Budapesti Műszaki és Gazdaságtudományi Egyetem

Közlekedésmérnöki Kar

Közlekedésautomatikai Tanszék

Busz előnyben részesítési algoritmus fejlesztése ACTROS VTC 3000 forgalomirányító berendezéshez

Szakdolgozat

Polgár János

BW3QUY

2009. szeptember-november

Absztrakt

Napjainkban a városi közlekedés helyzete egyre kedvezőtlenebbé válik. Az életkörülmények javítása érdekében szükségszerű a közforgalmú közlekedés modal splitben elfoglalt helyének stabilizálása, ill. részesedésének növelése az egyéni közlekedés rovására. Jelenleg nem áll rendelkezésre megfelelő mennyiségű pénzügyi forrás a közlekedés szerkezet teljes átalakítására, ezért szükséges „pótmegoldásokat” keresni. Az egyik ilyen lehetőség az autóbuszok csomóponti előnybiztosítása, melynek megvalósításához viszonylag kevés új hardverelemet kell üzembe állítani. Szakdolgozatomban elkészítettem egy ilyen autóbuszok közlekedésétől függő programot a Signalbau Huber GmbH vállalat ACTROS VTC 3000 forgalomirányító berendezésére.

Kulcsszavak

Csomóponti előnybiztosítás, társadalmi költség, előnybiztosítás megvalósítása

Konzulens

Egyetemi konzulens: Tettamanti Tamás (PhD hallgató; BME Közlekedésautomatikai Tanszék 1111 Budapest, Bertalan Lajos utca 2. Z ép. 506)
Vállalati konzulens: Nagy Tibor (szoftverfejlesztő; SWARCO Traffic Hungaria KFT 1103 Budapest, Gyömrői út 150.)

Köszönetnyilvánítás

Ezúton szeretném megköszönni Tettamanti Tamásnak a program kialakításában és tesztelésében, Dr. Varga Istvánnak a program stratégiájának kidolgozásában és Nagy Tibornak a Java fejlesztőkörnyezet és az ACTROS berendezés működésének megismerésében nyújtott pótolhatatlan segítségét.

Tartalomjegyzék

Ábrajegyzék

Táblajegyzék

1. Bevezetés

A városi közlekedésben a közforgalmú közlekedés előtérbe helyezése általános törekvés, ami összefügg a fenntartható fejlődés megvalósításával. A törekvés egyik oka az, hogy a közúti közlekedés emissziója jelentős, melynek hatásait minden városlakó nap mint nap tapasztalja. A kibocsátott káros anyagok mennyisége csökkenthető az egyéni közlekedés visszaszorításával, és a közforgalmú közlekedés előnyben részesítésével. Több módszer áll rendelkezésre a priorizálására, például sávlezárások, behajtási korlátozások bevezetése - azaz direkt közlekedéspolitikai eszközök használata; az egyéni közlekedés költségeinek növelése adóemeléssel; ill. a közforgalmú közlekedési kínálat minőségének javítása. A minőség emelése csak akkor sikeres, ha a rendelkezésre álló fizikai eszközöket (pl. járművek, csomópontok, megállóhelyek) és az ezeket rendszerbe kapcsoló logikai eszközöket (menetrendek, szabályok, intézkedések) együtt és egymással összhangban fejlesztik. Ennek a fejlesztésnek koncepcionálisnak kell lennie.

A közforgalmú közúti közlekedés bizonyos utazási teljesítmény felett helyettesíthető kötött pályás eszközökkel. Ezen közlekedési eszközök előnyei vitathatatlanok, ugyanakkor üzembe állításuk jelentős beruházást igényel mind a pálya, mind a jármű oldaláról. Az autóbuszok a többi alágazathoz képest olcsón beszerezhetők, viszont nem tudnak jelentős menetidő-csökkenést nyújtani az egyéni közlekedéssel szemben. Ezen probléma egyik részleges feloldása lehet az autóbuszok csomóponti előnybiztosítása. Jelentős menetidő-csökkenést ez a módszer sem tud biztosítani, viszont igen sok előnye van, többek között csökkenti az autóbuszon utazók késésből adódó veszteségeit, valamint az autóbuszok emisszióját.

