Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

A dekompozíció és az ütemezés iteratív összehangolása a rendszerszintű szintézisben

Szerző:
Markovits Tibor Gergely

Konzulensek:
Dr. Arató Péter
Rácz György

Kulcsszavak: dekompozíció, rendszerszintű szintézis, ütemezés, intelligens rendszerek, magasszintű szintézis, Súlyozott Normalizált Vágás

TDK 2016

Tartalomjegyzék

Köszönetnyilvánítás

Hálás köszönettel tartozom mentoromnak és konzulensemnek, Dr. Arató Péter professzor úrnak a témába való bevezetésért és a sok segítségért, amit a munkám során nyújtott.

Köszönet illeti Rácz György egyetemi tanársegédet, aki folyamatosan rendelkezésemre állt, javaslatai és észrevételei nagyban hozzájárultak kutatás sikeréhez.

Köszönöm dr. Laczkó Krisztina egyetemi docens segítségét, aki nyelvtani kérdéseimre adott válaszaival megkönnyítette dolgozatom megírását.

Abstract

The system level synthesis of heterogeneous multiprocessing architectures set some new requirements in the design flow in order to approach optimal embedding DSPs, FPGAs, and GPUs and other component processors. In the last decade, numerous system level synthesis tools have been presented.

In this paper, a novel approach is presented for designing heterogeneous multiprocessing systems. This is implemented by two, self-developed algorithms connected tightly with each other. The Weighted Normalized Cut (WNCut) replaces the decomposition and it has many advantages. Most of the design tools use iterations in searching the best solution for their partitions. Such intuitive methods slow down the design flow that is disadvantageous. The algorithm WNCut – implemented in closed form – reduces the number of iterations significantly. It delivers a non-discrete solution to validate the predetermined priorities during the scheduling.

The result of the WNCut is a data set that can be used as input for the scheduling and allocating phases. This is the aim of an another algorithm, called Dynamic Scheduler (DS). By replacing the decomposition and applying the WNCut algorithm, DS can be used directly for splitting the process into two or more parts, by considering the earliest start time of each single task. Thus, WNCut performs not only a simple decomposition, but it also analyses the whole task. Thereafter, DS performs the scheduling and the partitioning without needing to recalculate the whole decomposition. This is because the DS uses the already available data set for searching a better partition with respect to the scheduling.

This new method may be applied effectively in cloud application, where the design steps of task partitioning and scheduling must be repeated many times, because system parameters change during the execution time of the application.

The method presented in this paper may be applied as a basis of a competitive system level synthesis tool.

Kivonat

A heterogén multiprocesszoros rendszerek megjelenésével egy teljesen új út nyílt meg a rendszerszintű szintézis terén. A DSP-k, FPGA-k és GPU-k lényegesen megváltoztatták a tervezés menetét, valamint azt a fajta a gondolatmenetet, ahogyan az egyes feladatokat implementáljuk. Az elmúlt évtizedben számos rendszerszintű szintézer eszköz jelent már meg.

Ebben a dolgozatban egy új megközelítést szeretnék bemutatni az ilyen rendszerek tervezési folyamatára. Ezt két, általam kifejlesztett, egymáshoz kapcsolódó algoritmus valósítja meg. Az első algoritmus, a Súlyozott Normalizált Vágás (Weighted Normalized Cut, WNCut) a dekompozíciót helyettesíti, ennek lényeges előnyei bizonyíthatók. A legtöbb tervezőeszköz iterációs módszereket használ, hogy a tervező számára a legjobb megoldást keresse. Ez az intuitív módszer azonban többnyire lelassítja a tervezés folyamatát, és ez rendkívül hátrányos. A WNCut algoritmusa zárt formulákat használ; ezeknek matematikai megoldásai az iterációk számát jelentősen csökkentik, valamint nemdiszkrét megoldást szolgáltatnak annak érdekében, hogy az előre meghatározott prioritások később érvényre juthassanak az ütemezés során.

A WNCut eredményeként egy olyan adathalmazt kapunk, amely az allokáció és az ütemezés bemeneti paramétereként fog szolgálni. Ehhez fejlesztettem ki a másik új algoritmust: a Dinamikus Ütemezőt (Dynamic Scheduler). A dekompozíció helyettesítésével és a WNCut alkalmazásával elérhetjük, hogy közvetlenül a Dinamikus Ütemező algoritmusát használjuk a feladatok felosztására, majd az erőforrásokhoz rendelésére, figyelembe véve a feladatok legkorábbi indulási idejét. Ezáltal az első lépésben a WNCut analizálja a folyamatot, a második lépésben a Dinamikus Ütemező pedig elvégzi a feladat ütemezését és felosztását anélkül, hogy esetlegesen újra kellene futtatni egy dekompozíciós algoritmust, hiszen a Dinamikus Ütemező egy jobb vágás kiválasztása érdekében a már meglévő adathalmazt használja fel.

Az új módszer nagy hatékonysággal használható fel például felhőalapú alkalmazásokban, ahol a rendszernek működés közben akár többször is, minél rövidebb idő alatt el kell tudnia végezni a tervezési lépéseket a változó rendszerparaméterek miatt.

A dolgozatban bemutatott módszer – véleményem szerint – egy versenyképes rendszerszintű szintézist végző eszköz alapja lehet.

Bevezetés

A multiprocesszoros rendszerek az elmúlt másfél évtizedben egyre fontosabb szerepet kaptak a digitális célrendszerek területén. A bonyolultabb műveletek nagyobb számítási kapacitást igényelnek, az egyes feldolgozóegységek sebessége azonban már nem növelhető tovább olyan mértékben, hogy ki tudja szolgálni a rajtuk futó feladatokat.

A megoldást az erőforrások többszörözése jelenti, ez azonban számos problémát vet fel egy rendszer megtervezése és üzemeltetése során. A komplexebb rendszerek esetében a megvalósítás nem feltétlen egyértelmű, hiszen a különböző feladatokat nemcsak a megfelelő erőforrásokhoz kell rendelni, hanem úgy kell elosztani őket, hogy azok egymás között képesek legyenek hatékonyan kommunikálni. Ez rendkívül fontos, hiszen egy feladat kimenetén megjelenő adatok egy új feladat bementi paramétereit képezhetik. Ezért elengedhetetlen a folyamatok helyes felosztása és a megfelelő sorrendbe helyezése. Legtöbbször azonban nemcsak a feldolgozandó folyamat határozza meg a tervezés menetét, hanem egyes, a tervező által *előre meghatározott prioritások és követelmények* (Predetermined Priorities and Requirements, a továbbiakban *PdPR*). Követelményként beszélhetünk például az anyagi költségekről, a sebességről, pipeline működésről, párhuzamosíthatóságról stb. A követelmények egymástól is függnek, hiszen például, ha az anyagi keretek lehetővé teszik, rendszerünket megvalósíthatjuk egy vagy több nagyteljesítményű processzorral, tekintet nélkül arra, hogy hogyan használjuk ki, illetve mekkora adatcsomagokat továbbítanak egymás között, és milyen módon.

A fent leírt feladatokhoz célszerű volt létrehozni egy olyan komplex tervezési metódust, amelynek segítségével megvalósíthatjuk a tervező által meghatározott PdPR-eket kielégítő digitális célrendszert. A feldolgozóegységek többszörözéséből adódó előnyök maradéktalan kihasználásához magas absztrakciós szintről, ún. viselkedési specifikációból kell indítani a tervezési folyamatot, amelyet *rendszerszintű szintézisnek* (System Level Synthesis, továbbiakban *SLS*) nevezünk. A viselkedési szintről kiinduló szisztematikus tervezési módszerek az ún. *magas szintű szintézis* (High Level Synthesis, továbbiakban *HLS*) algoritmusainak kiterjesztése révén alakultak ki. [22,27,34] Az SLS során arra törekszünk, hogy különböző HLS-algoritmusok együttes használatával olyan tervezési módszereket alakítsunk ki, amelyekkel a lehető legjobban kihasználhatjuk a rendelkezésünkre álló erőforrásokat. Ezáltal hatásfokuk kimagasló lehet, figyelembe véve a PdPR-t.

A kiterjesztett HLS-algoritmusok egy-egy részfeladatát oldják meg az SLS-nek, és az esetek nagy részében olyan intuitív próbálgatásokra alapoznak, amelyeket iterációs eljárásokkal valósítanak meg. Ennek következtében a legtöbb algoritmus által használt matematikai eljárás nem oldható meg polinom időben, nem vezet kétszer ugyanahhoz az eredményhez, és nem biztosítják mindig a legjobb megoldást az előírt követelményeknek megfelelően. A feladat feldolgozására nem létezik determinisztikus algoritmus, mivel ez

NP-teljes probléma. Ezt az elmúlt évek során többen is bizonyították, de ebben az értekezésben Papadimitrou bizonyítására támaszkodunk, amely a dekompozíciós eljárás NP-teljes voltát igazolja. A későbbiekben azonban kiderül, hogy létezik olyan elfogadható közelítő megoldás, amely zárt alakot használ, és mellőzi az iterációs eljárást, ezáltal a tervezési feladat polinom időben megoldhatóvá válik.

A dolgozatban tárgyalt HLS-algoritmusok kifejlesztése során a legfontosabb szempont egy olyan eljárás kialakítása volt, amely a következő fejezetekben tárgyalt dekompozíció és ütemezés kapcsolatát rugalmasan képes kezelni. A későbbiekben látni fogjuk, hogy ezzel a megközelítéssel a particionálás szerepét átveszi az ütemező, ezáltal a dekompozícióként ismert tervezési lépés nem ad egyértelmű választ arra vonatkozóan, hogy az egyes műveletek vagy feladatok mely processzorokhoz fognak tartozni.

A következő fejezetben megismerhetjük a heterogén multiprocesszoros rendszereket és ezek előnyeit. A későbbi fejezetekben pedig áttekintjük a rendszerszintű szintézis általános lépéseit, valamint az új algoritmusok matematikai leírását és bizonyítását, a dolgozat végén pedig példákat láthatunk a WNCut és a Dinamikus Ütemező által feldolgozott folyamatgráfokra.

1 Multiprocesszoros rendszerek feldolgozó egységei

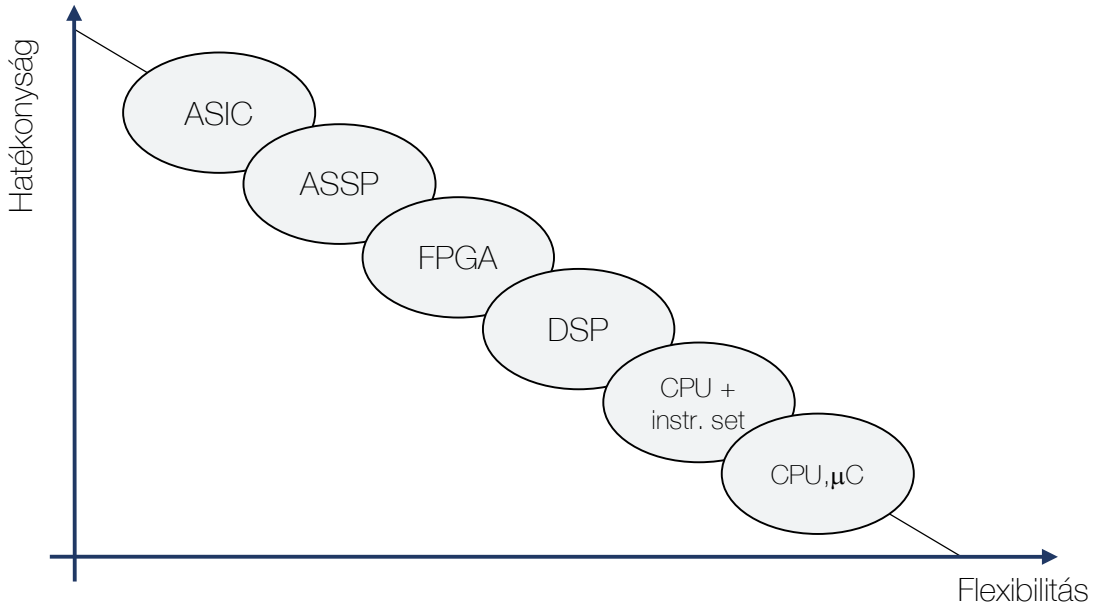
Az elmúlt 20 évben a processzorgyártó cégek nagy energiát fordítottak a műveletvégző egységek számítási kapacitásának megsokszorozására. A legkézenfekvőbb megoldás a processzor órajelének növelése, azonban ez erősen függ a technológiától, amely gátat szab a frekvencia növelésének. Számba vehetjük a belső részegységek (pl. magok), vagy magának az egész processzor többszörözésének lehetőségét. Ez a megoldás ugyan költséges, azonban segíthet a fent leírt követelmények betartásában, abban az esetben, ha megfelelő SLS-algoritmussal osztjuk szét és ütemezzük a feladatokat. Létrehozhatunk vagy többszörözhetünk speciális, belső műveletvégző egységeket, melyek ugyan egyetlen feladatot képesek elvégezni, de azt nagy hatékonysággal. Ilyen művelet lehet például az FFT (Fast Fourier Transform) vagy a konvolúció.

A fentiek alapján kijelenthetjük, hogy a feladatspecifikus rendszerek esetében figyelembe kell venni az előre meghatározott, időőigényes feladatokat. Az ilyen típusú számításokat egy általános célú CPU-nál sokkal hatékonyabban tudják elvégezni a digitális jelfeldolgozó egységek (DSP-k), a GPU-k és az FPGA-k. Ezeket a komponens processzorokat feladatspecifikus multiprocesszoros rendszerek esetén érdemes a hagyományos, általános célú CPU-kkal vagy mikrokontrollerekkel együtt használni. Az ilyen heterogén architektúrákat hívjuk heterogén multiprocesszoros rendszereknek (Heterogen Multiprocessing Systems-nek, továbbiakban HMS-nek).

1.1 Heterogén multiprocesszoros rendszerek

A rendszertervezés során tehát figyelembe kell vennünk a feladatok típusait. Fontos ügyelni arra is, hogy ezeket a feladatokat milyen komponens processzorokkal valósítjuk meg. Szignifikáns jellemzőjük a relatív hatékonyságuk (pl. számítási kapacitás, fogyasztás), illetve flexibilitásuk. A flexibilitás jellemzően a tárolt programú gépek tulajdonsága, amely lehetővé teszi könnyen változtatható programok futtatását. Beláthatjuk azonban, hogy a feldolgozó egységek flexibilitása a hatékonyság rovására megy. Minél flexibilisebb egy processzor, annál kevésbé lesz hatékony a feladatspecifikus rendszerek esetében. (1. ábra)

Az ASIC-eket, (Application-Specific Integrated Circuit), mint nevéből is kiderül, alkalmazásorinetált, egyedi feladatokra fejlesztik ki. Hatékonyságuk kimagasló, rendszertervezés során azonban ezeket az építőelemeket gyártási költségeik miatt nagyon ritkán használják fel. A következő feldolgozóegység-típus az ASSP (Application-Specific Standard Product), amely már nem egy egyedi, hanem széles körben elterjedt feladatot valósít meg (pl. videókonvertálás), ezért előállításuk jóval olcsóbb, így megtalálhatók a piacon. Nagyobb rendszerek esetén kifejezetten érdemes ezeket használni, hiszen egy ASSP komoly tehermentesítést jelenthet a CPU-k, a DSP-k vagy a GPU-k számára. Az FPGA-k (Field Programmable Gate Array) jóval nagyobb mozgási teret adnak a



1. ábra: Feldolgozóegységek hatékonysága a flexibilitás függvényében

tervezőnek, mint a korábban említett ASIC és ASSP feldolgozóegységek. Változatos méretük viszonylagosan olcsó megvalósítást tesz lehetővé. Az FPGA-k ma már lehetőséget adnak arra, hogy a huzalozott logika mellett létrehozzunk chipen belül lágyprocesszorokat, ún. Soft Core-okat. Ezáltal nemcsak a CPU-t tehermentesítheti, hanem át is vállalhat bizonyos feladatokat annak érdekében, hogy a processzorok közötti adatkommunikáció minimálisra csökkenjen. Az előbb megismertekhez nagyon hasonlóak a DSP-k, azzal a különbséggel, hogy ezeknek előre meghatározott belső műveletvégző egységeik és Harvard architektúrájuk van. HMS esetében manapság egyre többet használják ezeket, mivel rendkívül hatékonyak, ugyanakkor az áruk évről évre csökken. Ezt követik azok az általános célú CPU-k, amelyek kibővített utasításkészlettel rendelkeznek, tipikusan ilyenek az asztali PC-k processzorai. Ez a típusú processzor a legkevésbé alkalmazásspecifikus, nem tartalmaz periféria-áramköröket, és nehéz beleintegrálni őket a rendszerbe, ezért használatuk egyáltalán nem jellemző a HMS rendszerek esetében. A leginkább flexibilis feldolgozóegységek maradtak a végére: ezek az általános célú CPU-k és a mikrokontrollerek. Architektúrájuk lehetővé teszi különféle programok és bonyolultabb számítási műveletek végrehajtását is, ugyan jóval kisebb hatékonysággal, mint a korábban megismert komponensprocesszorok.

1.2 Rendszerszintű tervezés heterogén multiprocesszoros rendszerek esetén

A rendszerszintű tervezés akkor eredményes, ha az előbb felsorolt eszközök közül ki tudjuk választani azokat, amik az elvégzendő feladatok számára a leghatékonyabbak, és

ezeket egyetlen rendszerbe bele tudjuk helyezni, effektív kommunikációt kialakítva egymást között. [11]

Szemléltetve a fenti kijelentést, vegyünk példának egy bonyolultabb matematikai műveletet, a konvolúciót. A konvolúció megvalósítása lehetséges általános célú CPU segítségével is, de igazán hatékonyak akkor vagyunk, ha ezt DSP-vel oldjuk meg, ami akár több, mint egy nagyságrenddel gyorsabb műveletvégzést biztosít, processzortípustól függően. Ezen túl kihasználhatjuk a DSP magját arra, hogy átvállaljon bizonyos számítási műveleteket a hozzá kapcsolódó és az adatokat szolgáltató általános célú CPU-tól, aminek segítségével a processzorok közötti kommunikáció csökkenhet.

A dekompozíciós eljárás során érdemes úgy felosztani a feladatokat, hogy az azonos típusú műveletvégző eszközökkel elvégzendő számítások egy szegmensbe kerüljenek. Ezáltal csökkentjük a különböző felesleges komponens processzorok darabszámát, és csak azokat a műveleteket végeztetjük el velük, amik feltétlen szükségesek. Így spórolhatunk az anyagi költségeken, mivel általánosságban elmondható, hogy a legolcsóbb erőforrások az általános célú CPU-k és mikrokontrollerek, ezért a rendszertervezést mindig ezekkel a processzorokkal célszerű kezdeni [33].

Az előbb bemutatott problémák vitathatatlanul meghatározzák a HMS rendszerünk viselkedését, a hatékony működés azonban nemcsak az egyes tervező eszközöktől függ, hanem azok összehangoltságától is. Előfordulhat például, hogy egy jól sikerült dekompozícióból kiindulva az ütemező nem képes olyan hatásfokkal ütemezni, mintha egy kevésbé jó dekompozícióból indult volna ki, ezáltal a rendszerszintű szintézis kevésbé jó eredményt tud szolgáltatni a tervező számára. A már korábban is többször említett lépések (pl. dekompozíció és ütemezés) HLS-algoritmusai önmagukban működőképes eszközök lehetnek, és elfogadható megoldást adhatnak a HMS rendszerek egyes tervezési lépéseiben, ám még hatékonyabbak, ha képesek bővebb adatot szolgáltatni egymásnak a diszkrét eredményeken kívül[28].

