban található csúcsok. 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. Az algoritmus pszeudo kódja 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

    $found = branch(G, S)$

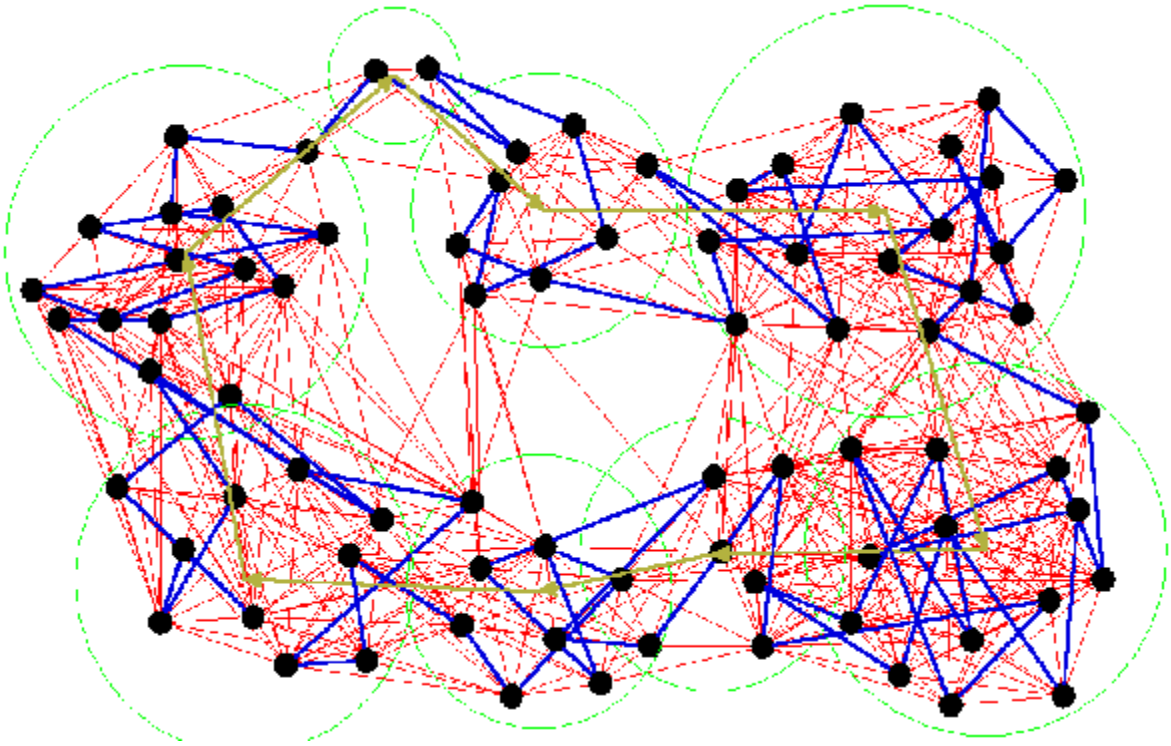
$done$

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 ∀ i ∈ open_vertices
                if((C[end − 1], i) ∈ E ∧ (C[end], i) ∈ E)
                        if(count = n − 1)
                                if((i, C[1]) ∈ E ∧ (i, C[2]) ∈ E ∧ (C[end], C[1]) ∈ E)
                                        solution = true
                                        return
                                fi
                        else
                                SET = SET ∪ {State([C|i], count + 1)}
                        fi
                fi
        done
end
```



5. ábra. Teszthálózat 80 csúccsal (fekete pöttyök). Piros élek: kommunikációs linkek, zöld körök: fő buborékok, barna nyílak: fő buborékok sorrendje, kék élek: b-Hamilton kör

IV. LÉTEZŐ HEURISZTIKÁK

Az irodalomban léteznek Hamilton-kör keresésére vonatkozó heurisztikák. Ezen heurisztikák alkalmazhatóságát vizsgáltuk b-Hamilton-kör keresésére.

Egy ilyen heurisztikát publikált Ashay Dharwadker 2004-ben [4]. 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 próbálkozásos alapon keres, mint a korábbi heurisztikák, amiknek a futási ideje ezáltal exponenciálisan nő a csúcsok számának növelésével, hanem heurisztikus módon elkezdi felépíteni a kört, és soha nem vált irányt, korábbi döntéseihez ragaszkodik. Az algoritmus részletes leírására itt nem térünk ki, csak lényegében ismertetjük. Első lépésként utat keres a gráfban, ezt addig csinálja, amíg talál újonnan bevehető csúcsot. Ha az eddig felépített utat nem tudja folytatni, akkor nem kezd el keresni másik irányba, hanem elővesz egy másik heurisztikát. 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 váltogatva polinom időben befejezi a futását. Ezek végén nagy valószínűséggel talál Hamilton-kört a gráfban, sőt legtöbb esetben többet is.

Ezt az algoritmust módosítottuk úgy, hogy képes legyen b-Hamilton-kört keresni. Ezt úgy valósítottuk meg, hogy az algoritmus lépésein és a heurisztikán nem változtattunk, pusztán az új csúcsok hozzáadásának, illetve a részutak megfordításának feltételét egészítettük ki úgy, hogy a műveletek végrehajtásához a másodlagos éleknek is létezniük kell a kacsolati mátrixban. Ezekkel a kiegészítésekkel alkalmassá tettük az