A dolgozat fő célja egy, az autóbuszok közlekedését figyelembe vevő, és azokat előnyben részesítő program kidolgozása ACTROS VTC 3000 típusú forgalomirányító berendezéshez. A megvalósításhoz ki kellett dolgozni egy előnyadási stratégiát, melyet több autóbusz egyidejű érkezésekor használ a berendezés. Az algoritmus által kijelölt autóbusz ezután a jelzésterv aktuális állapotának megfelelően zöld nyújtás, zöld előrehozás, vagy fázisbeillesztés segítségével előnyt kap, és a csomópontot azonnal elhagyhatja.

A dolgozatban először áttekintem a jelzőlámpás irányítás fajtáit, több szempontból csoportosítom azokat. Bemutatom, milyen passzív és aktív előnybiztosítási intézkedések állnak rendelkezésre ahhoz, hogy a hálózaton közlekedő autóbuszok minél hamarabb elérhessék úticéljukat. Ábra felhasználásával vázolom, milyen fizikai konfiguráció szükséges a csomóponti előnybiztosítás végrehajtásához, bemutatom a rendelkezésemre állt forgalomirányító berendezést és annak működését. Ezután következik a forgalomfüggő jelzésterv részletes leírása a struktúrán keresztül. A program kódja megtalálható a 3. mellékletben. Kitekintést teszek az alkalmazás fejlesztését illetően. A program futta-

tásából nyert szimulációs eredmények olvashatók a következő fejezetben. Az intézkedésekhez kapcsolódó költségcsökkenés demonstrálására – TDK dolgozat keretében – kidolgoztam egy jelen dolgozatban is szereplő számítási módszert. Az összefoglalóban vázolom a csomóponti előnybiztosítás alkalmazásának korlátait.

2. A jelzőlámpás forgalomirányítás fajtái [1]

A városi közlekedésben több szempont szerint csoportosíthatók a jelzőlámpával irányított csomópontok. Az első az irányítás forgalomtól való függése, a második az irányítás térbeli kiterjedtsége, a harmadik az irányítási architektúra szerkezete.

2.1. Forgalomfüggés szerinti csoportosítás

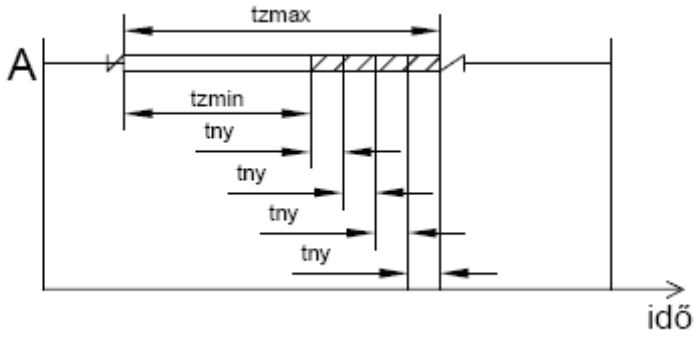
Ezen felosztás szerint léteznek fix programos, programválasztó és programalkotó rendszerek. Utóbbi kettő nevezhető forgalomfüggő irányításnak.

A **fix programmal** irányított csomópontokban a jelzésterv minden része előre definiálva van, a forgalomirányító berendezés nem változtathat ezen, csak lépésről lépésre végrehajthatja. Rögzített a ciklusidő és a fázissorrend is. A fázisterveket általában előzetes forgalomfelvételek alapján készítik el. Ezekben a csomópontokban csak passzív előnybiztosítási intézkedéseket (lásd 3.1. fejezet) lehet tenni.

A **programválasztó** berendezések képesek a beérkező forgalmi adatok alapján több előre definiált fázistervből az adott forgalmi szituációhoz legjobban illeszkedő programot kiválasztani. Az input adatok általában induktív hurokdetektorokról érkeznek. Magyarországon hiába működik több helyen ilyen berendezés, a hurokdetektorok gyakori meghibásodása (szakadása) miatt úgy működnek, mintha fix programot tápláltak volna beléjük. Ilyen rendszerek esetén alkalmazható ún. igényfázis is. (3.2.4. fejezet)

A **programalkotó** rendszerek két további csoportra oszthatók: az első csoportba tartozók a **meglévő jelzésterveket módosítják**, a második csoportba tartozók pedig a beérkező adatok alapján ciklusról ciklusra alkotják meg a fázistervet. Utóbbiakat **adaptív** forgalomirányító rendszereknek nevezzük.