Ehhez a következőkben a rendszerszintű tervezés egyes lépéseit fogjuk megvizsgálni, valamint azt, hogy az azok közötti kapcsolatokon milyen módszerekkel lehet javítani.

2 A rendszerszintű tervezés lépései HMS esetében

Először célszerű megvizsgálni a rendszerszintű szintézis tervező által előre meghatározott követelményeit és prioritásait (PdPR-eket), valamint főbb lépéseit az általánosított rendszertervezési szempontok szerint. Mivel az SLS-algoritmus nem csak és kizárólag HMS rendszerek tervezésére alkalmas; érdemes magasabb absztrakciós szintről indulni. A későbbiekben ezt átültetjük a jelen dolgozat által tárgyalt SLS-algoritmusra és annak lépéseire.

2.1 A rendszerszintű szintézis tervező által meghatározott prioritásai és alapkövetelményei

A korábban is említett PdPR (Predetermined Priorities and Requirements), vagyis a tervező által előre meghatározott prioritások és követelmények két csoportra bonthatók; a gyengébb, illetve az erősebb kritériumokra, ahol a gyengébbeket hívjuk prioritásnak, az erősebbeket pedig követelményeknek. Ezeket a kényszereket a tervező támasztja a rendszerrel szemben, attól függően, hogy milyen tulajdonságú, és mennyi eszközzel szeretné elvégezni a folyamatot, továbbá milyen újraindítási idővel üzemeljen stb. Ezek a rendszerparaméterek befolyással lesznek az egyes rendszerszintű tervezőeszközökre, valamint az SLS végkimenetelére is. Az előbb említett kritériumokat úgy kell megválasztani, hogy a folyamat az algoritmusok számára kezelhető maradjon, ezért a PdPR-ek meghatározása rendkívül fontos előfeltételei az SLS-nek. Tekinthetjük ezt 0. lépésnek is, mivel a logikus, észszerű kritériumok felállítása nem a HLS algoritmusok feladata, hanem a tervezőé.

Vegyünk példának egy 50 pontos FIR szűrést, amelyet 40MHz órajelű mikrokontrollerek (pl. PIC18fxxxx) segítségével akarunk megvalósítani, és megszabunk kritériumnak egy 10μs-os pipeline újraindítási időt (átbocsátási tényezőt). Ez esetben a rendszerszintű szintézishez kiválasztott HLS tervező eszközök nem fogják tudni realizálni a kitűzött feladatot. Ilyenkor változtatnunk kell a korábban meghatározott kritériumok egy vagy akár több paraméterén is.

Ilyen, és az ehhez hasonló helyzetek esetére érdemes megkülönböztetni a prioritásokat és a követelményeket. Ugyanis amíg egy követelmény sérthetetlen, és minden esetben igaznak kell lennie az SLS által létrehozott rendszerre, addig a prioritásoknak nem kell feltétlenül érvényre jutniuk a tervezés folyamán, mivel azok betartatása bizonyos esetekben nem is lehetséges. Ilyenkor el kell döntenünk, hogy melyik előre meghatározott paraméter a fontosabb számunkra, és az algoritmusok ezt próbálják meg érvényesíteni a rendszeren. Az előbbi esetben, ha ez egy valósidejű (real-time) rendszer, a 10μs-os újraindítási időt követelménynek kell vennünk, a komponensprocesszorok számánál és

típusánál azonban már rugalmas kritériumokat kell szabnunk, ezért elsődleges prioritásúnak állíthatjuk be a mikrokontrollerek használatát, de ha nem érhető el megfelelő műveletvégző egység erre a célra, akkor célszerű lesz egy hasonló órajelű (40MHz) DSP (pl. dsPIC) használata. Jelen esetben, ennél az egyetlen FIR szűrésnél valószínűleg a prioritás nem fog érvényre jutni, mivel a HLS algoritmusok azt eredményezik, hogy a 10µs-os újraindítási idő mellett lehetetlen egyetlen mikrokontrollerrel megvalósítani a szóban forgó rendszert.

A tervezés során természetesen a PdPR-ek nemcsak a tervezendő folyamattól és azok feladataitól függenek, hanem egymástól is. Egy kritérium óhatatlanul befolyásol más kritériumokat, ezért szükséges meghatározni azt is, hogy az egymásra hatással lévő prioritások közül melyek a fontosabbak számunkra. A követelmények esetében ezt nem kell megtennünk, hiszen a követelmények olyan szigorú kritériumok, amelyek egyértelműen jellemzik a rendszert, és nem mondanak ellent egymásnak. Ezért sorrendet csak a prioritások esetében érdemes létrehozni, és csak azok között a prioritások között érdemes felállítani, amelyek ellentmondanak egymásnak (1. táblázat).

	Predetermined Priorities and Requirements | PdPR	
	Követelmények	Prioritások
TULAJDONSÁGOK		
Kritérium súlya	Gyengébb	Erősebb
A tervezés folyamata során változhat	Tilos változnia	Változhat
Érvényre jutás hiányában	Tervezési hiba következtében az SLS hibaüzenetet fog adni	Automatikusan változik, vagy a tervező újakat jelölhet ki
Az egymásra hatások figyelembe vétele szükséges a szintézis során		●
Gyakran használt PdPR-ek	• Komponensprocesszorok száma, kapacitása, minimális kitöltöttségük[1], párhuzamos feldolgozásra való alkalmasság • Pipeline működés szükségessége és a kívánt újraindítási idő megszabása • Kommunikációs teher figyelembevétele • Feladatok egyenletes elosztása a processzorok között	

● igen

1. táblázat: Az előre meghatározott követelmények és prioritások összefoglaló táblázata

[1] A minimális kitöltöttség hatással van az energiafelhasználásra

Ebben a dolgozatban tárgyalt HLS algoritmusok a komponens processzorok közötti egyenletes feladatfelosztás, és az általuk előidézett kommunikációs teher csökkentésének mértékét engedik állítani. Ezeknek a prioritásoknak a beállítása mint későbbiekben látni fogjuk – rendkívül fontos szerepet játszik a rendszertervezés során.

A PdPR-ek meghatározását a kritériumok típusain kívül további két esetre bonthatjuk: amikor egy már megvalósított rendszert akarunk optimalizálni, illetve amikor egy teljesen új rendszert hozunk létre a feladat megvalósításához (2. táblázat).

	Meglévő rendszer optimalizálása	Új rendszer tervezése
PDPR-EK		
Komponens processzorok száma	○	●
Komponens processzorok kapacitása	○	●
Komponens processzorok közötti kommunikációs csatornák kialakítása	○	●
Újraindítási idő figyelembe vétele	●	●
Egyenlő feladatelosztás	●	●

○ Előre meghatározott PdPR ● Szabadon meghatározható PdPR

2. táblázat: Új és meglévő rendszer tervezése a PdPR-ek szempontjából

Az első esetben értelemszerűen a követelmények lesznek túlsúlyban, hiszen már egy létező hardverhez kell szintetizálni a tervezendő feladatot, ezáltal adottak lesznek az egyes komponensprocesszorok típusai, a számuk és a közöttük kialakított kommunikációs csatornák. Ezáltal az SLS-algoritmus mozgástere lecsökken, a prioritások kevesebb lehetőséget adnak a szintetizáló algoritmusok viselkedésének befolyásolására, mint ha egy teljesen új rendszert kellene tervezünk. Az utóbbi esetben, amikor a legelejéről kezdjük a tervezési folyamatot, a követelmények sokkal változatosabbak lehetnek, hiszen az is meg lehet szabva, hogy a tervező milyen komponens processzorokat használjon, és milyen egyéb tervezési megkötéseket vegyen figyelembe stb. A különböző lehetőségek kihatnak a prioritásokra is, hiszen ha például egy tervezőnek adott egy olyan technológia, amely rendkívül gyors kommunikációt képes biztosítani az egyes komponens processzorok között, akkor kevésbé fontos a kommunikációs teher minimalizálása, mint a feladatok egyenletes elosztása a műveletvégző egységek között.

A fentiek alapján kijelenthetjük, hogy a PdPR-ek meghatározása nagymértékben befolyásolja az SLS kimenetelét.

2.2 A rendszerszintű szintézis általános lépései

A rendszertervezés során az egymást befolyásoló döntések és folyamatok nem mindig láthatók előre, az SLS-algoritmusa nem tud minden körülményt figyelembe venni a kialakuló rendszer tulajdonságaira vonatkozóan. Ezáltal nincs lehetőség determinisztikus algoritmusok kifejlesztésére, ezért a jelenlegi rendszerszintű tervezés nagy részben intuitív jelleggel, próbálgatással történik. Létezik ugyan több olyan magas szintű logikai szintézer eszköz (HLS Tool), amely próbálja ezeket a problémákat kiküszöbölni és elfogadható megoldást találni. Ugyanakkor a későbbiekben látni fogjuk, hogy az egyes HLS eszközök közötti kapcsolatok nem annyira szorosak, hogy az algoritmusok képesek legyenek felismerni és számításba venni más algoritmusok kritikus pontjait. Erre a gondolatmenetre építve fogjuk bemutatni a későbbiekben a rendszerszintű szintézis új megközelítési módját (2.3 fejezet), előbb azonban tekintsük át az iparban már régebb óta használt, hagyományos lépésekkel megvalósított rendszerszintű szintézist[12,19].

1. lépés – A feladat vagy probléma megfogalmazása

Az SLS első lépése a probléma megfogalmazása, amely feladattól függően sokféle lehet. A leggyakoribb az egy adott programnyelven leírt forráskód, de kiindulhatunk pszeudokódból vagy akár egy folyamatgráfból is. A legfontosabb az, hogy a feladatokat és a feladatokat összekötő kapcsolatokat definiáljuk. Továbbá bemeneti paraméterként szolgálnak a korábban tárgyalt, tervező által meghatározandó PdPR-ek is.

HMS rendszerek esetében általában adott egy magas szintű nyelven (pl. C, Java) leírt program, ez a tervező által relatív könnyen érthető. Ennek ellenére a bemenetet képezhetik alacsonyabb szintű programozási nyelvek (pl. Assembly), vagy a nagyon magas szintű programozási nyelvek (pl. Python, JavaScript vagy a kereskedelmi szoftverek nyelvei). Tekintettel arra, hogy egy C nyelvű forráskódot a fordító is egyfajta Assembly nyelvre fordítja le, illetve a magasabb szintű programozási nyelveket is előszeretettel vezetik vissza ismertebb magas szintű nyelvekre, az algoritmusnak − bizonyos kiegészítésekkel − nem jelenthet problémát bármely szintű nyelven írt program rendszerszintű szintézise.

2. lépés – A folyamatgráf előállítása

A következő lépés a folyamatgráf előállítása. A folyamatgráfok segítségével a multiprocesszoros feladatok egyszerűen reprezentálhatók. A gráf pontjai jelképezik a folyamatban elvégzendő feladatokat, a hozzátartozó súlyok pedig meghatározzák a

feladatok komplexitását, és ez különböző műveletvégző egységek esetén különböző számítási időt jelent. A gráf élei a feladatok közötti adatkommunikációt jelöli. Az élekhez tartozó súlyok értékét a kommunikációban részt vevő adatok mérete határozza meg; minél nagyobb az átküldendő adat, annál több időt vesz igénybe az átküldése. Ez természetesen függ a kommunikációs interfész tulajdonságaitól (sebesség, integráltsági fok stb.), amelyekért az egyes komponens processzorok vagy maga a hardver tervezője a felelős. Azok között a feladatok között, ahol nem történik adatkommunikáció, szintén behúzhatunk éleket, és feljegyezhetünk őket, ezeknek a súlya zérus lesz [14,15].

Fontos, hogy a folyamatgráf egyetlen irányított kört (hurkot) se tartalmazzon, mert ha az irányított kör egyes feladatait különböző komponens processzorok valósítják meg, akkor a köztük felmerülő kommunikáció és az adatfüggés jelentősen lassítaná az adott feladat feldolgozását. Annak érdekében, hogy ezeket elkerüljük, az irányított körök feladatait egyetlen nagy, oszthatatlan feladatként kell tekintenünk, ezáltal megszűnik az irányított kör [2,13,20].

A magasszintű logikai szintézis folyamatában elkülöníthető a folyamatgráfok forráskódból történő előállítása, ezért a továbbiakban feltételezzük, hogy a gráf már rendelkezésre áll a dekompozíciós algoritmus számára, ezért ezzel a dolgozat következő részeiben nem foglalkozunk.

3. lépés – Dekompozíció (Decomposing)

Az SLS-algoritmus harmadik és egyik legfontosabb lépése a dekompozíció. A dekompozíció során a gráf pontjait felosztjuk két vagy több csoportra, amely csoportok műveleteit az egyes komponensprocesszorok fogják elvégezni. Legfontosabb bemeneti paramétere maga a gráf, amelyet az előző lépésből kaptunk meg. Ennek több formája lehet a dekompozíciós eljárástól függően: szomszédossági vagy illeszkedési mátrix a hozzájuk tartozó pontsúlyvektorral vagy egyszerűen a pontok felsorolása a súlyukkal, a hozzájuk csatlakozó többi ponttal és az őket összekötő élek súlyával.

Sokféle dekompozíciós algoritmus létezik [1,6,13,23,37,38], amelyeknek a legkülönfélébb céljuk van. Nem mindegyik algoritmusnak ugyanaz a feladata, ezáltal a folyamatgráf mellett – amelynek minden esetben rendelkezésre kell állnia bármely dekompozíciós algoritmus számára – a további bemeneti paraméterek a legkülönbözőbbek lehetnek. Ezek a paraméterek a tervező által meghatározott PdPR-ek megvalósítása miatt szükségesek, és csak kevés módszer tud olyan szintű rugalmasságot biztosítani, amely képes a PdPR minden prioritását és követelményét kezelni.

A fentiek értelmében beláthatjuk, hogy determinisztikus optimális megoldás nem létezik, hiszen a hatékony dekompozíció erősen függ a PdPR-től, így a tervező számára ez határozza meg a megfelelő algoritmus kiválasztását. Továbbá a tervezési feladat NP-teljes probléma, és senki sem garantálja, hogy a rendszer szintézisére csak egyetlen jó megoldás létezik. A dolgozat további részében az algoritmus számára a feladatok egyenletes eloszlását és a kommunikáció minimalizálását tűzzük ki célul, valamint a kettő

közötti kapcsolatot vizsgáljuk meg, figyelembe véve az egyes szegmensek által reprezentált komponensprocesszorok számát [8,9].

A dekompozíciós algoritmus kimeneteként megkapjuk, hogy az egyes számítási műveleteket melyik komponensprocesszornak kell elvégeznie ahhoz, hogy a tervező kívánsága szerinti PdPR teljesüljön.

4. lépés – Ütemezés (Scheduling)

Az ütemezés feladata a dekompozíció által meghatározott szegmensek, részfeladatok végrehajtási idejének pontos behatárolása. Ez egyben meghatározza minden részfeladat kezdeti időpontját, figyelembe véve az előírt időzítések betartását, például pipeline esetben a lappangási vagy újraindítási időt.

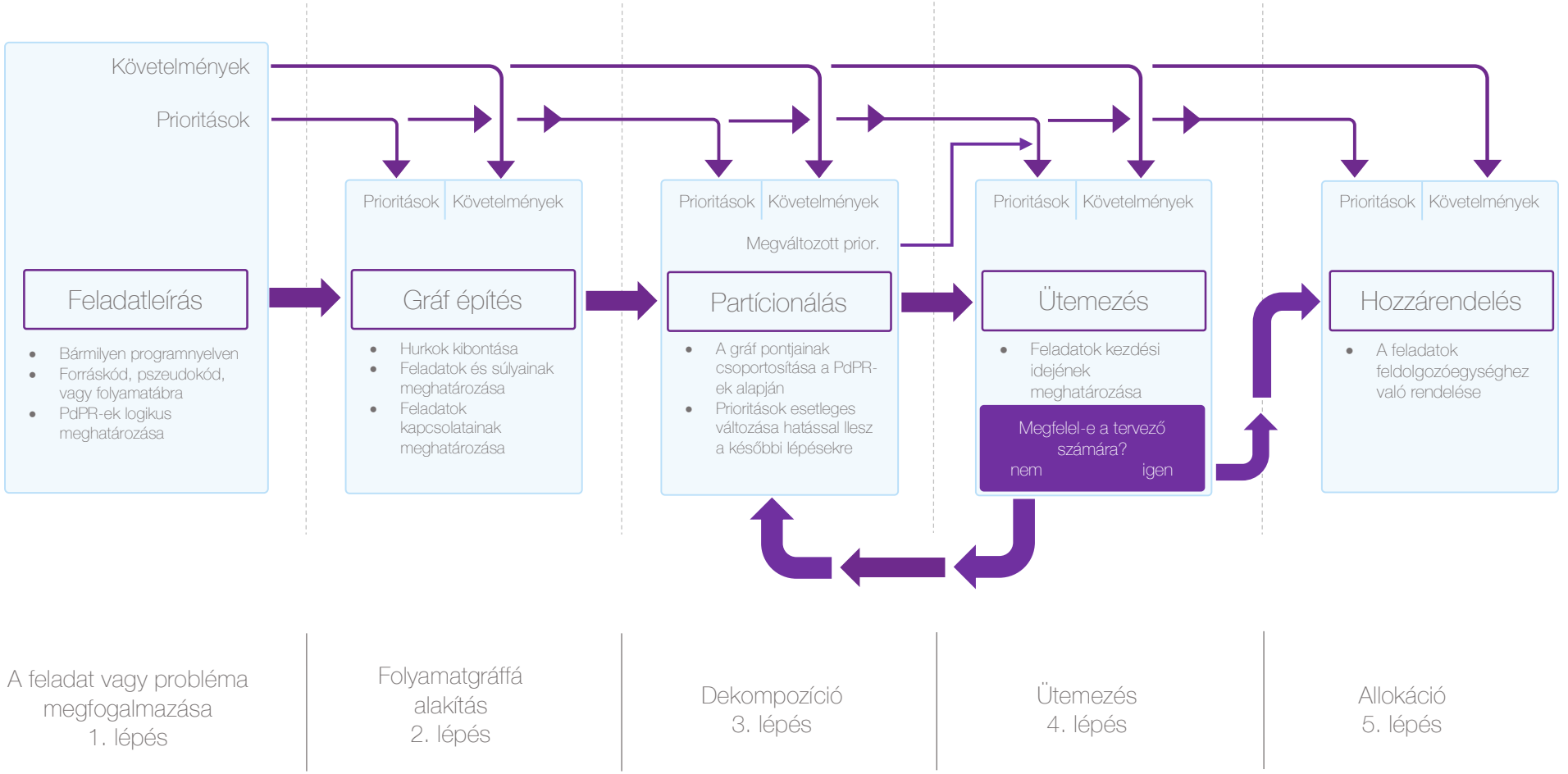
Ütemezőalgoritmusból szintén sokféle létezik [12,17,18,27], mivel a PdPR-ek a dekompozíció mellett az ütemezést is alapvetően meghatározzák, hiszen itt jutnak érvényre azok az időzítéssel kapcsolatos beállítások, amelyeket prioritásként vagy követelményként az 1. lépésben meghatároztunk. Ebben a dolgozatban továbbiakban az Erővezérelt Ütemező (Force Directed) algoritmus alapjaira fogunk támaszkodni, amely hatékonyságában és tervezési idejében elfogadottan [35,36] jó eredményeket mutat.