algoritmust, hogy b-Hamilton-kört keressen a gráfban. Következő lépésként futtattuk olyan gráfokra, melyekben létezik b-Hamilton-kör, és vizsgáltuk, hogy megtalálja-e a kört. A tapasztalatok azt mutatták, hogy az egyébként Hamilton-kör keresésére kiválóan működő algoritmus b-Hamilton-kört sokkal kisebb valószínűséggel talál a gráfban még akkor is, ha van benne. Ennek oka az, hogy a Hamilton-kör keresésére kifejlesztett heurisztikák a sokkal több kényszert tartalmazó b-Hamilton-kör keresésére nem alkalmasak. Emiatt az algoritmus ismeretében könnyen lehet olyen gráfot szerkeszteni, melyre a kiegészített algoritmus zsákutcába kerül akár már a keresés elején.

Egy másik heurisztikát publikált 1985-ben Sudhangsu B. Karmakar [5]. Ez az algoritmus egy gráf adjacencia mátrixának a tulajdonságait használja ki a kör kereséséhez. A [4] heurisztikájával ellentétben ez az eljárás nem szűkíti le a keresési teret, így mindenképpen megtalálja a kört, amennyiben van a gráfban. Tehát a futási ideje sem mérhető polinomiális időben. Tapasztalataink azt mutatták, hogy ezzel az algoritmussal a keresés jóval lassabban működik a [4]-ben bemutatottnál. Egy jól felépített buborékos kereséssel össze sem lehet mérni a futási idejét Hamilton-kör keresés esetében.

Az algoritmus néhány módosítással átírható b-Hamilton-kör keresésére, de ezzel elveszíti a heurisztikájának alapját. Ezzel is végeztünk összehasonlításokat, de ez az automata algoritmus se vált be. Nem működik jobban az algoritmus, mint az egyszerű brute-force, amit kiegészítettünk b-Hamilton-kör keresésére.

Összegzésként, léteznek jó heurisztikák szimpla Hamilton-kör keresésére, de ezen heurisztikák b-Hamilton-kör keresésére a tesztjeink szerint nem hatékonyak. Mivel ezzel a problémával még nem foglalkoztak, így közvetlenül b-Hamilton-keresésére készített heurisztikák nem léteznek.

HIVATKOZÁSOK

[1] S. C. Ergen and P. Varaiya. TDMA scheduling algorithms for wireless sensornetworks. *Wirel. Netw.*, 16:985–997, May 2010.

[2] G. Vakulya and Gy. Simon. Design of a sensornetwork based security system. In *Proc. of IEEE International Symposium on Intelligent Signap Processing*, Floriana,Malta,September19-212011.

[3] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. NewYork: Plenum Press, 1972.

[4] Ashay Dharwadker. A new algorithm for finding hamiltonian circuits. In *Proceedings of the Institute of Mathematics*, 2004.

[5] 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.