A jelzésterv módosítása legtöbbször a zöldidő nyújtásában nyilvánul meg (1. ábra). Négy előre definiált adatra van szükség a megvalósításhoz: a minimális zöldidőre egy adott fázisban (t_{zmin}), a zöldidőnyújtás idejére (t_{ny}), a maximális zöldidőre (t_{zmax}) és a két követő jármű közötti időre ($t_{köv}$). Működésekor a forgalomirányító berendezés a minimális zöldidőt mindenképpen kiadja, ezután áthaladásmérő detektorral szerez információt arról, hogy van-e még érkező jármű. Ha a $t_{köv}$ idő letelte után nem jön jármű, akkor a fázis nem kap több zöldet. Ha $t_{köv}$ letelte előtt érkezik jármű, akkor a berendezés t_{ny} idővel meghosszabbítja a fázis zöldidejét. Ezt mindaddig megteszi, amíg van $t_{köv}$ időn belül érkező jármű. A nyújtásnak határt szab a $_{tzmax}$, mely elérésekor mindenképpen a sárga jelzés következik. A vázolt folyamatot az 1. ábra szemlélteti. Létezik olyan megoldás is, ahol a t_{zmax}-hoz közeledve csökken a zöld megnyújtásának ideje. Utóbbi megoldással némileg növelhető a csomópont kapacitása.



1. ábra Zöldidőnyújtás ([2] alapján)

A jelzésterv szintén gyakori módosítása egy fázis kihagyásával történik. Ennek megvalósításához természetesen a ritkán használt forgalmi sávokba hurokdetektorokat kell telepíteni.

Programalkotó berendezések már aktív előnybiztosítási intézkedések meghozatalára is képesek. (lásd 3.2. fejezet)

Az adaptív működésű rendszerek minden egyes ciklusban újraalkotják a csomópont jelzéstervét. Ehhez felhasználják a csomóponti ágakból érkező aktuális forgalmi adatokat, előre megadott forgalmi szituációk jellemzőit, korábbi trendeket. Az ilyen rendszerek legnagyobb előnye az, hogy a valós idejű szituációkhoz határoznak meg egy optimumot. Optimalizálható például a sorban álló járművek száma, az utazási sebesség, stb. Ilyen például a Siemens MOTION rendszere, amelyet Budapesten is terveztek működtetni, (megfelelő minőségű mérőrendszer hiányában meghiúsult a próbaüzem). A rendszer jelenleg is megtalálható a Budapesti Forgalomirányító Központban, de nem használják. A legújabb kutatásokban olyan irányítási stratégiát kívánnak kidolgozni, mely a közlekedési jellemzők javítása mellett egyidejűleg a károsanyag-kibocsátást is minimalizálni próbálja.

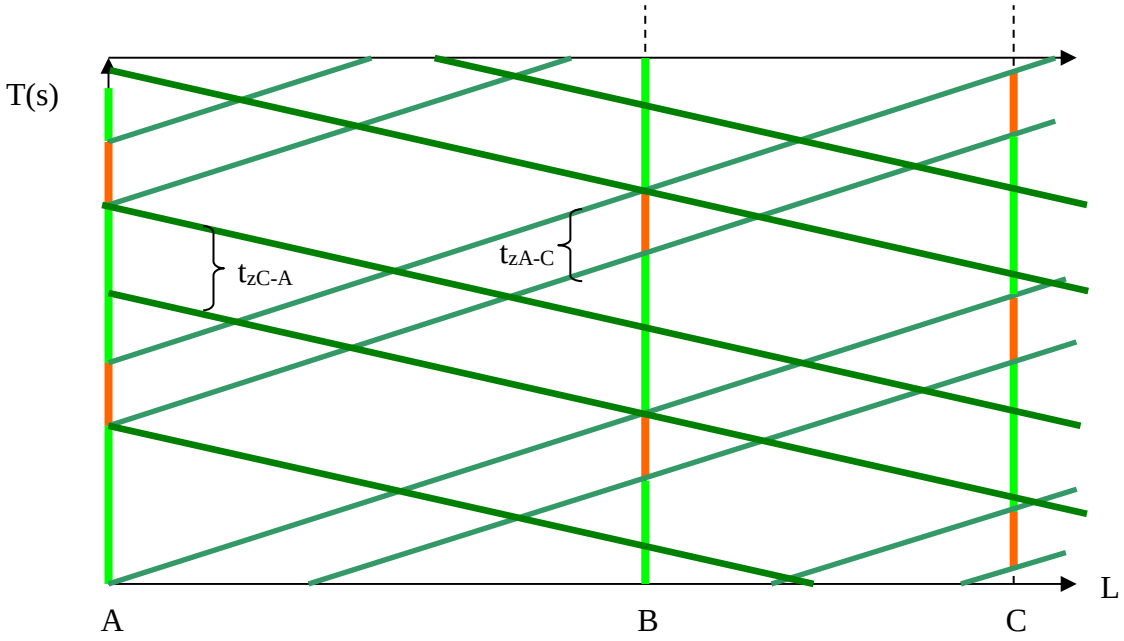
2.2. Térbeli kiterjedtség szerinti csoportosítás

Térbeli kiterjedés szempontjából megkülönböztetünk csomóponti, vonali és hálózati irányítást.