5. lépés – Allokáció (Allocating)

Az allokáció célja a beütemezett részműveletek konkrét feldolgozóegységhez való hozzárendelése. Az allokáció az ütemező algoritmustól függően általában a dekompozíció eredményétől is függ. HMS rendszer esetében azonban az egyes részfeladatokat speciális műveletvégző egységekkel célszerű megvalósítani, ezért fontos, hogy ezeket a nekik megfelelő komponens processzorokhoz társítsuk.

2.3 Rövid áttekintés

A fenti lépések alapvető fontosságúak a rendszerszintű szintézis során. A feladat megfogalmazásából kiindulva (1. lépés) megállapítjuk a feldolgozandó feladatot, amely az SLS algoritmusainak egy fix bemeneti paramétere lesz. Ha az SLS-t egy bonyolult, összetett logikai függvénynek tekintjük, akkor a feldolgozandó feladatot akár független változóként is értelmezhetjük. A feldolgozandó feladat leírásának és reprezentálásának módja lépésről lépésre folyamatosan változik; folyamatgráfot hozunk létre belőle (2. lépés), szegmensekre bontjuk (3. lépés), ütemezzük és allokáljuk (4–5. lépés) . Ez a folyamat a 2. ábrán látható.



2. ábra: A hagyományos rendszerszintű szintézis lépései

További független változóként beszélhetünk a PdPR követelményeiről, hiszen azokat azért támasztottuk a rendszerrel szemben, hogy biztosan teljesüljenek, ezért nem függhetnek semmitől, és nem változhatnak meg a tervezés folyamata során. Ezért a PdPR követelményei az egyes lépésekre nézve mindig ugyanazok maradnak, és több esetben is felhasználjuk őket, anélkül, hogy egymás hatásairól tudnának. Ennek a későbbiekben még fontos szerepe lesz .

Az SLS harmadik fontos bemeneti paraméterei a PdPR prioritásai. Ezeket, a kezdeti értékeiket leszámítva, alapjában véve függő változónak tekinthetjük, hiszen az, hogy érvényre jut-e egy adott prioritás, az magától a feladattól, a PdPR-ek követelményeitől és a különböző HLS algoritmusoktól függ.

A 3.1 fejezetben megvizsgáljuk az egyes lépéseket és az ezeket összekötő kapcsolatok problémáit, amelyek gátolják az imént megismert alapvető SLS lépések hatékony működését. Ezt követően felvázolunk egy új rendszerszintű szintézis metódust, amely figyelembe veszi mindazon a problémákat, amelyekről a következő alfejezetben lesz szó.

3 A rendszerszintű szintézis új megközelítési módja

3.1 A hagyományos SLS gyenge pontjai

Mint korábban már említettük: az egyes HLS algoritmusok közötti kapcsolatok nem annyira szorosak, hogy az algoritmusok képesek legyenek felismerni és számításba venni más algoritmusok kritikus pontjait. Ez azonban szükséges lenne, hiszen egy-egy lépés, egy-egy HLS algoritmus önmagában csak egyetlen részfolyamatát végzi el az SLS-nek, így hiába szolgáltatnák az egyes részfeladatokra a PdPR-eket, figyelembe véve a lehető legjobb megoldást, a rendszerszintű tervezés végeredményében nem feltétlenül a tervező számára kevésbé kedvező megoldás jelenhet meg.

1. A prioritások visszacsatolásának problémája

Mint azt korábban már láthattuk, a PdPR-ek két csoportba oszthatók, prioritásokra és követelményekre. A követelmények a tervezés lépései során nem okoznak problémát, hiszen független változónak tekintettük az SLS bemeneti paraméterlistáján.

A prioritások azonban nem problémamentesek a tervezés menetére nézve, mivel ezeket magától a folyamatosan alakuló rendszertől (tervezési feladattól), valamint a követelményektől tettük függővé. Ugyanakkor a rendszer is függ a prioritástól, hiszen a cél az, hogy a prioritások érvényre jussanak, ha azonban nem tudnak, akkor ezt a rendszernek vissza kell jeleznie a tervezési folyamat során, és újra kell futtatnia az egyes tervezési lépéseket más prioritási beállításokkal. A kölcsönös függés által visszacsatolások jönnek létre a tervezési módszerben, amelyek nemcsak az adott prioritás paraméterét

felhasználó SLS lépésre hat vissza, hanem a többi tervezési lépésre is. Ezáltal előtérbe kerülnek az intuitív tervezési módszerek, amelyek leginkább a visszacsatolás miatt iterációs jellegűek (2. ábra).

A fentieket tekintve a feladatfeldolgozási folyamatot elképzelhetjük egy olyan bonyolult sorrendi hálózathoz hasonlított tervezési modellként, ahol az egyes állapotok jelentik az előző fejezetben megismert lépéseket, és egy-egy stabil állapot jelképezi az egyes fázisok részeredményét. Tervezés közben a visszacsatolások következtében az algoritmusok folyamatosan iterálnak mindaddig, amíg nem kerülnek egy stabil állapotba, amelyben az előírt prioritás teljesül, vagy az adott priortás megváltoztatására van szükség.

Ez az egyik legfontosabb oka annak, hogy iterációs eljárásokat alkalmazunk az SLS algoritmusokban és azok összekapcsolásához, mivel a visszacsatolásokat nem lehet determinisztikus módszerekkel, zárt alakban kezelni.

Beláthatjuk továbbá, hogy ennek elkerülése érdekében nem lehet minden tervezési paramétert követelményként venni, mert túlságosan merevvé válna a tervezés menete, az algoritmusoknak minimálisra csökkenne a mozgástere, és nem lenne képes meghatározni a tervező számára legmegfelelőbb megoldást.

2. A dekomopzíció és az ütemezés kapcsolatának iteratív problémája

A prioritások visszacsatolásához hasonlóan, ebben a problémában is az iteráció okozza a legfőbb gondot, itt azonban nem közvetlenül a PdPR-re fókuszálunk, hanem a prioritások okozta iterációs kapcsolatra az algoritmusok között.

Az előzőek alapján világossá vált, hogy az egy-egy lépést megvalósító algoritmus addig iterál, míg a PdPR-eknek eleget nem tesz. Mivel azonban a prioritások és azoknak beteljesülése vagy be nem teljesülése kihat a teljes tervezési folyamatra, ezért az egyes lépésekben nem ismeretes, hogy a későbbi vagy korábbi lépést figyelembe véve a végeredmény mennyire lesz kedvező vagy kedvezőtlen. Lehetséges például, hogy egy kevésbé jó dekompozíciós eredményt ütemezve sokkal jobb megoldást kapunk allokáció után, mint ha az egyes lépések algoritmusai egymást figyelmen kívül hagyva a lehető legjobb részmegoldásra törekednek.

A hagyományos intuitív és iterációs tervezési módszerek a fentiekben vázolt problémát az alábbi módon kezelik: ha egy lépés eredménye nem felel meg az előírásoknak és a PdPR-eknek, akkor megváltoztatott prioritások értékeivel újrakezdjük a tervezést egy korábbi ponttól vagy magától a folyamat elejétől. Ez azonban a tervezési idő jelentős megnövekedéséhez vezet, amely komplexebb rendszerek esetében különösen kritikussá válhat.

A fenti két problémát összegezve szükségünk lenne egy olyan tervezési módszerre, amely képes jelentősen csökkenteni az egyes tervezési lépések futásainak a számát, ezáltal az iterációs tulajdonságok hátrányai is kisebb súllyal határoznák meg a tervezés sikerét. Fontos az is, hogy a folyamat során egymásra hatással lévő PdPR-eket minél kevesebb lépés használja fel, így az egymásra hatás mértéke is csökken.

Ezért új nézőpontból kell szemlélnünk a rendszerszintű szintézist.

3.2 Az új megközelítési módszer lépései

A rendszertervezés legkényesebb pontjai a dekompozíció és az ütemezés. A PdPR-ek ezekben a lépésekben határozták meg a legtöbb olyan paramétert, amely a kialakult rendszer viselkedéséért felelős. Ezek a leglényegesebb kölcsönhatások, amelyeket a PdPR váltott ki, és a legtöbb iterációval ezeknél a lépéseknél kell számolni. Ezért célszerű lenne, ha a két tervezési lépést valamilyen módszerrel összevonnánk, így egy tervezési lépésen belül juthatnának érvényre a prioritások a dekomopozíció és az ütemezés tulajdonságait egyszerre figyelembe véve.

Erre azonban a hagyományos dekompozíciós eljárások nem adnak lehetőséget kimeneti paraméterlistáik miatt, hiszen ezek diszkrét eredményeket adnak arra vonatkozóan, hogy az egyes feladatokat mely komponens processzoroknak kell megvalósítaniuk, ám többletinformációt nem tartalmaznak. Ezért az ilyen dekompozíciós eljárások helyett olyan analizáló algoritmusra van szükség, melynek kimenete bővebb információ-halmazt ad az ütemező algoritmus számára.

Ezáltal az ütemező algoritmusnak olyan bemeneti paraméterei lehetnek, amelyek a PdPR-ektől függően többféleképpen értelmezhető variációkat tartalmaznak a particionáló részalgoritmus számára. Így maga döntheti el a tervezési kritériumokat kielégítő szegmenseket, és a szegmentálási módszert. Beláthatjuk, hogy egy ilyen adatstruktúráknak a kezelésére olyan ütemezési eljárásokat kell kialakítani, amelyek képesek az analizáló algoritmus kimenetét kezelni. Ezek alapján felvázolhatjuk az új rendszerszintű szintézis módszerének a lépéseit:

1. lépés – A feladat vagy probléma megfogalmazása

Az SLS első lépése ebben a módszerben megegyezik a hagyományos módszerekkel. A kiindulási pont egy tetszőleges programnyelven írt forráskód, pszeudokód vagy akár maga a folyamatábra, valamint a PdPR meghatározása.

2. lépés – A folyamatgráf előállítása

A folyamatgráf előállítása szintén megegyezik az SLS hagyományos módszereivel. A gráfot illeszkedési vagy szomszédossági mátrixszal definiáljuk a harmadik lépés számára.

3. lépés – Analizálás

Az új tervezési módszer első nagy különbsége a harmadik lépésben található, ahol a dekompozíciót felváltja az analizálás. Ezt a feladatot módszertől függően sokféle algoritmus feldolgozhatja. Ebben a dolgozatban – mint később látni fogjuk – a gráf spektrumát határozzuk meg, és az ebből nyert adatokat továbbítjuk a következő lépés számára. Az első, speciálisan erre a célra kifejlesztett algoritmus a Súlyozott Normalizált Vágás (WNCut), amellyel a 4. fejezetben fogunk foglalkozunk.

4. lépés – Dinamikus ütemezés

Az analízist megvalósító Súlyozott Normalizált Vágás kimeneteként vektorokat kapunk, amelyeket a dinamikus ütemezés során használunk fel. A dinamikus ütemezés az analízis eredményeként létrejövő vektorokból nyert adatokat használja fel az egyes feladatok indítási idejének a meghatározásához. Ennek következtében nincs szükség dekompozíciós algoritmus újra futtatására más prioritási paraméterekkel. A PdPR közvetlenül ebben a lépésben határozza meg a rendszer azon tulajdonságait, amelyekért a dekompozíciós és az ütemező algoritmus futása során lenne a felelős, így a prioritások visszacsatolása ugyan meg nem szűnik, de sokkal jobban kezelhetővé válik.

A második, speciálisan erre a célra kifejlesztett algoritmus egy módosított ütemező algoritmus, amelynek neve Dinamikus Erővezérelt Ütemező. Mint a nevéből is kiderül, ez az algoritmus az Erővezérelt Ütemező (Force Directed) algoritmusra épít, amellyel az 5. fejezetben foglalkozunk.

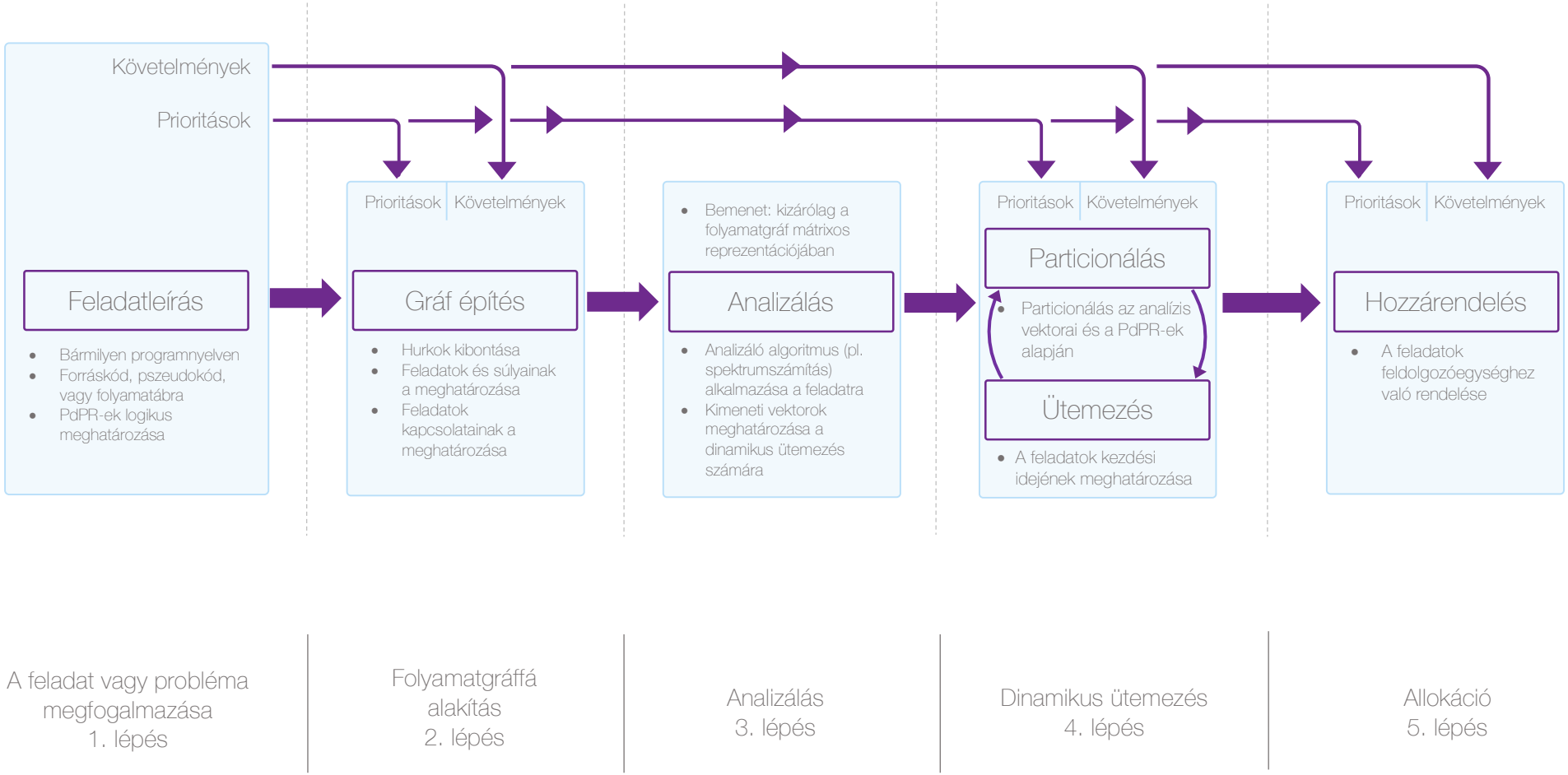
5. lépés – Allokáció

Az új tervezési módszer utolsó lépése, az allokáció szintén megegyezik a korábban megismert általános SLS módszer lépésével.

Az előbbiekben láthattuk az új rendszerszintű szintézis módszerének a lépéseit. Előnye, hogy jelentősen csökkenti az iterációk okozta problémákat, melyekről a korábbi alfejezetekben szó volt, és próbál javítani rajtuk. Ezáltal a PdPR-ek egymásra hatásait egyszerűbben és nagyobb hatásfokkal lehet kezelni, így a tervezési idő jelentősen csökkenhet (3. ábra).

Hátránya, hogy az analizálás és a dinamikus ütemezés lépései közé szükség van egy olyan interfészre, amely által a fenti említett lépéseket megvalósító algoritmusok a rendszerszintű szintézis folyamatában szabadon cserélhetők. Mivel ez egy új megközelítési módnak tekinthető, ezért a jelen dolgozat még csak egy-egy algoritmust és az együttműködésüket biztosító interfészt mutatja be részletesen.

A következő fejezetek részletesen leírják az analizálást és a dinamikus ütemezést megvalósító algoritmusokat.



3. ábra: Az új rendszerszintű szintézis lépései

4 Az analizáló algoritmus

A dolgozat következő alfejezetei a dekompozíciós algoritmust kiváltó eljárással foglalkozik, az analizáló folyamattal. Az új eljárás segít lecsökkenteni a tervezés során a PdPR-ek miatt létrejövő iterációs hurkok számát és az általuk elvégzendő feladat mennyiségét. Ennek eredményeképpen az analizáló algoritmus nem használja fel konkrétan a PdPR-eket, de megteremti a lehetőséget arra, hogy a kialakuló adatstruktúrában alkalmazzuk őket azért, hogy a tervező által megszabott prioritások és követelmények érvényre juthassanak.

Az eljárás egy ismert dekompozíciós elven alapul, amelyet *Normalizált Vágásnak* [1] hívnak. A Normalizált Vágást minimalizálási elve és rendszerleírása alapján választottam ki az analizáló algoritmus kiindulópontjaként. A minimalizálási elv a Rayleigh-hányadoson alapul, a rendszerleírás pedig a Laplace-mátrix szerint történik, amely a minimalizálandó függvény zárt alakját biztosítja. Ez a módszer még önmagában nem használható fel analizálásra, azonban az eljárásnak köszönhetően, azonban a régi algoritmus gondolatmenetére egy teljesen újat ültetve megkaphatjuk az új SLS módszeréhez szükséges, a Normalizált vágás után elnevezett Súlyozott Normalizált vágást.

Az új eljárás két részletben mutatjuk be:

1) *Az eredeti algoritmus – A Normalizált Vágás* (Ncut) című fejezetben megismerhetjük azt az algoritmust, amelyből a fejlesztés során kiindultunk. Ez Jianbo Shi és Jitendra Malik, a Berkeley Egyetem kutatóinak a munkája [1], amelyet képszegmentálásra használnak fel. Az eredeti algoritmus megismerése azért fontos, mert annak matematikai levezetésére támaszkodva fejlesztettük ki és bizonyítottuk az új analizáló módszert.

2) *A módosított algoritmus – A Súlyozott Normalizált Vágás* (WNCut) című fejezetben kifejtjük az előzőben megismert algoritmus problémáit a HMS rendszerek esetén történő particionálás szempontjából. Bemutatjuk az új algoritmusnak, a Súlyozott Normalizált vágásnak a lépéseit, levezetését és bizonyítását, amelyek a problémák korrigálására szolgálnak.