A **csomóponti irányítás** alkalmazásakor az irányított kereszteződés „izolált” csomópontnak tekinthető, ugyanis irányítása nem függ a környezetétől. Ebben az esetben nem beszélhetünk semmilyen összehangoltságról a csomópontok között.

Vonali irányítás: ebben az esetben már több egymás után elhelyezkedő csomópont irányítása együtt történik valamilyen irányítási módszerrel. Leggyakoribb megjelenési formája a zöldhullám. A zöldhullám készítésekor meghatároznak egy összehangolási

sebességet. Mivel a csomópontok nem egyenlő távolságra helyezkednek el, ezért kétirányú összehangolás esetén a két összehangolási sebesség nem azonos. Fix programos irányítás esetén minden összehangolt csomópontban azonos ciklusidőt kell alkalmazni. Egy fix programos három csomópontban alkalmazott összehangolás út-idő diagramját mutatja a 2. ábra. Forgalomfüggő irányítás esetén az összehangolás irányára merőleges csomóponti ágak zöldidejét lehet rugalmasan kezelni. Adaptív rendszerekbe is beépíthető ilyen struktúra.



2. ábra Vonali összehangolás út-idő diagramja

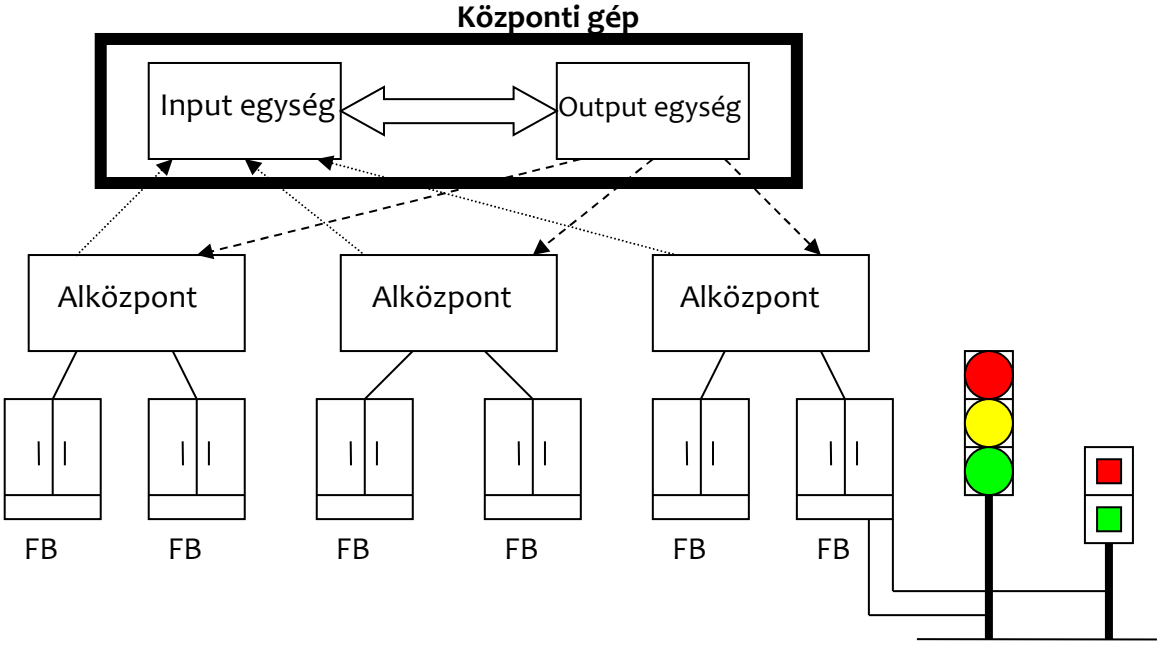
A legkiterjedtebb a **hálózati irányítás**. Ebben az esetben egy lehatárolt terület összes forgalomirányító berendezése valahogyan össze van hangolva. Ezt az összehangolást fix programmal nem érdemes megvalósítani. Egy hálózati szintű optimalizálás nagymértékben megnöveli a számítási igényt, azonban mégis ezt a módszert célszerű alkalmazni. Szignifikáns javulás olyan hálózaton érhető el, melyben nincsen sok kitüntetett vagy kiugróan nagy forgalmú útvonal. A már említett MOTION rendszer hálózati szintű irányítást tesz lehetővé. Jelenleg is folynak kutatások ilyen rendszerek továbbfejlesztésére, tökéletesítésére.