Az így létrejött partícionáló módszer legutolsó lépését, maguknak a feladatoknak a szegmensekbe való sorolását már a dinamikus ütemező algoritmus hajtja végre, amely így egyszerre figyelembe tudja venni nem csak az ütemezési, hanem a dekompozíciós szempontokat is felhasználva a PdPR-eket. A WNCut alkalmazásával a komponensprocesszorok közötti kommunikáció és a feladatok egyenletes elosztásának a prioritása beállíthatóvá válik a tervezés későbbi lépéseiben.

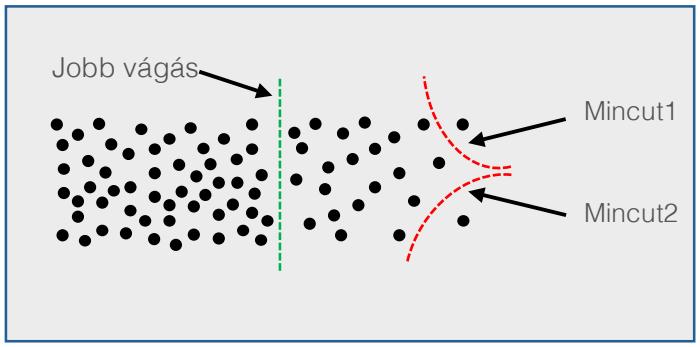
A továbbiakban feltételezzük, hogy rendelkezésünkre áll a feladatleírásból előállított folyamatgráf, és annak mátrixos formulával (illeszkedési vagy szomszédossági mátrix) történő pontos definiálása, valamint az egyes pontok által reprezentált feladatok súlyaiból alkotott vektor (pontsúlyvektor).

4.1 Az eredeti algoritmus – A Normalizált Vágás

Egy $\mathrm{G} = (V, E)$ gráf pontjai két különálló halmazba sorolhatók, A-ra, illetve B,-re, melyekre igaz, hogy $A \cup B = V$ és $A \cap B = \emptyset$. Ez az A és B halmaz közötti élek elhagyását jelenti. Feltételezve, hogy az egyes élek adatok kommunikációját reprezentálják előre meghatározott élsúllyal, az eltávolított élek súlyainak összege a két diszjunkt halmaz közötti teljes adatkommunikáció. Gráfelméleti nyelven ezt a *vágásnak* hívjuk:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v). \tag{1}$$

Könnyen beláthatjuk, hogy a PdPR-ek prioritásainak egyik fő paramétere az egyes szegmensek közötti kommunikáció csökkentésének a mértéke, ezért dekompozíciós szempontból akkor járunk jól, ha ennek a vágás értékének a minimalizálására törekszünk.



4. ábra: Olyan eset, ahol a minimális vágás rossz eredményt ad

Számos olyan gráf particionáló algoritmus létezik [4,6,7], amely megtalálja a $\min_{E} cut(A, B)$-t, az ún. minimális vágást. Ugyanakkor több tudományos munka is beszámolt már arról [39], hogy a minimális vágás kritériuma kedvez a gráf egyes izolált pontjai számára. Ez nem meglepő, hiszen az imént definiált vágás (1) nem veszi figyelembe az egyes szegmensek élsúlyainak összegét és egymáshoz viszonyított számukat. A 4. ábra egy ilyen szituációt mutat be. Feltételezve, hogy az élek súlya fordítottan arányos a pontok közötti távolsággal, láthatjuk, hogy a Mincut1 vagy a Mincut2 vágásra relatív kicsi értéket kapunk. Beláthatjuk, hogy bármelyik vágás értéke a jobb térrészen elvégezve kisebb eredményt fog hozni, mint ha középen vágnánk két partícióra az ábrát. Ez az SLS számára is fontos, hiszen ez – mint majd később látni fogjuk – közvetetten a feladatok csoportonkénti mennyiségére is utal, amely a PdPR prioritások szempontjából a kommunikációhoz hasonlóan szintén kritikus kérdés.

Annak érdekében, hogy $cut(A,B)$ A, és B csoport közel azonos számú pontot tartalmazzon, és egyik csoportot se egyetlen izolált pont alkossa, új típusú disszociációs leírást vezetünk be a gráfok particionálásának érdekében. Az élek összsúlyának minimalizálása helyett az élek súlyát vesszük az összes élsúly hányadosában. A terminológiában ezt hívják Normalizált Vágásnak:

$$\mathrm{Ncut}(\mathrm{A},\mathrm{B}) = \frac{\mathrm{cut}(\mathrm{A},\mathrm{B})}{\mathrm{assoc}(\mathrm{A},\mathrm{V})} + \frac{\mathrm{cut}(\mathrm{A},\mathrm{B})}{\mathrm{assoc}(\mathrm{B},\mathrm{V})}, \tag{2}$$

ahol $\mathrm{assoc}(\mathrm{A},\mathrm{V}) = \sum_{\mathrm{u}\in\mathrm{A},\mathrm{t}\in\mathrm{V}} \mathrm{w}(\mathrm{u},\mathrm{t})$ az összes kapcsolat az A csoport és a G gráf összes pontja között, és $\mathrm{assoc}(\mathrm{B},\mathrm{V})$ is hasonlóan definiálható. Az $\mathrm{Ncut}(\mathrm{A},\mathrm{B})$ bevezetésével A, és B csoporthoz tartozó élek súlyának összege közel azonos lesz, ez nem hoz létre izolált pontokat. A fenti példát tekintve a Normalizált Vágás értéke nagyobb lesz Mincut1 és Mincut2 esetében.

Papadimitriou által bizonyított [3], hogy a Normalizált Vágás egy NP-teljes probléma, még az egyszerűbb gráfok esetében is. Ugyanakkor megtapasztalhatjuk, hogy a gyakorlatban létezik olyan diszkrét közelítő megoldás, amely hatékonyan fel tudja osztani a folyamatgráfot két csoportra a normalizált vágás követelményének megfelelően.

4.1.1 A partíciók meghatározása

Adott egy gráf V pontjainak két partíciója A és B, illetve legyen $\boldsymbol{x}$ egy $\mathrm{N} = \dim(\boldsymbol{x}) = |V|$ dimenziós jelzővektor, amely $\boldsymbol{x}_i = 1$, ha az i-edik pont az A csoporthoz tartozik, és -1, ha nem. Legyen $\boldsymbol{d}(\mathrm{i}) = \sum_j w(i,j)$ azon élek súlyainak összege, amelyek az i-edik pontból indulnak ki, és az összes többi pontba tartanak. Az $\boldsymbol{x}$, illetve a $\boldsymbol{d}$ vektorok definíciójának ismeretében újra felírhatjuk az $Ncut(A,B)$ Normalizált Vágást, mint:

$$\begin{aligned} Ncut(A,B) &= \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} = \\ &= \frac{\sum_{\boldsymbol{x}_i>0,\boldsymbol{x}_j<0} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i>0}\boldsymbol{d}_i} + \frac{\sum_{\boldsymbol{x}_i<0,\boldsymbol{x}_j>0} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i}. \end{aligned} \tag{3}$$

Legyen $\mathbf{D}$ egy $\mathrm{N}\times\mathrm{N}$-es diagonális mátrix, amelynek az átlója rendre a d vektor elemeit tartalmazza, $\mathbf{W}$ pedig szintén egy $\mathrm{N}\times\mathrm{N}$-es szimmetrikus mátrix, amelyre igaz, hogy $\mathbf{W}(i,j) = w_{ij}$ (ez lesz a már korábban említett szomszédossági mátrix), valamint

$$k = \frac{\sum_{x_i>0} \boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i}, \tag{4}$$

és $\mathbf{1}$ legyen egy $\mathrm{N}\times 1$-es vektor, amelynek mindegyik eleme 1-et tartalmaz.

Vezessünk be $\boldsymbol{x}_i > 0$ és $\boldsymbol{x}_i < 0$ esetén új jelzővektorokat. Könnyen beláthatjuk, hogy $\frac{\mathbf{1}+\boldsymbol{x}}{2}$, illetve $\frac{\mathbf{1}-\boldsymbol{x}}{2}$ jelzővektorok kiváltják a pontok két csoportba osztására szolgáló $\boldsymbol{x}_i$ és $\boldsymbol{x}_j$ vektorokat, hiszen -1-et, illetve $+1$-et behelyettesítve 0-át kapunk, amely segít kiválasztani azokat az éleket és ezáltal pontokat, amelyek az egyik partíció részét fogják képezni. Az új jelzővektorok nevezőjét átrendezhetjük az egyenlet másik oldalára, mivel ez a minimalizálási problémát nem fogja befolyásolni, ugyanakkor egyszerűsíti az egyenletünket.

Ezeknek ismeretében átírhatjuk az $Ncut(\boldsymbol{x})$ függvényünket $4[\mathrm{Ncut}(\boldsymbol{x})]$-re a következőképpen:

$$4[Ncut(\boldsymbol{x})] = \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{k(\mathbf{1}^T\mathbf{D}\mathbf{1})} + \frac{(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{(1-k)(\mathbf{1}^T\mathbf{D}\mathbf{1})}, \tag{5}$$

ez tovább alakítható:

$$4[Ncut(\boldsymbol{x})] = \frac{\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x} + \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{k(1-k)(\mathbf{1}^T\mathbf{D}\mathbf{1})} + \frac{2(1-2k)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{k(1-k)(\mathbf{1}^T\mathbf{D}\mathbf{1})}. \tag{6}$$

A továbbiakban legyen

$$\begin{aligned} \alpha(\boldsymbol{x}) &= \boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}, \\ \beta(\boldsymbol{x}) &= \mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}, \\ \gamma &= \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}, \end{aligned} \tag{7}$$

és

$$M = \mathbf{1}^T\mathbf{D}\mathbf{1}. \tag{8}$$

Ezekkel a behelyettesítésekkel élve tovább alakíthatjuk az egyenletet:

$$= \frac{(\alpha(\boldsymbol{x}) + \gamma) + 2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} =$$

$$\frac{(\alpha(\boldsymbol{x}) + \gamma) + 2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} - \frac{2(\alpha(\boldsymbol{x}) + \gamma)}{M} + \frac{2\alpha(\boldsymbol{x})}{M} + \frac{2\gamma}{M}. \tag{9}$$

Az utolsó értéket elhagyhatjuk, hiszen $\gamma = 0$, mivel a $(\mathbf{D} - \mathbf{W})$ mátrix pozitív szemidefinit. Ezután a tagokat összevonva a következő kifejezést kapjuk:

$$= \frac{(1-2k+2k^2)(\alpha(\boldsymbol{x}) + \gamma) + 2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} + \frac{2\alpha(\boldsymbol{x})}{M} =$$

$$\frac{\frac{(1-2k+2k^2)}{(1-k)^2}(\alpha(\boldsymbol{x}) + \gamma) + \frac{2(1-2k)}{(1-k)^2}\beta(\boldsymbol{x})}{\frac{k}{(1-k)}M} + \frac{2\alpha(\boldsymbol{x})}{M}.$$

Legyen $b = \frac{k}{1-k}$, és mivel $\gamma = 0$, ezért:

$$= \frac{(1+b^2)(\alpha(\boldsymbol{x}) + \gamma) + 2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} =$$

$$= \frac{(1+b^2)(\alpha(\boldsymbol{x}) + \gamma)}{bM} + \frac{2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} + \frac{2b\gamma}{bM}.$$

Az $\alpha(\boldsymbol{x})$, $\beta(\boldsymbol{x})$, γ, és M kifejezéseket visszahelyettesítve:

$$= \frac{(1+b^2)(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x} + \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{2(1-b^2)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} +$$

$$+ \frac{2b\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{2b\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} =$$

$$= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b(\mathbf{1}^T\mathbf{D}\mathbf{1})} + \frac{b^2(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{b(\mathbf{1}^T\mathbf{D}\mathbf{1})} +$$

$$+ \frac{2b(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b(\mathbf{1}^T\mathbf{D}\mathbf{1})} =$$

$$= \frac{[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]}{b(\mathbf{1}^T\mathbf{D}\mathbf{1})}.$$

Válasszunk $\boldsymbol{x}$ jelzővektor helyére egy új, $\boldsymbol{y}$ vektort: $\boldsymbol{y} = (\mathbf{1}+\boldsymbol{x}) - b(\mathbf{1}-\boldsymbol{x})$. Fontoljuk meg, hogy ha $\boldsymbol{y}$ vektort használjuk a pontok csoportba rendezésére, akkor

$$\boldsymbol{y}^T\mathbf{D}\mathbf{1} = \sum_{\boldsymbol{x}_i>0} \boldsymbol{d}_i - b\sum_{\boldsymbol{x}_i<0} \boldsymbol{d}_i = 0, \tag{10}$$

mivel $b = \frac{k}{1-k} = \frac{\sum_{x_i>0}\boldsymbol{d}_i}{\sum_{x_i<0}\boldsymbol{d}_i}$, azaz az egyik csoportban szereplő élek számosságát ki lehet fejezni a másik csoportban lévő élek számosságának és b-nek szorzatával. Miután levezettük, hogy $\boldsymbol{y}^T\mathbf{D}\mathbf{1} = 0$, lássuk be, hogy $\boldsymbol{y}^T\mathbf{D}\boldsymbol{y} = b\mathbf{1}^T\mathbf{D}\mathbf{1}$, hiszen mindent összevetve

$$\begin{aligned}\boldsymbol{y}^T\mathbf{D}\boldsymbol{y} &= \sum_{\boldsymbol{x}_i>0}\boldsymbol{d}_i + b^2\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i = b\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i + b^2\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i = \\ &= b\left(\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i + b\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i\right) = b\mathbf{1}^T\mathbf{D}\mathbf{1}.\end{aligned} \tag{11}$$

Ha az eredeti feladatot tekintjük, az $Ncut(\boldsymbol{x})$ függvényt a fentiek alapján átírhatjuk a következő minimalizálási problémára:

$$\min_{\boldsymbol{x}}\big(Ncut(\boldsymbol{x})\big) = \min_{\boldsymbol{y}}\left(\frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}\right), \tag{12}$$

megfelelve a következő feltételeknek: $\boldsymbol{y}^T\mathbf{D}\mathbf{1} = 0$ és $\boldsymbol{y}(i) \in \{1, -b\}$, tehát az $\boldsymbol{y}$ jelzővektor az i-edik pontra vonatkozóan nem a diszkrét -1 vagy 1 értéket fogja adni. Erre a későbbiekben még visszatérünk. Az $\boldsymbol{y}$ vektor szerinti minimalizálási probléma már egy jó közelítéssel megoldható feladat.

Vegyük észre, hogy a fenti kifejezés egy Rayleigh-hányados [5]. Ha $\boldsymbol{y} \subset \mathbb{R}$, akkor minimalizálhatjuk (12)-es egyenletet a generált sajátértékrendszer megoldásával,

$$(\mathbf{D}-\mathbf{W})\boldsymbol{y} = \lambda\mathbf{D}\boldsymbol{y}. \tag{13}$$

Mindemellett két kényszerünk is van $\boldsymbol{y}$-ra nézve, amelyek az $\boldsymbol{x}$ jelzővektor feltételeiből adódnak. Először is gondoljunk az $\boldsymbol{y}^T\mathbf{D1} = 0$ feltételre. Láthatjuk, hogy ez a feltétel automatikusan kielégül az generált sajátértékrendszer kiszámításával.

Vegyük a (13)-as egyenletet, transzformáljuk át standard sajátérték rendszerre, és lássuk be, hogy az előbbi feltétel kielégül. A (13) egyenletet felírhatjuk a következőféleképpen:

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z} = \lambda \boldsymbol{z}, \tag{14}$$

ahol $\boldsymbol{z} = \mathbf{D}^{\frac{1}{2}}\boldsymbol{y}$. Könnyedén ellenőrizhetjük, hogy $\boldsymbol{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$ a 0-ás sajátértékhez tartozó sajátvektor, melyet az áttranszformált sajátértékrendszerből írtunk fel. Ezek alapján az első feltétel igaznak bizonyul, hiszen $\boldsymbol{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$-t behelyettesítve a (14) egyenletbe a következőt kapjuk:

$$\begin{aligned}
\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{\frac{1}{2}}\mathbf{1} &= \lambda\mathbf{D}^{\frac{1}{2}}\mathbf{1} \\
\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{1} &= \lambda\mathbf{D}^{\frac{1}{2}}\mathbf{1} \\
(\mathbf{D}-\mathbf{W})\mathbf{1} &= \lambda\mathbf{D}\mathbf{1}.
\end{aligned} \tag{15}$$

Továbbá tudjuk, hogy $(\mathbf{D}-\mathbf{W})$, akárcsak $\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}$, pozitív szemidefinit mátrixok. A $(\mathbf{D}-\mathbf{W})$ mátrixot másnéven Laplace-mátrixnak is szokták nevezni. A Laplace mátrixban a sorok és az oszlopok összege 0, tehát ha a (15) egyenlet alsó sorát tekintjük, beláthatjuk, hogy az egyenlet bal oldala ekvivalensen nullvektort eredményez, tehát $(\mathbf{D}-\mathbf{W})\mathbf{1} = \mathbf{0}$, ami csak akkor lehet, ha az egyenlet jobb oldalán $\lambda = 0$.

Ennélfogva $\boldsymbol{z}_0$ a legkisebb sajátértékhez, tehát a 0-hoz tartozó sajátvektora lesz a (14) egyenletből generált sajátértékrendszernek és az összes egyenlethez tartozó sajátvektor merőleges lesz egymásra. Feltételezve, hogy a sajátértékek szerint növekvő sorrendben számozzuk a sajátvektorokat, $\boldsymbol{z}_1$ a második legkisebb sajátértékhez tartozó sajátvektor, amely merőleges $\boldsymbol{z}_0$-ra. Ha az előbbi állításokat visszahelyettesítjük a (13) egyenletben szereplő generált sajátérték rendszerbe, a következőket kapjuk:

1) $\boldsymbol{y}_0 = \mathbf{1}$ a legkisebb sajátértékhez tartozó sajátvektor, és
2) $\boldsymbol{z}_1^T\boldsymbol{z}_0 = 0$, behelyettesítve $\boldsymbol{z} = \mathbf{D}^{\frac{1}{2}}\boldsymbol{y}$, illetve $\boldsymbol{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$ kifejezéseket, láthatjuk, hogy $\boldsymbol{y}_1^T\mathbf{D1} = 0$, ahol $\boldsymbol{y}_1$ a második legkisebb sajátvektora a (13) egyenletnek.

Vizsgáljunk meg egy egyszerű tényt a Rayleigh-hányadossal kapcsolatban[5]:

> Legyen **A** egy valós, szimmetrikus mátrix. Azzal a kikötéssel, hogy $\boldsymbol{x}$ ortogonális, tehát $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{j-1}$ vektorok merőlegesek egymásra, a hányados $\frac{\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ minimalizálható a következő legkisebb sajátvektorral, $\boldsymbol{x}_j$-vel, és a minimum értéke a hozzá tartozó sajátérték, λ_j.

A fentieket figyelembe véve:

$$\boldsymbol{z}_1 = \arg.\min_{\boldsymbol{z}^T\boldsymbol{z}_0=0}\left(\frac{\boldsymbol{z}^T\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z}}{\boldsymbol{z}^T\boldsymbol{z}}\right), \tag{16}$$

és következetesen:

$$\boldsymbol{y}_1 = \arg.\min_{\boldsymbol{y}^T\mathbf{D}\mathbf{1}=0}\left(\frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}\right). \tag{17}$$

Így a generált sajátvektor rendszer második legkisebb sajátértékhez tartozó sajátvektora fogja adni a valós megoldást a Normalizált Vágás problémához. Sajnos azonban ez az algoritmus nem szolgálja ki teljes mértékben az eredeti problémánkat, mivel az $\mathbf{y}$ vektor egyes elemei nem a két diszkrét értéket veszik fel, hanem egy valós számot 1 és $-\mathrm{b}$ között. Később látni fogjuk, hogy ez a módszer kifejezetten előnyös a dinamikus ütemező algoritmus számára, hiszen a diszkrét $\{1, -1\}$ értékek alapján történő szegmentálás a későbbiekben nem engedne lehetőséget arra, hogy rugalmasan kezelje a korábban megismert PdPR-eket.

Nem szabad azonban figyelmen kívül hagynunk, hogy ez a matematikai módszer ilyen formában még nem felel meg a rendszerszintű szintézis dekompozíciót kiváltó lépéséhez, az analízishez.

4.2 A módosított algoritmus – A Súlyozott Normalizált Vágás

4.2.1 A Normalizált Vágás hibái

Az előző fejezetben láthattuk, hogy bár a Normalizált Vágás NP-teljes probléma, egy jó közelítő módszerrel meg tudjuk oldani a minimalizálási feladatot. Ez az algoritmus azonban nem ültethető át módosítás nélkül a gyakorlati alkalmazásba HMS rendszerek

esetén, mert számos problémát tartalmaz, és ez alkalmatlanná teszi a folyamatgráf analizálására.

Shi és Malik a Normalizált Vágás algoritmusát kép szegmentálás céljából fejlesztették ki, melyben a pixelek közötti színátmenetet a gráf egyes éleinek a súlyával jelölik. A pontoknak önmagukban nincsen tulajdonságuk, ezért a Normalizált Vágás egyenletében ($\mathrm{Ncut}(\boldsymbol{x})$) nem szerepelnek szignifikáns tulajdonságaik szerint, csupán az $\boldsymbol{x}_i$ jelzővektorokban, amelyek az egyenletben figyelembe vett élek kiválasztására szolgálnak.

A multiprocesszoros rendszerek esetében azonban a folyamatgráf pontjai tartalmazzák az egyes feladatok számítási igényeit, amelyet a $\boldsymbol{v}$ pontsúlyvektorból tudunk kiolvasni a $v(i)$ függvény segítségével. Ez többnyire megszabja a processzor műveletvégzési idejét, így ügyelnünk kell arra, hogy mennyi feladattal látjuk el az egyes komponensprocesszorokat.

Olyan algoritmusra van szükség a folyamatgráf analizálásához, amely figyelembe veszi nemcsak a vágás értékét és az egyes csoportban található élsúlyok összegének az egyenlőségét, hanem azt is, hogy a két szegmensbe kerülő pontok összsúlya közel azonos legyen. Ezek alapján ha $\forall\, v(i) \in V$, akkor:

$$\sum_{v(i)\in A} v(i) - \sum_{v(i)\in B} v(i) \approx 0, \tag{18}$$

feltéve, hogy $A \cap B = \emptyset$. Ez visszavezet a 4.1 fejezetben tárgyalt izolált pontok problémájához, hiszen beláthatjuk, hogy a (18) egyenlet, illetve a $\min(Cut(A,B))$ egyidejű teljesülésével szintén garantálni tudjuk, hogy a vágás ne eredményezzen izolált pontokat.

4.2.2 A Súlyozott Normalizált Vágás – Elméleti megközelítés

Modellezzük a problémát a 4.1 fejezetben már megismert ponthalmazzal. Adott egy $\mathrm{G}' = (V, E)$ gráf, $w_{ij} \in E$ és legyen

$$w_{ij} \simeq d(i,j), \tag{19}$$

ahol $d(i,j)$ az i-edik és a j-edik pont közötti távolságot jelöli. Minél közelebb van egymáshoz két pont, az őket összekötő él súlya annál nagyobb. Beláthatjuk, hogy ha $\mathrm{P}_i(x,y)$ és $\mathrm{P}_j(x,y)$ pontot a végtelenségig közelítjük egymáshoz, távolságuk infinitezimálisan kicsi lesz, határértéke 0:

$$\lim_{\mathrm{P}_i \to \mathrm{P}_j} d(i,j) = 0, \tag{20}$$

a fordított arányosság miatt pedig:

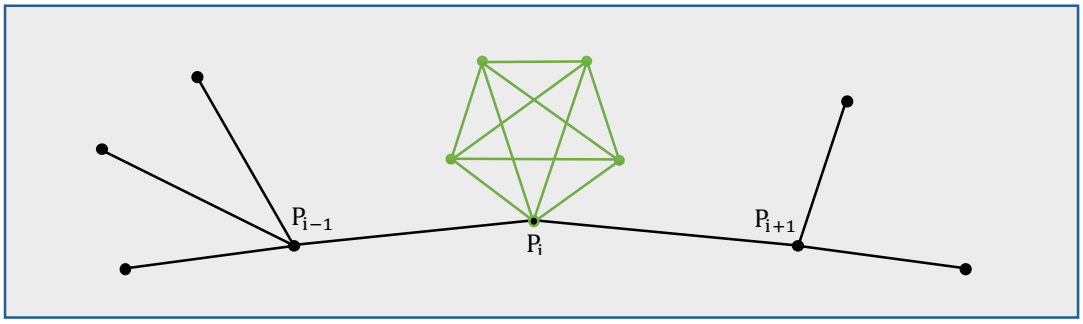
$$\lim_{d(i,j)\to 0} w_{ij} = +\infty. \tag{21}$$

A fentiek értelmében ha a P_i pont tart P_j-hez, $\mathrm{w_{ij}}$ a $+\infty$-be divergál, ennek következtében az $\mathrm{Ncut(x)}$ algoritmus a pontokat azonos szegmensbe fogja besorolni, tehát P_i, illetve $\mathrm{P_j}$ biztosan egy csoportba kerülnek, mivel a korábban tárgyalt generált sajátérték rendszer megoldása során (13) az $\boldsymbol{x}$ vektorban a hozzá tartozó értékük azonos lesz.

Tételezzük fel, hogy $\mathrm{P_i}$ pont infinitezimálisan kicsi környezetében pontosan $v(i) - 1$ darab pont helyezkedik el, tehát az eredeti pontot is számításba véve $\mathrm{P}_{i,v(\mathrm{i})}, \mathrm{P}_{i,v(i)-1}$, $\mathrm{P}_{i,v(i)-2} \dots \mathrm{P}_{i,2}, \mathrm{P}_{i,1}$, amely pontok mind kapcsolatban vannak egymással, és az őket összekötő élek súlyát $w_{i,k} = +\infty$-nek tekintjük. Vegyük észre, hogy ezek a pontok egy $\mathrm{K}_{v(i)}$ teljes gráfot alkotnak, amely az előbbi $\mathrm{P}_{i,k}$ pontokkal kibővített G gráf klikke, tehát:

$$\mathrm{K}_{v(i)}(V_K, E_K) \subseteq \mathrm{G}(V_G, E_G), \tag{22}$$

amely valódi részgráf az eredeti gráf $\mathrm{P_i}$ pontját helyettesíti $v(i)$ súlyt reprezentálva. Vizsgáljuk meg az eredeti G', az új G gráf és $\mathrm{K}_{v(i)}$ pontjai, valamint élei közötti kapcsolatot. Vegyük az alábbi ábrán látható egyszerű példát:



5. ábra: Teljes gráf növesztése $\mathrm{P_i}$ pontra

Láthatjuk, hogy $\mathrm{P_i}$ egyaránt szerepel a zöld színnel jelölt $\mathrm{K}_{v(i)}$ és az eredeti, fekete színnel jelölt G' gráfban is, ahol súlyát $|E_K| = v(i) = 5$-re állítottuk be. Beláthatjuk, hogy

$E_K \cap E_{G'} = \{P_i\}$, tehát a két részgráf egyetlen közös pontja P_i, vagyis a Normalizált Vágásban bevezetett $\boldsymbol{x}$ jelzővektorunk $\boldsymbol{x}(i)$-edik pontjához kapcsolódó élek nem szűnnek meg, így azok továbbra is kijelölhetők maradnak.

Lássuk be, hogy a G gráf minimális vágása sosem fogja tartalmazni V_K egyetlen elemét sem, hiszen ha

$$V_K \cap V_{G'} = \emptyset,$$
$$V_K \cup V_{G'} = V_G, \tag{23}$$

hiszen V_K és $V_{G'}$ diszjunkt halmazok, és

$$\lim_{i \to j} \forall w_{ij}\big|_{w_{ij} \in V_K} = +\infty, \tag{24}$$

akkor

$$\min_{\boldsymbol{x}} Cut(G, \boldsymbol{x}) = \min_{\boldsymbol{x}} Cut(G', \boldsymbol{x}). \tag{25}$$

A fentiek értelmében két kényszerünk is van arra nézve, hogy a minimális vágás nem fogja elvágni $K_{v(i)}$ teljes részgráfot.

1) V_K és V_G diszjunkt halmazok, ezért ha az eredeti G' gráfhoz hozzávesszük a teljes gráfot, V_G halmaz elemei, és a rajtuk végzett minimális vágás sértetlen marad.
2) Ha a teljes részgráfot összekötő élek értékét közelítőleg $+\infty$-nek tekintjük, a minimalizáló algoritmus sohasem fogja őket kiválasztani, hiszen G' gráf élei továbbra is rendelkezésre állnak az algoritmus számára, amely minimalizálásra törekszik.

Vegyük azonban figyelembe, hogy egy gráf élsúlyának értéke a valóságban nem lehet $+\infty$, ami az előbb vizsgált 2. kényszernek mond ellent. A Súlyozott Normalizált Vágás – Gyakorlati megközelítés című alfejezetben bebizonyítjuk, hogy tudunk egy olyan felső becslést adni a klikk éleinek súlyára, amely minden esetben elegendő lesz ahhoz, hogy ne vágja el a teljes részgráfot. E mellett megoldást adunk az imént vizsgált élsúlyok pontok szerinti kezelésére, amely az egyenletes pontsúlyok alapján történő elosztást biztosítja.

Ezen kívül számolnunk kell még egy fontos problémával. Vegyük észre, hogy a 4. fejezetben a vágás normalizálására az $\mathrm{assoc}(A, V)$, illetve az $\mathrm{assoc}(B, V)$ kifejezések szolgálnak, amelynek definíciója $\mathrm{assoc}(X, V) = \sum_{u \in X, t \in V} w(u, t)$. Ennek értelmében a (2)-

es egyenlet minimalizálása nem a gráf pontjait osztja egyenletesen két részre, hanem arra ügyel, hogy az $\frac{1}{\mathrm{assoc(A,V)}} + \frac{1}{\mathrm{assoc(B,V)}}$ értéke minimális legyen. A feltétel az élsúlyok összegének egyenletes eloszlását biztosítja, és ez nem feltétlenül egyezik meg a pontsúlyok egyenletes elosztásával, ezért más módon kell garantálni az pontsúlyok összegének az egyenlőségét az egyes szegmensekben. A probléma megoldásával a következő fejezetben foglalkozunk.

4.2.3 A Súlyozott Normalizált Vágás – Gyakorlati megközelítés

Egy $\mathrm{G} = (V, E)$ gráf pontjai két diszjunkt halmazra bonthatók A-ra, illetve B-re. A korábban megismert $\mathrm{Ncut(A, B)}$ analógiájára felírhatjuk az új, *Súlyozott Normalizált Vágást*, (Weighted Normalized Cut) $\mathrm{WNcut(A, B)}$:

$$WNcut(A,B) = \frac{cut(A,B)}{assoc(A,V) + sum(A)} + \frac{cut(A,B)}{assoc(B,V) + sum(B)}\ , \qquad (26)$$

ahol a $cut(A,B) = \sum_{u\in A, k\in B} w(u,k)$-t jelöli, tehát a vágásban szereplő élek összsúlya, $\mathrm{assoc(A, V)} = \sum_{\mathrm{u\in A, t\in V}} \mathrm{w(u, t)}$ az összes kapcsolat az A csoport és a G gráf összes pontja között, $sum(A) = \sum_{i\in A} v(i)$, vagyis az A csoportban szereplő pontsúlyok összege. Az $assoc(B,V)$ és $sum(B)$ kifejezéseket hasonlóképpen definiáljuk azzal a kivétellel, hogy nem az A, hanem a B csoportra vonatkoztatjuk őket.

Idézzük fel a 4.1.1 fejezet elején bevezetett $\mathrm{N} = |V|$ dimenziós x jelzővektort, amely 1 vagy -1 diszkrét értékeket vesz fel a gráf pontjainak hovatartozását illetően. Definiáltuk $\mathbf{d}'(\mathrm{i})$ vektort, mint azon élek súlyainak az összegét, amelyek az i-edik pontból indulnak ki, és az összes többi pontba tartanak. A fentiek ismeretében a (3)-as egyenletet átírhatjuk a következőképpen:

$$WNcut(\boldsymbol{x}) = \frac{\sum_{x_i>0, x_j<0} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{x_i>0} \boldsymbol{d'}_i + \sum_{x_i>0} \boldsymbol{v}_i} + \frac{\sum_{x_i<0, x_j>0} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{x_i<0} \boldsymbol{d'}_i + \sum_{x_i<0} \boldsymbol{v}_i}. \qquad (27)$$

Legyen $\mathbf{D}'$ a már korábban megismert $\mathrm{N}\times\mathrm{N}$-es diagonális mátrix, amelynek az átlója rendre a $\mathbf{d}'$ vektor elemeit tartalmazza, $\mathbf{D}$ pedig egy $\mathrm{M}\times\mathrm{M}$-es diagonális mátrix, ahol

$$\mathrm{M} = \sum_{i=1}^{|E|} v(i), \qquad (28)$$

és átlója rendre $\mathbf{d}$ vektor elemeit tartalmazza. A $\mathbf{d}$ vektor $\mathbf{d}'$ és $\mathbf{d}''$ elemeit tartalmazza, mely $\mathbf{d}''$ az i-edik pontra növesztett, $\mathrm{K}_{v(i)}$ klikk pontjaiból kiinduló élek súlyának az összegeit tartalmazza:

$$\boldsymbol{d''}_i = \sum_{w_{jk} \in V_{K,i}} w_{jk}. \tag{29}$$

Ennek analógiájára a $\mathbf{W}'$ szintén egy N×N-es szimmetrikus mátrix, amely a Normalizált Vágás algoritmusából megismert G gráf szomszédossági mátrixa, és $\mathbf{W}(\mathrm{i},\mathrm{j}) = \mathrm{w_{ij}}$, tehát a gráf élsúlyait tartalmazza. $\mathbf{W}$ pedig legyen egy M×M-es mátrix, mely tartalmazza az i-edik pontra növesztett, $\mathrm{K}_{v(i)}$ klikk éleit is, a d vektor szerinti sorrendben elhelyezkedve.

Vegyük észre, hogy $\mathrm{K}_{v(i)}$ teljes gráf éleinek száma kölcsönösen egyértelműen függ $|E_K| = v(i)$-től, vagyis a gráf pontjainak számától:

$$|V_K| = \frac{|E_K| \cdot (|E_K| - 1)}{2}. \tag{30}$$

Lássuk be, hogy a kölcsönösen egyértelmű függvénykapcsolat következtében az eredeti G gráf pontsúlyait az egyes pontokra illesztett teljes gráf élei képesek reprezentálni. Az összefüggés azonban négyzetes, ezért minden egyes, a gráfban szereplő klikk esetén, az őket összekötő élek száma csoportonként összeszámolva azt eredményezné, hogy a pontok nem lennének egyenlően elosztva. Ezt korrigálni tudjuk, ha a G-ben szereplő klikkek élsúlyát nem konstansnak tekintjük, hanem külön-külön értéket adunk nekik:

$$w_{jk,\,\mathrm{K}_i} = \frac{2}{\left|\mathrm{E}_{K,i}\right| - 1} \cdot w_{min}, \tag{31}$$

ahol w_{min} az a minimális élsúly, amely mellett az algoritmus nem fogja elvágni a klikk éleit. A fenti egyenlet alapján pedig

$$v(i) = \sum_{w_{ij} \in V_{K,i}} w_{jk,\,\mathrm{K}_i}. \tag{32}$$

A fentiek értelmében ha $\mathbf{D}$ és $\mathbf{W}$ mátrixban a klikkek éleinek súlyát kicseréljük $\frac{2}{|\mathrm{E}_K|-1} \cdot w_{min}$-re, ekkor a pontok súlyát a pontok helyére helyettesített részgráf éleinek összsúlya fogja reprezentálni.

A klikkek alkalmazása és az élsúlyok korrigálása után a $4[\mathrm{Ncut}(\boldsymbol{x})]$ függvényt:

$$4[Ncut(\boldsymbol{x})] = \frac{(\mathbf{1}' + \boldsymbol{x}')^T(\mathbf{D}' - \mathbf{W}')(\mathbf{1}' + \boldsymbol{x}')}{k'(\mathbf{1}'^T\mathbf{D}'\mathbf{1}')} + \frac{(\mathbf{1}' - \boldsymbol{x}')^T(\mathbf{D}' - \mathbf{W}')(\mathbf{1}' - \boldsymbol{x}')}{(1-k')(\mathbf{1}'^T\mathbf{D}'\mathbf{1}')}, \quad (33)$$

átírhatjuk $4[WNcut(\boldsymbol{x})]$-re a következőféleképpen:

$$4[WNcut(\boldsymbol{x})] = \frac{(\mathbf{1} + \boldsymbol{x})^T(\mathbf{D} - \mathbf{W})(\mathbf{1} + \boldsymbol{x})}{k(\mathbf{1}^T\mathbf{D}\mathbf{1})} + \frac{(\mathbf{1} - \boldsymbol{x})^T(\mathbf{D} - \mathbf{W})(\mathbf{1} - \boldsymbol{x})}{(1-k)(\mathbf{1}^T\mathbf{D}\mathbf{1})}. \quad (34)$$

Az új függvényben $k = \frac{\sum_{x_i>0} \boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i}$, és $\mathbf{1}$ legyen egy $\mathrm{M}\times 1$-es vektor. Az $\boldsymbol{x}$ jelzővektor új dimenziója $\dim(\boldsymbol{x}) = \mathrm{M}$.