2.3. Architektúra szerinti csoportosítás

Centralizált, decentralizált és vegyes struktúrájú irányítást használnak.

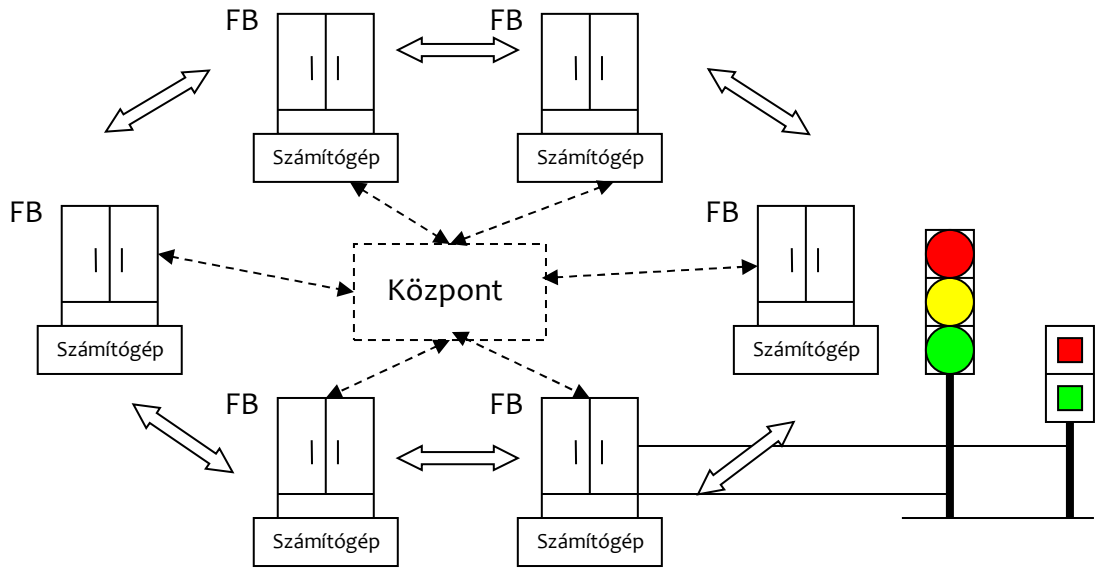
A történelemben elsőként **centralizált hálózatokat** építettek ki. A jelzőlámpával irányított csomópontok számának növekedésével jelentősen nőtt a bejövő adatmennyiség, ezért alközpontokat is létre kellett hozni. Forgalomfüggő irányítás esetén az összes in-

put adat egy központi számítógépbe kerül. Ez a nagy teljesítményű gép számolja ki a szükséges jellemzőket, majd az eredményt visszaküldi a helyszínen lévő berendezésekbe, amelyek csak egy „szükségprogramot” tartalmaznak központi vagy kommunikációs hiba esetére. A struktúra előnye az, hogy a helyszíni berendezéseknek nem kell bonyolultnak lenniük. Hátránya, hogy egy drága központi gépet kell alkalmazni, valamint a kétirányú adatkapcsolatnak is mindig működnie kell. A hálózati struktúra a 3. ábrán tanulmányozható.



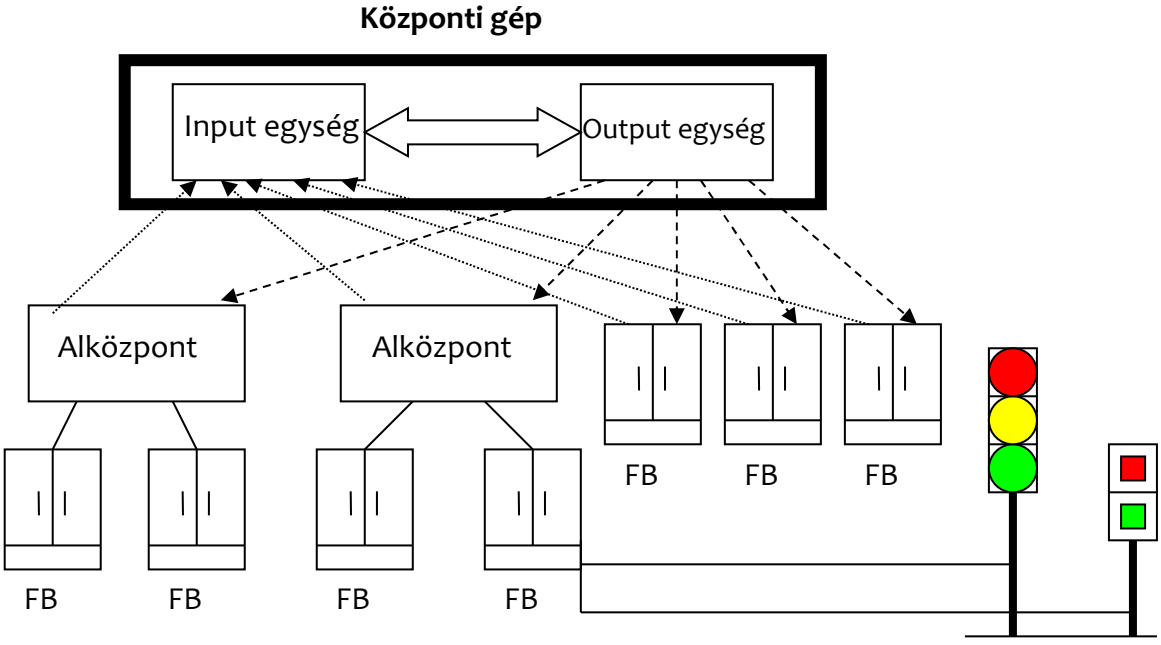
3. ábra Centralizált forgalomirányítási struktúra)

Decentralizált hálózatokban (4. ábra) a számítási funkció el van osztva a forgalomirányító berendezések között, ezért a központi gép elhagyható. Forgalomirányító központ ennek ellenére szükséges, mert a műszaki felügyeletet itt is meg kell oldani. Ez a megoldás kiküszöböli a centralizált rendszer hátrányait, azonban ilyen rendszerben egy forgalomirányító berendezés meghibásodása esetén annak funkcióit egy másiknak át kell vennie, hogy a rendszer működése biztosítva legyen.



4. ábra Decentralizált forgalomirányítási struktúra

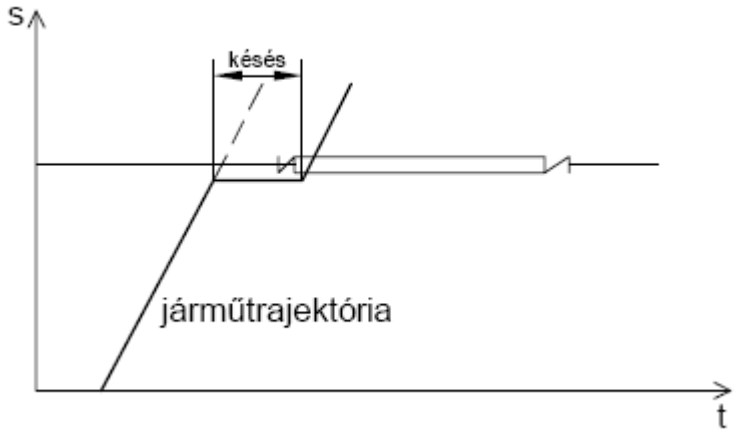
A **vegyes rendszerekben** a berendezések egy része közvetlenül csatlakozik a központhoz, másik részük alközponton keresztül. (5. ábra) Általában a központhoz közelebb eső gépek tartoznak az első csoportba. A budapesti forgalomirányítás vegyes struktúrájú.