A korábban már megismert (25) egyenlet feltételei a gyakorlati megvalósítás során sérülnek, hiszen nem választhatunk élsúlynak $+\infty$-t. Létezik azonban olyan módszer, amellyel felső becslést adhatunk arra nézve, hogy mekkorának kell lennie a klikk élsúlyainak, annak érdekében, hogy $\min_{\boldsymbol{x}} Cut(\mathrm{G}, \boldsymbol{x}) = \min_{\boldsymbol{x}} Cut(\mathrm{G}', \boldsymbol{x})$ igaz maradjon. A definícióból adott, hogy $\mathbf{D}'$ mátrix az egyes pontokhoz tartozó élek összsúlyait tartalmazza. Beláthatjuk, hogy ha a gráfban található összes él súlyát összeadjuk, akkor még legkedvezőtlenebb esetben is:

$$\mathrm{tr}(\mathbf{D}') \geqq (\mathbf{1} + \boldsymbol{x})^T(\mathbf{D}' - \mathbf{W}')(\mathbf{1} + \boldsymbol{x}), \quad (35)$$

tehát a $\mathbf{D}'$ mátrix trace-e biztosan nagyobb lesz, mint a vágás értéke a lehető legrosszabb esetben, amikor az összes él szerepel a vágásban. Ennélfogva, ha a klikk száma kisebb, mint az eredeti G' gráf minimális vágása, és a klikk éleinek $\mathrm{tr}(\mathbf{D}')$-t választjuk, a minimális vágás ez esetben sem fogja tartalmazni V_K elemeit.

Az 5.2 fejezetben szereplő 1. kényszer, illetve az előbb tett becslés alapján a minimális vágás nem fogja tartalmazni $\mathbf{W} \setminus \mathbf{W}'$ halmaz elemeit, ezáltal a klikkek éleinek nincsen hatása a kibővített G gráfra. Most lássuk be, hogy a Normalizált Vágás sem fogja megváltoztatni G gráf minimális vágását. Legyen:

$$\Theta(\boldsymbol{x}) = (\mathbf{1} + \boldsymbol{x})^T(\mathbf{D}' - \mathbf{W}')(\mathbf{1} + \boldsymbol{x}), \quad (36)$$

ahol a $\Theta(\boldsymbol{x})$ függvény valamely $\boldsymbol{x}$ vektor által kijelölt vágás értékét jelenti. Továbbá ha definiáljuk:

$$\phi(\boldsymbol{x}) = k = \frac{\sum_{x_i>0} \boldsymbol{d'}_i}{\sum_i \boldsymbol{d'}_i}, \tag{37}$$

akkor a (34) egyenletből felírhatunk egy egyenlőtlenséget (38). Az alábbi egyenlőtlenségben vizsgáljunk meg két esetet, amelyben két különböző felosztás szerint analizálunk. Az $\boldsymbol{x_1}$ jelzővektort tartalmazó esetben a vágás nem tartalmazza egyetlen klikk élét sem, $\boldsymbol{x_2}$ esetében pedig a vágást úgy jelöltük ki, hogy az egyik klikk valamely élét elvágja.

Igazoljuk, hogy optimális esetben egy klikk éleit is tartalmazó vágás értéke soha nem lehet kisebb, mint ha a vágás egyetlen klikk élét sem tartalmazza, vagyis a Rayleigh-hányados felírásával generált sajátértékrendszer minimalizálása után az algoritmus a bal oldali esetet fogja adni:

$$\begin{gathered}\frac{\Theta(\boldsymbol{x_1})}{\mathrm{tr}(\mathbf{D})\phi(\boldsymbol{x_1})} + \frac{\Theta(\boldsymbol{x_1})}{\mathrm{tr}(\mathbf{D})(1-\phi(\boldsymbol{x_1}))} < \frac{\Theta(\boldsymbol{x_2})}{\mathrm{tr}(\mathbf{D})\phi(\boldsymbol{x_2})} + \frac{\Theta(\boldsymbol{x_2})}{\mathrm{tr}(\mathbf{D})(1-\phi(\boldsymbol{x_2}))} \\ \frac{\Theta(\boldsymbol{x_1})}{\mathrm{tr}(\mathbf{D})\phi(\boldsymbol{x_1})(1-\phi(\boldsymbol{x_1}))} < \frac{\Theta(\boldsymbol{x_2})}{\mathrm{tr}(\mathbf{D})\phi(\boldsymbol{x_2})(1-\phi(\boldsymbol{x_2}))}.\end{gathered} \tag{38}$$

Mivel $\mathrm{tr}(\mathbf{D}) \in \mathbb{R}^+$, ezért azt kiejthetjük az egyenlőtlenségből, ezzel is igazolva, hogy a minimális vágás értéke nem függ a kibővített gráf klikkjeitől. Továbbá felhasználjuk azt a tényt, hogy $\boldsymbol{\phi(x)} \cup (\mathbf{1} - \boldsymbol{\phi(x)}) \cup \Theta(\boldsymbol{x}) \in \mathbb{R}^+$:

$$\frac{\Theta(\boldsymbol{x_1})}{\phi(\boldsymbol{x_1})(1-\phi(\boldsymbol{x_1}))} < \frac{\Theta(\boldsymbol{x_2})}{\phi(\boldsymbol{x_2})(1-\phi(\boldsymbol{x_2}))} \tag{39}$$

Lássuk be, hogy $\boldsymbol{x_2}$ vágás esetén $\Theta(\boldsymbol{x_2})$ értéke minimum:

$$\Theta(\boldsymbol{x_2}) \geqq \Theta(\boldsymbol{x_1}) + \mathrm{tr}(\mathbf{D}'), \tag{40}$$

mivel a két vágás között az a különbség, hogy $\boldsymbol{x_2}$ tartalmazza valamelyik klikk legalább egy élét, és ennek eredményeképpen $\boldsymbol{x_2}$ vágás belevág a teljes részgráfba.

$$\frac{\Theta(\boldsymbol{x_1})}{\Theta(\boldsymbol{x_1}) + \mathrm{tr}(\mathbf{D'})} < \frac{\phi(\boldsymbol{x_1})\big(1-\phi(\boldsymbol{x_1})\big)}{\phi(\boldsymbol{x_2})\big(1-\phi(\boldsymbol{x_2})\big)} \tag{41}$$

Ha $\boldsymbol{x_1}$-t és $\boldsymbol{x_2}$-t a lehető legrosszabbnak választjuk, vagyis majdnem az összes élet kijelöljük velük, az egyenlőtlenség bal oldalán a lehető legnagyobb érték maximum $0{,}5$ lehet, hiszen $\mathrm{tr}(\mathbf{D}') \geqq \Theta(\boldsymbol{x})$, ezért a legkedvezőtlenebb esetben $\frac{\mathrm{tr}(\mathbf{D}\prime)}{\mathrm{tr}(\mathbf{D}\prime)+\mathrm{tr}(\mathbf{D}\prime)} = 0{,}5$. Ha a legkedvezőtlenebb esetet tekintjük, tehát mindkét jelzővektort csupa egyre állítjuk be, akkor az egyenlet jobb oldalán:

$$\frac{\phi(\boldsymbol{x_1})(1-\phi(\boldsymbol{x_1}))}{\phi(\boldsymbol{x_2})(1-\phi(\boldsymbol{x_2}))} \approx 1, \tag{42}$$

amely nagyobb $0{,}5$-nél. A minimalizáló algoritmus $\phi(\boldsymbol{x_2}) = (1-\phi(\boldsymbol{x_2}))$-re fog törekedni, és ebben az esetben a nevezőben $0{,}25$ található, akkor kiszámolható, hogy mekkora az a felosztási arány, amelynél az egyenlőtlenség már nem lesz igaz. Számítások alapján a pontok $15\% - 85\%$-os felosztási arányánál rosszabb eredmények esetében nem fog teljesülni az egyenlőtlenség, és ez a minimalizálási probléma tulajdonságaiból adódóan a gyakorlatban soha nem fog előfordulni, hiszen egyenlő felosztásra törekszünk.

A fenti levezetésből láthatjuk, hogy $\mathrm{w_{min}} = \mathrm{tr}(\mathbf{D})$-re érdemes beállítani, figyelembe kell azonban vennünk a korábban tárgyalt teljes gráf pontjai és élei közötti összefüggést. Az egyes éleket úgy kell meghatározni, hogy a legkisebb pontszámú klikkhez tartozó élek súlyai $\mathrm{tr}(\mathbf{D})$ legyenek. Ahhoz, hogy ez teljesüljön, a (31) egyenletben szereplő w_{min}-t át kell írni $w_{min} \coloneqq (\max_i v(i) - 1)\, w_{min}$-re, ezáltal biztosíthatjuk a fent leírt követelményeket. Ezek alapján az egyes klikkek élsúlyai a következőképpen határozhatók meg:

$$w_{jk,\,\mathrm{K}_i} = \frac{2\,(\max_i v(i) - 1) \cdot \mathrm{tr}(\mathbf{D})}{\left|\mathrm{E}_{K,i}\right| - 1}. \tag{43}$$

A kibővített gráf élsúlyainak ismeretében felírható $\mathbf{D}$ és $\mathbf{W}$ mátrixból, az új M dimenziós $\boldsymbol{x}$ jelzővektor segítségével a (34) egyenlet, amely a Normalizált Vágással megegyező algoritmussal tovább alakítható $b = \frac{k}{1-k} = \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}$ bevezetésével a következőre:

$$WNcut(\boldsymbol{x}) = \frac{[(\mathbf{1}+\boldsymbol{x}) - b(\mathbf{1}-\boldsymbol{x})]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+\boldsymbol{x}) - b(\mathbf{1}-\boldsymbol{x})]}{b(\mathbf{1}^T\mathbf{D}\mathbf{1})}. \tag{44}$$

A 4. fejezetben megismert módszerek segítségével a fenti egyenlet átírható:

$$WNcut(\boldsymbol{y}) = \frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}, \tag{45}$$

$\boldsymbol{y} = (\mathbf{1}+\boldsymbol{x}) - b(\mathbf{1}-\boldsymbol{x})$ helyettesítéssel. Az ebből generált sajátértékrendszert felírva és áttranszformálva standard sajátérték-problémára, a következőket kapjuk:

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z} = \lambda\boldsymbol{z}, \tag{46}$$

A korábban tárgyalt Rayleigh-hányadossal kapcsolatos kritériumokat és állításokat alkalmazva a fenti egyenletre, megkaphatjuk a minimalizálás módszerét:

$$\boldsymbol{z}_1 = \arg.\min_{\boldsymbol{z}^T\boldsymbol{z}_0=0}\left(\frac{\boldsymbol{z}^T\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z}}{\boldsymbol{z}^T\boldsymbol{z}}\right), \tag{47}$$

és következetesen:

$$\boldsymbol{y}_1 = \arg.\min_{\boldsymbol{y}^T\mathbf{D}\mathbf{1}=0}\left(\frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}\right). \tag{48}$$

Az eredeti, Normalizált Vágás algoritmushoz hasonlóan, a generált sajátérték rendszer második legkisebb sajátértékéhez tartozó sajátvektor fogja adni a megoldás kulcsát. A későbbiekben azonban látni fogjuk, hogy PdPR-től függően a harmadik legkisebb sajátértékhez tartozó sajátvektor is fontos szerepet kaphat a dinamikus ütemezés során. Minden egyéb megállapítás, amelyet a 4.1.1 fejezet végén tettünk, igaz marad az algoritmus módosítása után is.

4.2.4 Összefoglalás

Az analizáló algoritmus ezen a ponton ér véget, és itt adja át az adatait (az előbb említett sajátvektorokat, valamint a pontsúlyvektort) a dinamikus ütemező számára.

Beláthatjuk, hogy ha a tervezés folyamán bármelyik prioritás változik, legyen szó akár arról hogy teljesül, vagy arról, hogy nem teljesül, szükségtelen lesz újra analizálni egy másfajta szegmentálás céljából. Ezáltal a PdPR-ek nem szólnak bele közvetlenül a rendszerszintű szintézis ezen lépésébe. A későbbiekben azonban az ütemezéssel

egybekötött particionálás során lehetőség lesz arra, hogy érvényre jussanak. Így a 3.1 fejezetben tárgyalt első problémára találtunk hatékony megoldást.

Szintén beláthatjuk, hogy az analizálás bevezetésével új utak nyílhatnak meg olyan dinamikus ütemező algoritmusok fejlesztésére, amelyek rugalmasan tudják kezelni a szegmentálás határait az ütemező algoritmus függvényében. Ezzel a dekompozíciót és az ütemezést összekapcsoló iterációs problémának a súlyát is csökkenthetjük, amelyet a második problémaként jegyeztünk fel a tervezési lépések vizsgálata során. Lényeges, hogy mivel az analízis csak egyszer fut le, ezért az SLS egyes fázisai között az új módszer nem iterál.

A következő fejezetben a dinamikus ütemezővel, és az azt megvalósító Dinamikus Erővezérelt Ütemező algoritmussal fogunk foglalkozunk.

5 A dinamikus ütemező

5.1 Beveztés

A dinamikus ütemező az SLS szintézis negyedik lépése, amely az előző fejezetben bemutatott analizálás fázisa után következik, és az abból nyert adatok alapján végzi el a feladatot reprezentáló folyamatgráf felosztását és ütemezését.

A 'dinamikus' előtag a *dinamikus optimalizálás* általános kifejezéséből adódik, melynek tulajdonságai az eredeti SLS-rendszertervezés fázisaival ellentétben az új módszer lépéseiben megtalálhatók. Hiszen egy rendszer időben változó folyamatainak számos lefutási lehetőségét a dinamikus optimalizálás során analitikus módszerrel, az 4.1.1 fejezetben található, (5) állapotegyenlet felírásával közelítjük meg, miközben minden egyes lehetséges megoldáshoz egy értékelésre alkalmas skaláris mennyiséget rendelünk. Az (5) egyenlet maga az ún. költségfüggvény, amelynek minimalizálása során megkapjuk a megoldásra vonatkozó formulákat. Jelen esetben optimalizálásról nem beszélhetünk a feladat NP-teljes volta miatt, csak az optimálishoz közeli megoldásokról, hiszen senki nem garantálja, hogy csak egy jó megoldása lehet az adott feladatnak. Ettől eltekintve azonban a fenti definíció teljes mértékben illik az új SLS-szintézis módszerére.

A dinamika továbbá annak is betudható, hogy – mint korábban már említettük – ez a tervezési módszer az egyes lépések között nem iterál, ezért az algoritmus nem áll le minden ütemezés után (nem torpan meg), hogy visszalépjen az előző lépésre egy új dekompozíció érdekében, hanem az analízis során kapott adathalmazt használja fel egy másik particionálás meghatározásához, így a PdPR-ek érvényre jutásáig folyamatosan az ütemezés fog futni, és ezáltal válik dinamikussá.

A következő dinamikus ütemező egy már széles körben elterjedt ütemező algoritmusra épít, az Erővezérelt Ütemezőre (Force Directed Scheduler) [35,36]. Annak érdekében azonban, hogy tudja kezelni a PdPR-eket, és ne csak ütemezzen, hanem a partíciókat is meg tudja határozni, módosításokat kell véghezvinni az algoritmus szerkezetében. Az így kapott új algoritmust az eredeti Erővezérelt Ütemező után *Dinamikus Erővezérelt Ütemezőnek* (Dynamic Force Directed Scheduler, DFD Scheduler) neveztük el.

5.2 A Dinamikus Erővezérelt Ütemező (Dynamic Force Directed, DFD Scheduler)

5.2.1 A PdPR figyelembevétele az ütemezés során

A DFD Scheduler egyik legnagyobb különbsége az eredeti Erővezérelt Ütemezőhöz képest, hogy képes bizonyos PdPR-ek kezelésére, és az anlizálásból kapott adatok alapján meghatározza az ütemezés számára legmegfelelőbbnek bizonyuló partíciókat.

A PdPR-ek annyira sokfélék lehetnek, hogy a most bemutatott algoritmus csak a legalapvetőbb követelményeket és prioritásokat tudja kezelni, ezért jelenleg nincsen lehetőség speciális, a dekomopozícióra és az ütemezésre vonatkozó PdPR-ek választására a tervezés első lépésénél. Későbbiekben a különböző ütemező, vagy dinamikus ütemező algoritmusok kisebb módosításaival a PdPR-ek alkalmazása kiterjeszthető más követelményekre vagy prioritásokra is.

A WNCut analizáló algoritmus lefutásával 3 vektort kapunk, amelyekből kettő a generált sajátértékrendszer második és harmadik legkisebb sajátértékeihez tartozó $(\boldsymbol{z}_2 \quad \boldsymbol{z}_3)$ sajátvektorai lesznek, egy pedig a $\boldsymbol{v}$ pontsúlyvektor, amelyet már a folyamatgráf készítése közben meghatároztunk. Ekkor definiálható egy $\mathbb{R}^2$ vektortér, amelyben az $P_i(x, y)$ pontpárokra igaz, hogy:

$$\mathbb{R}^2 = \left\{ P_i(x,y) \middle| x \in \boldsymbol{z}_{2,i} \wedge y \in \boldsymbol{z}_{3,i} \right\}, \tag{49}$$

ahol P_i az a folyamatgráf i-edik pontját jelöli, amelynek értékét a $(\boldsymbol{z}_2 \quad \boldsymbol{z}_3)$ vektorok elemei adják.

Ezáltal olyan számpárokat kapunk, amelyek egy Descartes-féle koordinátarendszerben ábrázolhatók, és információkat tartalmaznak arra vonatkozóan, hogy a folyamatgráf pontjai mennyire szorosan kapcsolódnak egymáshoz. Abban az esetben, ha a gráf két pontját összekötő él értéke a többihez viszonyítva nagy, akkor ez azt jelenti, hogy a két feladat között a rendszer megvalósítása során nagy adatkommunikációt igényelne, ezáltal a koordinátarendszerben a pontok nagyon közel kerülnek egymáshoz. A WNCut alkalmazása során létrejövő klikkek is hasonló tulajdonságokat fognak mutatni, a teljes részgráf pontjai azonban nem csak közel lesznek egymáshoz, hanem egymásra is kerülnek. Ez azt jelenti, hogy ugyanazokat az értékeket veszik fel a koordinátarendszerben, leszámítva a klikk eredeti gráfhoz tartozó pontját, ám az is olyan közel kerül az előbb említett ponthalmazhoz, hogy a particionálás során nem fog problémát okozni.

Rendszertervezéskor általában az egyik legfontosabb PdPR a komponens processzorok számára vonatkozó követelmények szoktak lenni. Ez meghatározza, hogy hány részre osszuk fel a folyamatgráfunkat, ezért a Dinamikus Erővezérelt Ütemezést különböző

szegmentáló módszerek szerint csoportosítjuk. Erre a kérdésre vonatkozóan három különböző particionáló algoritmust különböztetünk meg:

1. Mean-median módszer

A mean-median módszer az imént definiált vektortér pontjainak x értékét veszi figyelembe a számítás során, vagyis P_i pontokat merőlegesen vetítjük a Descartes-féle koordinátarendszer abszcisszájára, és ezeket az $x \in \mathbb{R}$, egydimenziós vektortérben elhelyezkedő pontokat vesszük figyelembe a későbbi számítások során.

Beláthatjuk, hogy a kommunikációs teher, vagyis a vágás értékének a minimalizálása és a feladatok feldolgozóegységek közötti egyenletes feladatelosztása az esetek túlnyomó többségében nem teljesülhet egyidejűleg. A WNCut levezetése során láthattuk, hogy az előbb említett tervezési szempontokat figyelembe vettük a vágás költségfüggvényében, azt azonban már csak később, a particionálás során kell meghatározni, hogy melyik kritérium fontosabb számunkra. Ezt a PdPR-ek megszabásával tehetjük meg, és az algoritmus ez alapján fogja eldönteni, hogy hogyan particionálja a folyamatgráfot az ütemezéshez.