5. ábra Vegyes forgalomirányítási struktúra

3. Az előnybiztosítás logikai megvalósítása [2]

Az előnyt biztosító rendszerek alapvető célja az, hogy csökkentsék a forgalmi helyzet miatt kialakuló késéseket. A viszonylag pontosan közlekedő – azaz a menetrendet tartó – járművekkel elérhető az eredetileg tervezett férőhelykínálat, így a közlekedési vállalatok nem kényszerülnek tartalékjárművek üzembe állítására, illetve ezek hiányában esetleges kártérítés, kötbér megfizetésére. A menetrend kialakításakor nem számolnak úgy menetidőt, hogy minden csomóponton feltartóztatás nélkül haladhat át a jármű, azonban az átlagos forgalomnagyságra elkészített menetrend nem tartható nagy forgalmi terhelés esetén. Az előnybiztosítás alapvető feltétele, hogy a menetrendet tartó autóbusz nem kap előnyt, hiába kér. Szintén nem kapnak prioritást a teljesen üres járművek. (Ez adódhat csekély kihasználtságból, vagy üzemviteli okból is, pl. garázsmenet, szolgálati járat.) A 6. ábrán látható egy előnybiztosítás nélküli jármű út-idő diagramja. Az ábrán megjelölt „késések” csak egy bizonyos számú előfordulásáig tartható a menetrend, utána a jármű valóban késni fog.



6. ábra Előnybiztosítás nélküli jármű mozgása ([2] alapján)

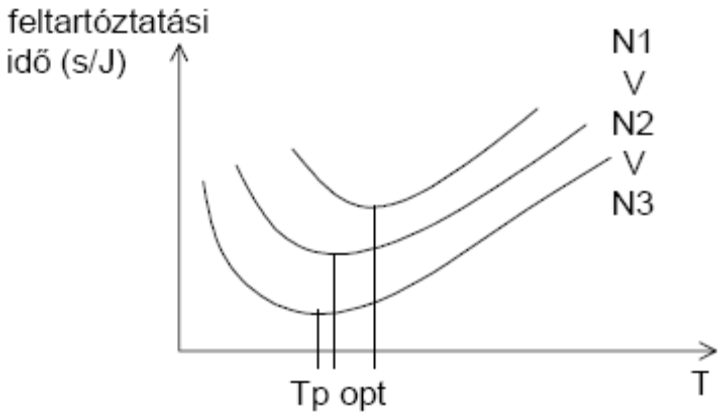
Ha figyelembe vesszük, hogy a csomópontba érkező járművekben hány személy utazik, egészen más értéket kapunk, mintha csak a járművek számát tekintjük. Például egy 45 másodpercet késő 3 utast szállító jármű és egy 45 másodpercet késő 115 utast szállító jármű igen nagy „súlykülönbséggel” szerepel a számításban. Ehhez azonban szükség van olyan adatgyűjtő és adattovábbító rendszerekre, amelyek el tudják juttatni az utasszámot a forgalomirányító berendezésbe. Az előnybiztosítási módszerek alapvetően kétféle csoportba oszthatók: a passzív ill. az aktív rendszerek csoportjába. A következőkben ezeket a módszereket részletezem.

3.1. Passzív előnybiztosítás

Ezen intézkedések a közlekedési jelzések megfelelő beállításával tehetők meg. Az intézkedések **minden ciklusban** lefutnak függetlenül attól, hogy jelen van-e közforgalmú közlekedési jármű, vagy nem.

Legegyszerűbb formája az, hogy az előnyben részesítendő viszonylat által használt csomóponti ágak **több zöldidőt kapnak**, így az érkező jármű nagyobb valószínűséggel találkozik szabad jelzéssel. Ezzel szemben az összes többi csomóponti ágba érkező jármű hosszú feltartóztatást kénytelen elszenvedni.

Egy másik módszer szerint **csökkentik** a csomópontban használt **jelzésterv ciklusidejét**, így egymás után gyakrabban következik szabad jelzés. Ezzel viszont csökken a csomópont kapacitása is, valamint arányaiban sokkal több veszteségidőt kell a sorbanálló járműveknek kivárni, mivel a közbenső idők többször is beillesztendők. (A közbenső idők értékei függetlenek a fázistervtől, csakis a csomóponti geometriától függenek.) A periódusidő és a járművekre jutó feltartóztatás ideje a 7. ábrán látható diagramok szerint összefügg egymással. Az ábráról leolvasható, hogy van egy a feltartóztatás szempontjából optimális periódusidő.



7. ábra A periódusidő és az egy járműre jutó feltartóztatás összefüggése

Ez a stratégia nagy forgalmi terhelésnél nagyon megnöveli a késéseket, habár bizonyos járművek rövid idő alatt átjuthatnak a csomóponton.

A harmadik megoldásban **egy fázis zöldidejét megfelezik**, és ugyanaz a fázis kétszer szerepel egy cikluson belül. Ezzel a megoldással is csökken a várakozás a következő zöldre, azonban itt nem csak arányaiban, hanem ténylegesen is nő a csomóponti veszteségidő, mert több fázist alkalmaz, így több közbenső idővel kell számolni.