Az abszcisszán sorba rendezett pontok között meg kell határoznunk egy osztópontot, amely feletti értékek esetén P_i pont az A csoportba kerül, ellenkező esetben a B csoportba fog tartozni. A PdPR érvényre jutását maga az osztópont és ennek megfelelő kiválasztása biztosítja. Beláthatjuk, hogy ha a felvett x értékek alapján az adathalmaz mediánját választjuk osztópontnak, akkor egy kényszerített két egyenlő részre osztás történik. Ez alapján a gráf pontjainak csoportosítása A és B csoportra történő osztás esetén a következőképpen definiálható:

$$\begin{matrix} x_i > \{median(X) | \forall x \in X\} & akkor & \mathrm{P}_i \in A \\ & különben & \mathrm{P}_i \in B \end{matrix} . \tag{50}$$

A mediánnal történő vágás esetében ellenőriznünk kell, hogy a klikkek sértetlenek maradtak-e, és nem történt-e belevágás, mivel ez a módszer ugyanis erőltetett megoldás, hiszen ez mindenképpen egyenletes elosztást eredményez. Ugyanakkor, ha belevágott, akkor csak egyetlen ponttal kell jobbra, vagy balra menni (átlagtól függően) az új osztópont meghatározásához, mivel a klikkek pontjaihoz tartozó értékek azonosak lesznek az eredetihez kapcsolódó pontot leszámítva.

Abban az esetben, ha a PdPR prioritását figyelembe véve az adathalmaz átlagát választjuk osztópontnak, akkor a (44)-es egyenlet alapján láthatjuk, hogy a vágás értékének minimalizálására törekszünk. Ez a tervezendő feladat szempontjából azt jelenti, hogy a végrehajtandó feladatok a lehetőségek szerint egyenletesen oszlanak el, a vágás azonban úgy történik, hogy a kommunikációs teher a lehető legkisebb legyen. Ebben az esetben, ha $\forall x \in X$, akkor a gráf pontjainak csoportosítása A és B szegmensekre a következőképpen írható fel:

$$x_i > \frac{1}{|X|}\sum_{\mathrm{i}=1}^{|X|} x_i \quad \text{akkor} \quad \mathrm{P}_i \in A \qquad \text{különben} \quad \mathrm{P}_i \in B \tag{51}$$

A fentiekből egyértelműen következik, hogy ezek az eljárások csak két szegmensbe képesek osztani a folyamatgráfot. Ezek alapján a mean-median módszert a DFD Scheduler csak biparticionálás esetén választja, és csak abban az esetben, ha a PdPR-ek megahatározása során követelményként definiáltuk. Az ilyen tervezői döntésnek több oka is lehet, de a leggyakoribb indíték az, hogy a kivitelezési szempontok két komponens processzorral történő megvalósítást írnak elő.

Ahhoz, hogy három vagy több csoportra osszuk a feladatokat, más particionáló módszereket kell használnunk.

2. K-means módszer

A három vagy több csoportra történő bontás esetére vezettük be a k-means szegmentálási módszert, amellyel viszonylagosan egyszerű k csoportra osztani a folyamatgráf által reprezentált feladatokat.

Vegyük a korábbi $\mathbb{R}^2$ vektortérben meghatározott P_i pontokat, és nevezzük el az origóból az i-edik pontba mutató vektort $\boldsymbol{x}_i$-nek. Ekkor adott egy olyan $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$ vektorokból álló halmaz, ahol mindegyik vektor kétdimenziós, valós vektor, és a k-means csoportosítás hatására az n vektor k darab ($k \leq n$) csoportba rendeződik, $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$, ahol a csoportokon belüli WCSS négyzetösszegeket minimalizáljuk. (WCSS: a távolságfüggvények négyzetösszege, minden csoport pontjától a K középhez viszonyítva.) Matematikailag leírva a k-means célja:

$$\arg.\min_{\boldsymbol{S}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \epsilon S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \tag{52}$$

ahol $\boldsymbol{\mu}_i$ az S_i pontok átlaga.

A k-means szegmentáló algoritmus a következő lépésekkel valósítható meg:

Inicializáljuk az algoritmust, és az eredeti módszer helyett, amely random pontok felvételét javasolja, vegyük fel kezdő középpontoknak az osztópontok által kijelölt, $\boldsymbol{z}_2$ vektor értékeiből számolt k-kvantiliseket. 2 részre történő osztás esetén a mediánt, 3 részre történő osztás esetén a terciliseket, 4 részre osztás esetén a kvartiliseket stb. (ennek a

későbbiek során még szerepe lesz). Az osztópontok segítségével számoljuk ki az egyes szegmensek átlagait, amelyek megadják a kezdeti K közepet, minden egyes klaszterhez.

1) Minden egyes pontot abba a klaszterbe sorolunk, amelynek középpontjához a legközelebb helyezkedik el.
2) Kiszámoljuk az új klaszterek középpontját.
3) Addig ismételjük a 3-as és a 4-es lépést, amíg valamilyen konvergencia kritérium nem teljesül (általában az, hogy az egyes pontok besorolása nem változik).
4) A fentiek alapján láthattuk, hogy a k-means algoritmussal történő particionálás esetén lehetőségünk van kettőnél több szegmens meghatározására, egy másik fontos PdPR érvényre jutása azonban nem következhet be. Ugyan a WNCut (44) minimalizálása alapján mindig törekszünk az egyenletes feladatfelosztásra, jelen algoritmus esetében azonban elveszik az a lehetőség, hogy pl. medián alapján tökéletesen egyenletesen helyezzük el a gráf egyes pontjait az adott csoportokban.

Ugyanakkor lehetőségünk van arra is, hogy a kezdeti K pontok segítségével (amelyek a k-kvantilisek alapján lettek meghatározva), intuitív módszerekkel próbálkozzunk k egyenlő részre részre osztani a folyamatgráfot. Ez a módszer azonban nem adja ki feltétlenül a tervezés szempontjából legelőnyösebb feladatelosztást, mert az adatkommunikáció mértékét nem veszi olyan módon figyelembe, mintha elvégeztük volna a k-means algoritmust.

3. Rekurzív mean-median módszer

A rekurzív mean-median módszer a már korábban megismert mean-median módszeren alapul, és a biparticionálás problémájára ad egy második megoldást. A csoportosítás algoritmusa a következő:

1) A generált sajátértékrendszer eredményéből az 5.2.1 fejezetben leírt mean-median módszer szerint elvégezzük a gráf biparticionálását.
2) A diszkrét eredményt ellenőrizzük, hogy stabil-e, vagyis vágott-e klikk élet. Ha igen, akkor korrigáljuk a vágást a korábban megismert módszerek szerint.
3) Rekurzívan megismételjük az 1–2. pontokat, feltételezve, hogy az egyes gráfok az előzően felosztott gráf A vagy B részgráfjai.

A rekurzív biparticionálás eredménye azonban az lesz, hogy az így megvalósított rendszer csak kettő hatvány számú komponens processzor használatát engedélyezi, ellenben itt is érvényre juthatnak a PdPR prioritásai, és ez a tervező rugalmas rendszertervezését biztosítja.

5.2.2 Az Erővezérelt Ütemező algoritmus kapcsolata a particionáló módszerekkel

Az előbbiekben megismerhettük a PdPR-ek alapján csoportosított particionáló módszereket. Ebben a fejezetben megvizsgáljuk, hogy magáért az ütemezésért felelős eljárás (amely jelen esetben az Erővezérelt Ütemező) a particionálás mely lépései közé ékelődik be, hogy hatékony megoldást adjon a tervezendő rendszerre nézve. Tekintsük át az Erővezérelt Ütemező egyes lépéseit:

1) Minden kapott partícióra nézve kiszámítható az egyes műveletvégző egységek által végrehajtandó feladatok mobilitása az ASAP (As Soon As Possible) és az ALAP (As Late As Possible) elvek szerint.
2) A mobilitásból felírható a feladatok eloszlás diagramja, amelyeket művelet típusonként összegezhetünk a következőképpen:

$$\mathrm{DG}(i) = \sum Prob(OP, i) \tag{53}$$

3) Minden feladatnak van az egyes időintervallumokra vonatkozóan egy ún. 'saját ereje', amely a rugóerő felírásához hasonlítható: $\mathrm{F} = \mathrm{Kx}$. Ez alapján a saját erő a következőképpen számítható:

$$\mathrm{SelfForce(j)} = \sum_{i=t}^{b} [\mathrm{DG}(i) \cdot x(i)] \tag{54}$$

4) A tervezés szempontjából előnyös ütemezés esetében a saját erő negatív lesz.
5) Egy tetszőleges művelet rögzítése után elvégezzük újra az 1–4. lépéseket a maradék műveletekre, majd ezt ismételjük addig, ameddig minden művelet indítási időpontja meg nem lesz határozva.

Az Erővezérelt Ütemező az összes, előző fejezetben említett particionáló módszerek mindegyikébe kicsit másképp, de beilleszthető.

A mean-median módszer esetén minden olyan vágásra lefuttatható az ütemező algoritmus, amelyek az átlag és a medián által meghatározott osztópontok között helyezkednek el. A különböző vágások az ütemezés során az egyes feladatok mobilitását változtatják meg, ám nem szükségszerűen minden esetben, ezért a mobilitás és a saját erők meghatározását nem kell az elejétől kezdve végigszámolni, csak akkor, ha változás történt. Ezáltal az algoritmus is gyorsabb futást biztosít a tervező számára, miközben figyelembe veszi a PdPR-t. Az ütemezésből kapott újraindítási idő, és a processzor kihasználtsága alapján meghatározhatjuk, hogy a figyelembe vett prioritások alapján melyik a tervező számára legmegfelelőbb megoldás.

A k-means esetében az ütemező algoritmust a kialakult klaszterek új középpontjának kiszámolása után célszerű lefuttatni. Ezt azonban csak akkor érdemes az iteráció minden lépésében megtenni, ha két teljesen egyenlő részre szeretnénk osztani a gráfunkat. Ugyanis a k-kvantilisek alkalmazásával – a mean-medián eljáráshoz hasonlóan – szintén egy erőltetett módszerrel osztjuk egyenlő részekre a folyamatgráfot. Ezt a PdPR-ek fogják meghatározni, amelyeket a feladat leírásával együtt kell definiálnia a tervezőnek.

Fontos megemlíteni egy másik PdPR-t is, amely a feladat előírásaitól függően lehet prioritás vagy követelmény, ez pedig az előírt újraindítási idő. Ennek az ütemezés során és pipeline rendszerek tervezése esetén van fontos szerepe, mivel alapvetően meghatározza a rendszer működését. Ezért például real-time rendszerekben követelményként elkerülhetetlen a betartása. Abban az esetben, ha a meghatározott újraindítási idő nem teljesül az ütemezés során, akkor az iteráló particionálás segítségével további lehetőségeket kell megvizsgálni és ezeket ütemezni. Ha ezzel a módszerrel sem találunk a rendszerre megfelelő realizációt, akkor változtatni kell a PdPR prioritásain, és például több komponensprocesszorral kell megvalósítani a rendszert, vagyis többfelé kell osztani a folyamatgráfot.

5.2.3 Összefoglalás

Az előző két fejezetben láthattuk az új rendszerszintű szintézishez módosított dinamikus ütemező algoritmust, a Dinamikus Erővezérelt Ütemezőt (DFD Schedulert).

A legnagyobb különbség a hagyományos SLS szintézis során használt dekompozíciós eljárásokhoz és ütemezőkhöz képest, hogy bár itt is iterációs módszerekkel határozzuk meg az egyes partíciókat, azonban ezt mindig a dinamikus ütemező fogja megtenni az analízis során kapott vektorok segítségével, úgy, hogy a lehető legkevesebbszer kelljen újraszámolnia egy-egy partíció ütemezését. Ezáltal az analizáló algoritmusnak nem kell az egyes ütemezések után újra futnia.

Másik fontos különbség a hagyományos SLS-hez képest, hogy a DFD Scheduler maga választ partícionáló módszert az előre meghatározott PdPR-ek alapján. A hagyományos rendszertervezés során a tervezőnek joga van eldönteni az egyes HLS algoritmusok végén, hogy megfelel-e számára a kapott eredmény, ebben az esetben azonban algoritmus el tudja dönteni, hogy teljesülnek-e az adott PdPR-ek, mind a prioritásokra, mind a követelményekre nézve. Azáltal, hogy el tudja dönteni a PdPR-ek teljesülését, lehetőség nyílik az automatikus javításra is, amely jelen esetben a DFD Scheduler újrafutását jelenti. Ezáltal a folyamatot automatizálni lehet, és ez sok esetben előnyös magasabb bonyolultsági fokú rendszerek tervezése során.

A 3. táblázat egy összefoglalja a DFD Scheduler partícionáló módszereit, és ezek legfőbb jellemzőit:

	Mean-median	K-means	Rekurzív mean-median
JELLEMZŐK			
Biparticionálás	●	●	●
Kettő hatvány számú partíció		●	●
Tetszőleges számú partíció		●	
Újraindítási idő figyelembe vétele	●	●	●
Min. kommunikációra törekvés	●	●	●
Egyenlő feladatelosztás	●		●

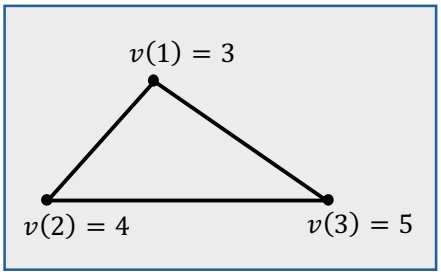
3. táblázat: A DFD Scheduler partícionáló módszerei a PdPR-ek függvényében

6 Példák

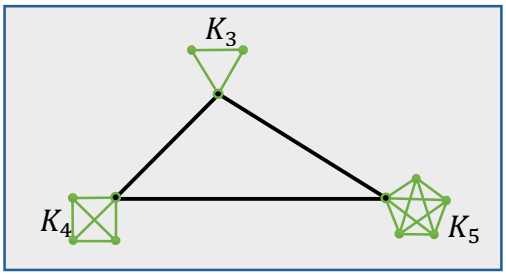
Ebben a fejezetben két példán keresztül szemléltetem az analizáló algoritmus, és a particionáló módszerek működését. Az első példa egy egyszerű gráf, amely három pontból áll, és csak az analizáló módszert és a particionálást végezzük el rajta, bemutatva a WNCut pontos működését és a gráf bővítését. A második példában egy bonyolultabb folyamatgráfból indulunk ki, és az analízis mellett elvégezzük a dinamikus ütemezést is.

6.1 Első példa

Vegyük fel az 6. ábrán látható következő egyszerű G gráfot:



6. ábra: Egy hárompontú gráf



7. ábra: A kibővített gráf

A fenti gráf minden élsúlyának értéke 1, pontjainak a súlya rendre 3, 4, 5. A PdPR-eket úgy határoztuk meg, hogy a feladat alapján a gráfot két részre ossza, figyelembe véve a kommunikációs költséget, amelynek lehetőség szerint a legkisebbnek kell lennie. Mivel ezt az egyszerű példát nem ütemezzük, csak particionáljuk, ezért további PdPR-ek meghatározása nem szükséges.

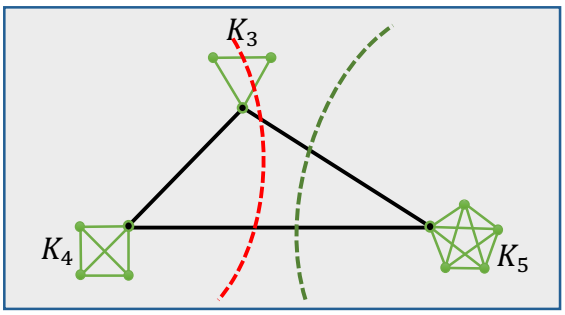
A 7. ábrán jól láthatók az eredeti G gráf egyes pontjaira növesztett K_3, K_4, K_5, zöld színnel jelölt teljes részgráfok (klikkek), amelyek az eredeti $\boldsymbol{v}$ súlyvektorok által meghatározott pontsúlyokat jelképezik. A mátrix bővítése pontonként történik, az egyes pontsúlyoknak és a (43) egyenletnek megfelelően. Ezáltal megkapjuk a Laplace-mátrixot, amely készen áll a sajátértékek és a sajátvektorok meghatározására. A sajátvektorok meghatározása után a következő $\boldsymbol{y}$ vektort kapjuk:

A gráf eredeti pontjai			Az egyes pontokra épített klikkek további pontjai								
$P_{1,1}$	$P_{2,1}$	$P_{3,1}$	$P_{1,2}$	$P_{1,3}$	$P_{1,4}$	$P_{1,5}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	$P_{3,2}$	$P_{3,3}$
-0.318	0.321	0.108	-0.329	-0.329	-0.329	-0.329	0.328	0.328	0.328	0.109	0.109
●	○	○	●	●	●	●	○	○	○	○	○

○ Az A csoportba tartozó pontok ● A B csoportba tartozó pontok

4. táblázat: A kibővített gráf particionálása

A táblázat felső sorában az egyes pontok jelölése, a középső sorban a hozzájuk tartozó értékek találhatók. A fejlécben láthatóan jól elkülönülnek az eredeti pontok, és az azokra épített klikkek további pontjai. A PdPR alapján a mean-medián módszer választandó, mivel az biparticionálást ír elő a gráf számára. Azon belül is a PdPR azt határozta meg, hogy a vágás minimalizálására ügyeljünk (még akkor is, ha ebben a feladatban ennek a prioritásnak nincsen jelentősége).



8. ábra: A dekompozíció eredménye

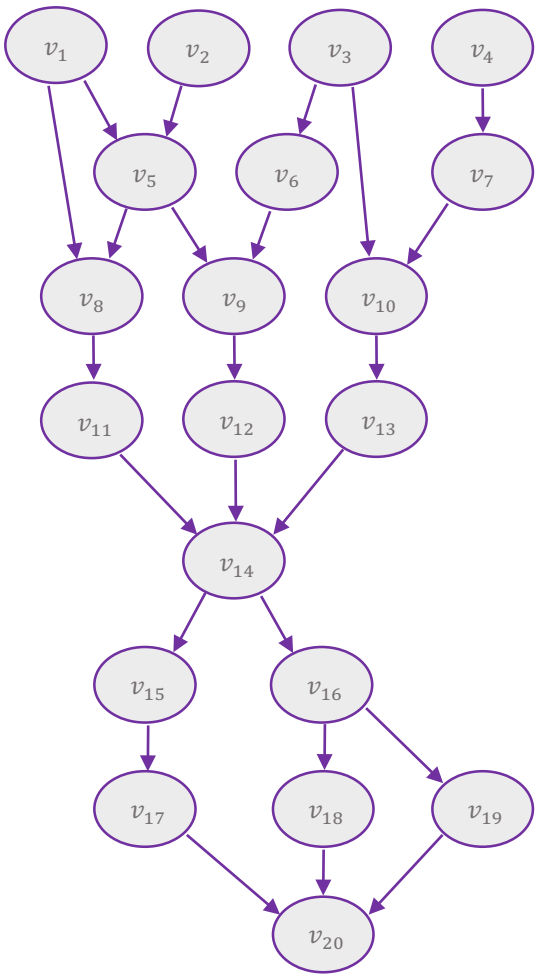
Ezek alapján a 8-as ábrán pirossal jelölt szaggatott vonal az eredeti dekompozíciós algoritmus által meghatározott vágás, amely belevág a klikkbe, a zöld vágás pedig a WNCut algoritmus eredményét mutatja, amely figyelembe veszi az egyes pontok súlyát.

6.2 Második példa

Vegyük fel a 9. ábrán található gráfot.

Az ábrán minden csomópont súlyát konstans 1-nek tekintjük, és első esetben a pontokat összekötő élsúlyokat szintén 1-nek definiáljuk. PdPR követelményként előírjuk, hogy három részre akarjuk osztani a gráfot, valamint a kommunikáció minimalizálásának szeretnénk magasabb prioritást adni.

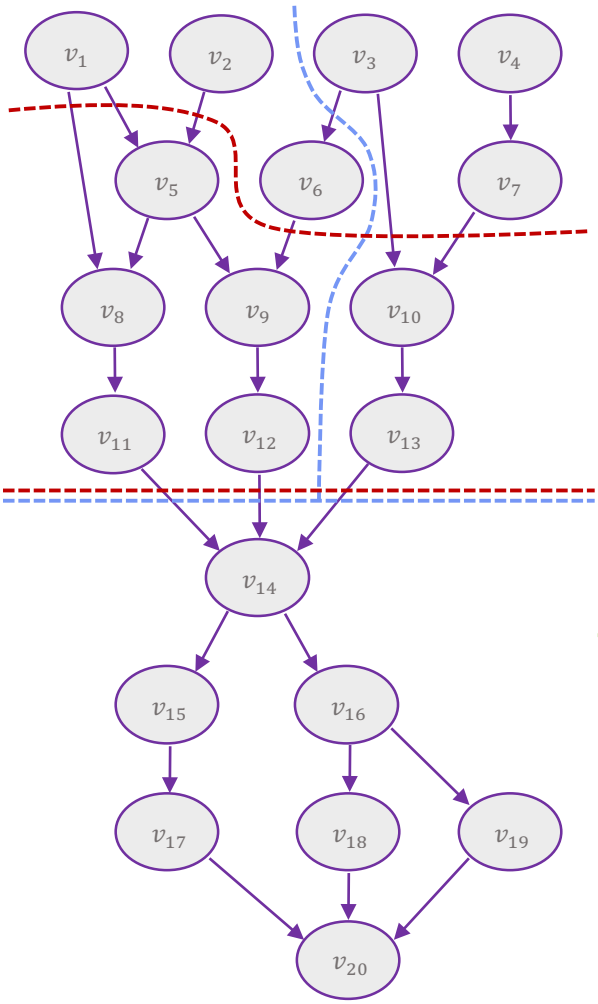
Az imént meghatározott PdPR-ek, és a gráf alapján először két különböző rendszerszintű szintézist végzünk a jobb oldalon látható folyamaton. A hagyományos SLS-t egy már korábban többször használt, ACL [38] nevű dekompozíciós eljárással, és Force Directed ütemezéssel valósítjuk meg, míg az új SLS-módszerhez a 4. és az 5. fejezetben bemutatott WNCut algoritmust és a DFD Schedulert fogjuk használni.



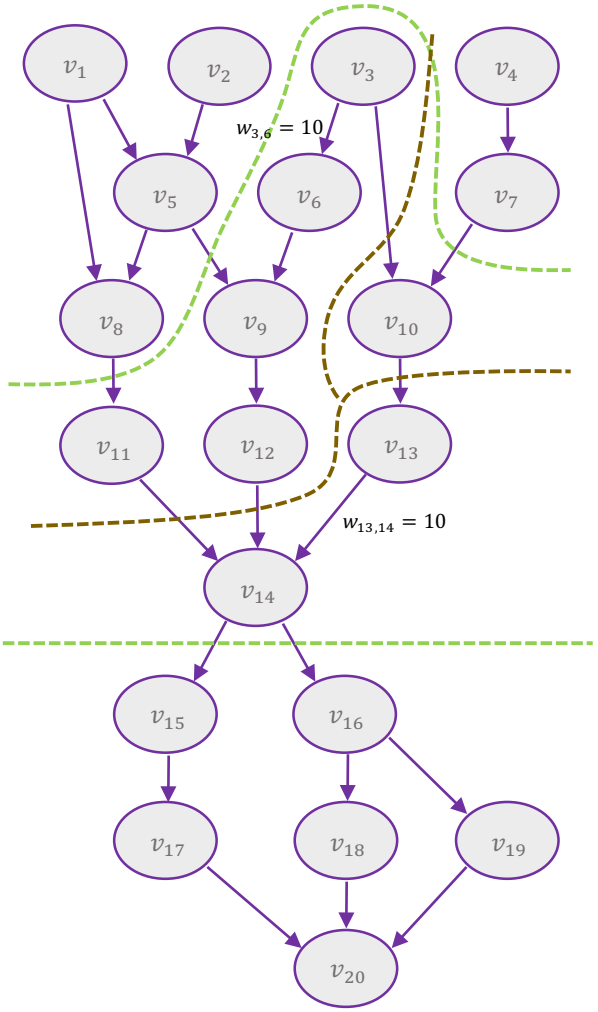
9. ábra: 20 pontos példagráf

Vizsgáljuk meg a két módszer által kapott partíciókat, valamint az ütemezés által meghatározott újraindítási időt és késleltetést.

A 10. ábrán láthatók a folyamatgráf felosztásai; a piros szaggatott vonal az ACL vágását jelöli, a kék pedig a WNCut és DFD Scheduler párosa által meghatározott partíciókat



10. ábra: ACL vs. WNCut+DFD Scheduler



11. ábra: Minimális vágás vs. egyenletes elosztás PdPR-ek

mutatja be. Az 5. táblázat első két oszlopa összefoglalást tartalmaz a Force Directed algoritmust és a DFD Schedulert megvalósító PIPE [27] nevű program által számolt ütemezések eredményeiből. Láthatjuk, hogy a hagyományos módszer 9-es újraindítási idő mellett 33-as késleltetést ad, miközben egy időben két buszt használ a kommunikáció kielégítésére. Az új módszer esetében az újraindítási idő 9, a késleltetés 26, és mindemellett csupán egy kommunikációs buszra van szükség. A két módszert összevetve kijelenthetjük, hogy a WNCut, és a DFD Scheduler algoritmusok együttes munkája jobb eredményre vezet, mint az ACL és a Force Directed algoritmus.

A következőkben változtassuk meg két él súlyának értékét 1-ről 10-re, hogy belássuk, az analizáló algoritmus képes megkülönböztetni az élek súlyát, és ennek függvényében fogja keresni a minimális vágást. Ahhoz, hogy ezt a legjobban be tudjuk mutatni, válasszuk a $v_6 - v_3$ és a $v_{13} - v_{14}$ pontokat összekötő éleket 10-es súlyúnak, hiszen a korábbi vágásban ezeken a helyeken vágta el az algoritmus a gráfot, ha azonban figyelembe veszi az élek súlyának változását, akkor az új partícióknak biztosan más képet kell adniuk.

Ilyen változtatásokat figyelembe véve végezzük el az analizálást és a dinamikus ütemezést. A 11. ábrán látható vágások alapján kétféle particionálást végeztünk el. Az első esetben PdPR-nek a minimális kommunikációt állítottuk be, melyet barna szaggatott vonallal jelöltünk második esetben pedig a feladatok egyenletes elosztására törekedtünk, melyet zöld szaggatott vonal jelez. A többletkommunikáció nagyobb újraindítási időt okoz, ugyanakkor a késleltetésből származó lappangási idő nem növekszik túl nagy mértékben. A százalékban kifejezett átlagos processzorhasználat a nagyobb élsúlyok miatt összességében csökken, de nem számottevően.

A kétféle prioritási kritérium által eredményezett buszhasználatot összehasonlítva láthatjuk, hogy a minimális kommunikáció esetében egyidejűleg kevesebb buszra van szükség, mint a feladatok egyenletes elosztása esetében (11. ábra utolsó két oszlopa). Ugyanakkor az egyenletes feladatelosztás esetében az átlagos processzorhasznált jobbnak bizonyul, mint a minimális kommunikáció esetében, hiszen az egyenletes elosztás arra törekszik, hogy az egyes komponens processzorok kihasználtsága minél jobb legyen.

	10. ábra		11. ábra	
	ACL	WNCut+DFD Scheduler	Minimális kommunikáció	Egyenletes elosztás
JELLEMZŐK				
Újraindítás	9	9	13	10
Késleltetés	33	26	27	28
Buszhasználat	2	1	1	2
Egyidejűleg használt processzorok száma	333323212	233333221	2222222221111	3333322221
Átlagos processzorhasználat	81,4%	81,4%	73,3%	80%

5. táblázat: Különböző particiók alapján ütemezett vágások

Összefoglalás és további fejlesztési lehetőségek

A dolgozat során megismerhettük a heterogén multiprocesszoros (HMS) rendszerek felépítését, valamint tervezési módszerüket: a rendszerszintű szintézist (SLS). A hagyományos rendszertervezés problémáit szem előtt tartva kifejlesztettünk egy új eljárást a HMS rendszerek tervezésére, amely figyelembe veszi a tervező által előre meghatározott követelményeket és prioritásokat (PdPR). A dekompozíciót és az ütemezést olyan új algoritmusokkal váltottuk ki, amelyek figyelembe veszik a prioritások okozta egymástól való függésüket, ezáltal nagy mértékben csökkentik az iterációk számát a szintézis során.

A dekompozíciót kiváltó analizáló WNCut algoritmus nem függ a PdPR-ektől, csak és kizárólag a tervezendő feladatot reprezentáló folyamatgráftól, ugyanakkor nem határozza meg a partíciókat, hanem egy speciális spektrumot szolgáltat a gráfról. Ezt az adatot a dinamikus ütemező algoritmus kapja meg, a DFD Scheduler, amely egyben a partíciók meghatározásáért is felelős, ezáltal az ütemezést a particionálással egyidőben, egymást figyelembe véve tudják meghatározni. Ezáltal a tervező sokkal hatékonyabb heterogén többprocesszoros (HMS) rendszert kaphat, mintha a hagyományos SLS-algoritmusokat használná.

Az új SLS-módszer számos fejlesztési lehetőséget kínál, amelyek a PdPR-ek számának egyszerű növelését biztosítják a tervező számára. A további PdPR-ek figyelembe vétele az egyes particionáló és ütemező módszerek apróbb módosításaival lehetséges, amelyek még kifinomultabbá tennék a tervező algoritmusokat. Érdemes lenne olyan dinamikus ütemező módszert fejleszteni, mely tovább csökkenti az iterációk számát a tervezés folyamata során.

A dolgozatban leírt új tervezési algoritmusok olyan eljárást adnak a tervezők kezébe, melyek a jelenlegi módszerekhez képest sokkal könnyebben kezelhetők, rugalmasabbak, és hatékonyabb működést eredményeznek.

Hivatkozások

[1] Jianbo Shi és Jitendra Malik, *„Normalized Cuts and Image Segmentation”* IEEE Transactions on pattern analysis and machine intelligence, vol. 22, No. 8, 2000

[2] Rácz György, Markovits Tibor Gergely, Pilászy György és Arató Péter, *„Handling Data Dependent Execution Times of Software Loops in the High Level Design of Real Time Systems”* ISBN:978-963-396-049-3

[3] S. Dasgupta, C.H. Papadimitriou, és U.V. Vazirani, *„NP-Complete problems”*

[4] A. Pothen, H.D. Simon, és K.P. Liou, *„Partitioning Sparse Matrices with Eigenvectors of Graphs,o SIAM J. Matrix Analytical Applications”,* vol. 11, pp. 430-452, 1990.

[5] Antal Péter, Antos András, Horváth Gábor, Hullám Gábor, Kocsis Imre, Marx Péter, Millinghoffer András, Pataricza András, Salánki Ágnes *„Intelligens adatelemzés”* ISBN 978-963-279-171-5

[6] Péter Arató, Dániel András Drexler and György Rácz *„Analyzing the effect of decomposition algorithms on the heterogeneous multiprocessing architectures in system level synthesis”* , ISSN 1224-600X

[7] Hendrickson, B; Leland, R., *"An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations"*,. SIAM J. Sci. Stat. Comput., 16(2), pp. 452,469, 1995.

[8] Hendrickson, B.; Leland, R., *"A Multilevel Algorithm for Partitioning Graphs"*,. In Proc. Supercomputing '95. (Formerly, Technical Report SAND93-1301, 1993).

[9] Karypis, G.; Kumar, V., *"Metis-unstructured graph partitioning and sparse matrix ordering system"*, version 2.0”, 1995.

[10] Trifunovic, A.; Knottenbelt, W. J., *"Parkway 2.0: a parallel multilevel hypergraph partitioning tool"*, in Computer and Information Sciences-ISCIS 2004, Springer, 2004, pp. 789,800.

[11] Goraczko, M.; Liu, J.; Lymberopoulos, D., *"Energy-Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems"*, DAC 2008, June 8–13, 2008, Anaheim, California, USA

[12] Gergely Suba, Péter Arató (2015) Concept of the system-level synthesis framework PipeComp. In *WAIT 2015 : Workshop on the Advances of Information Technology*. Budapest.

[13] Péter Arató, Dániel András Drexler, Gábor Kocza (may 2014) A method for avoiding loops while decomposing the task description graph in system-level synthesis. In *2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. pp. 231–235. IEEE.

[14] Péter Arató, Gergely Suba (may 2014) A data flow graph generation method starting from C description by handling loop nest hierarchy. In *IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. pp. 269–274. IEEE.

[15] Gergely Suba, Péter Arató (2013) A new method for transforming algorithm into VHDL by starting from a Haskell functional language description. In *Middle-European Conference on Applied Theoretical Computer Science*.

[16] Péter Arató, Gábor Kocza, István Loványi, László Vajta (feb. 2008) Hardware-software Codesing of Feature Tracking Algorithms. In *Intelligent Engineering Systems, 2008. INES 2008. International Conference on*. pp. 41 -45.

[17] S. Gupta, S. Katkoori Force-directed scheduling for dynamic power optimization, Print ISBN: 0-7695-1486-3, Publisher: IEEE, *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*

[18] P.G. Paulin, J.P. Knight, Force-directed scheduling for the behavioral synthesis of ASICs, Published in: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (Volume: 8, Issue: 6, Jun 1989)

[19] Gerstlauer, A.; Haubelt, C.; Pimentel, A.D.; Stefanov, T.P.; Gajski, D.D.; Teich, J., "Electronic System-Level Synthesis Methodologies", Computer-Aided Design of Integrated Circuits and Systems, *IEEE Transactions on , vol.28, no.10, pp.1517,1530*, Oct. 2009

[20] Corvino, R.; Gamatié, A.; Geilen, M. & Józwiak, L. (2012), "Design space exploration in application-specific hardware synthesis for multiple communicating nested loops", *in 'ICSAMOS' , IEEE, , pp. 128-135* .

[21] Carminati, A.; de Oliveira, R.S.; Friedrich, L.F., "Exploring the design space of multiprocessor synchronization protocols for real-time systems", *Journal of Systems Architecture, Volume 60, Issue 3*, March 2014, pp.258,270, ISSN 1383-7621

[22] Cilardo, A.; Gallo, L.; Mazzocca, N., "Design space exploration for high-level synthesis of multi-threaded applications", *Journal of Systems Architecture, Volume 59, Issue 10, Part D*, November 2013, pp. 1171,1183, ISSN 1383-7621

[23] Arató, P.; Mann, Z. A.; Orbán, A., "Finding optimal hardware/software partitions", *Formal Methods in System Design, Volume 31*, Issue 3, pp. 241,263. 2007

[24] Wu Jigang; Thambipillai, S; Tao Jiao, "Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design", *Design Automation for Embedded Systems, Volume 12, Issue 4, pp. 345,375*, 2008.

[25] Ernst, R.; Henkel, J.; Benner, T., "Hardware-software cosynthesis for microcontrollers," *Design & Test of Computers, IEEE , vol.10, no.4, pp.64,75*, Dec. 1993

[26] Pradeep K. M.; Somnath, P.; Venkataraman, S.; Gupta, R., "Hardware/Software Co-design of a High-end Mixed Signal Microcontroller" *In Proceedings of the Eleventh International Conference on VLSI Design: VLSI for Signal Processing (VLSID '98)*. IEEE Computer Society, Washington, DC, USA, 91-.

[27] Arató, P.; Visegrády, T.; Jankovits, I., "High Level Synthesis of Pipelined Datapaths", John Wiley & Sons, New York, ISBN: 0 471495582 4, 2001

[28] Meeus, W.; Van Beeck, K.; Goedemé, T.; Meel, J.; Stroobandt, D., "An overview of today's high-level synthesis tools", *Design Automation for Embedded Systems, volume 16, Issue 4, pp. 31,51*, 2012

[29] Chatha, K.S.; Vemuri, R., "A tool for partitioning and pipelined scheduling of hardware-software systems", System Synthesis, 1998. Proceedings. 11th International Symposium on , pp.145,151, 2-4 Dec 1998

[30] Arató, P.; Mann, Z. A.; Orbán, A, "Time-constrained scheduling of large pipelined datapaths", *Journal of Systems Architecture, Volume 51, Issue 12, pp. 665,687*, 2005.

[31] Grajcar, M., "Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System", *In Proceedings of the 36th ACM/IEEE Conference on Design Automation* (1999).

[32] Ding, J-H.; Chang, Y-T.; Guo, Z-D.; Li, K-C.; Chung, Y-C., "An efficient and comprehensive scheduler on Asymmetric Multicore Architecture systems", *Journal of Systems Architecture, Volume 60, Issue 3, March 2014, pp. 305,314*, ISSN 1383-7621

[33] Lin, C-S.; Lin, C-S; Lin, Y-S.; Hsiung, P-A.; Shih, C., "Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems", *Journal of Systems Architecture, Volume 59, Issue 10, Part C, November 2013, pp. 1083,1094*, ISSN 1383-7621

[34] Pilászy, Gy.; Rácz, Gy.; Arató, P., "The effect of increasing the latency time in High Level Synthesis", 2013, *Periodica Polytechnica*

[35] O'Neal, K.; Grissom, D.; Brisk, P., "Force-Directed List Scheduling for Digital Microfluidic Biochips", VLSI and System-on-Chip (VLSI-SoC), 2012 *IEEE/IFIP 20th International Conference on , pp. 6,7-10* Oct. 2012

[36] Paulin, P.G.; Knight, J.P., "Force-Directed Scheduling in Automatic Data Path Synthesis", *Design Automation, 1987. 24th Conference on , , pp.195,202*, 28-1 June 1987

[37] Hendrickson, B.; Leland, R., "The Chaco User's Guide Version 2.0", *Technical Report SAND94--2692*, 1994.

[38] Arató Péter, Drexler Dániel András, Rácz György Analyzing the Effect of Decomposition Algorithms on the Heterogeneous Multiprocessing Architectures in System Level Synthesis *SCIENTIFIC BULLETIN OF POLITECHNICA UNIVERSITY OF TIMISOARA TRANSACTIONS ON AUTOMATIC CONTROL AND COMPUTER SCIENCE* 60:(74) pp. 39-46. (2015)

[39] Z. Wu and R. Leahy, aAn Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmenta- tion,o *IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 15, no. 11, pp. 1,101-1,113*, Nov. 1993.