Egy másik módszer a már említett vonali összehangoláshoz kapcsolódik. Az **összehangolás sebességét a közforgalmú járművek sebességéhez igazítják**. Ez azonban nem

mindig állandó, mivel az útvonalon megállóhelyek is vannak, amik önmagukban is változatossá teszik a jármű út-idő diagramját, emellett nem határozható meg pontosan, hogy egy megállóban mennyi ideig tart majd az utascsere. Az egyéni közlekedést használók jelentős késéseket szenvednek ilyen stratégia alkalmazásakor.

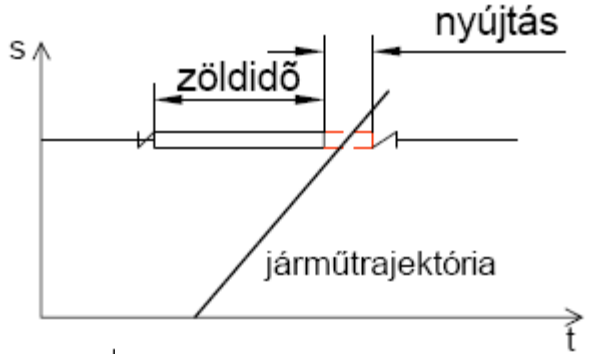
A fentiekből leszűrhető, hogy a passzív stratégiák nem hatékonyak. Alkalmazásukkor a csomóponti optimumtól jelentősen eltérő fázisterveket kell végrehajtani. Ez a többi közlekedő számára jelent nagy hátrányt. Az alkalmazásukkal kapott nyereség nincsen arányban a fellépő hátrányokkal.

3.2. Aktív előnybiztosítás

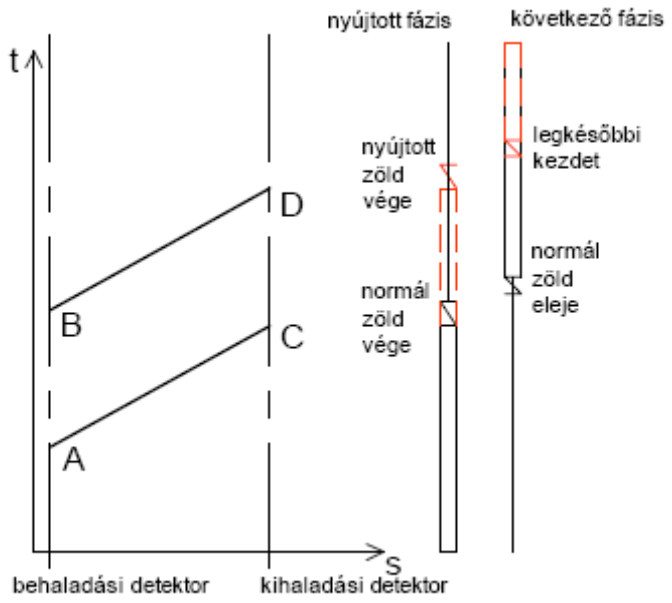
A passzív intézkedésekkel ellentétben ezek a módszerek csak akkor működnek, ha **jelen van előnyben részesítendő jármű**, azaz valós időben (real-time) alkalmazkodnak a forgalmi szituációkhoz. Megvalósításukhoz a 2.1. fejezetben említett programalkotó (opcionálisan adaptív) forgalomirányító rendszerek illetve adatgyűjtő és –továbbító berendezések szükségesek. Utóbbiak segítségével jut el a forgalomirányító berendezésbe a közforgalmú közlekedési jármű érkezésének ténye. Alapvetően négy aktív intézkedés kínálkozik a csomópontba érkező jármű prioritásának biztosítására.

3.2.1. Zöldidő nyújtása

A programalkotó rendszereknél leírt metódust lehet használni némi módosítással, miszerint a zöldidőt nem az érkező egyéb járművekre alapozva kell nyújtani, hanem az érkező autóbuszra vagy villamosra. Így a kitűntetett jármű elhaladása után már nem kell újabb hosszabbításokat végrehajtani. Az elvi működés a 8. ábrán követhető nyomon.



8. ábra A zöld nyújtás elvi működése ([2] alapján)



9. ábra A zöld nyújtás megvalósítása a fázistervben ([2] alapján)

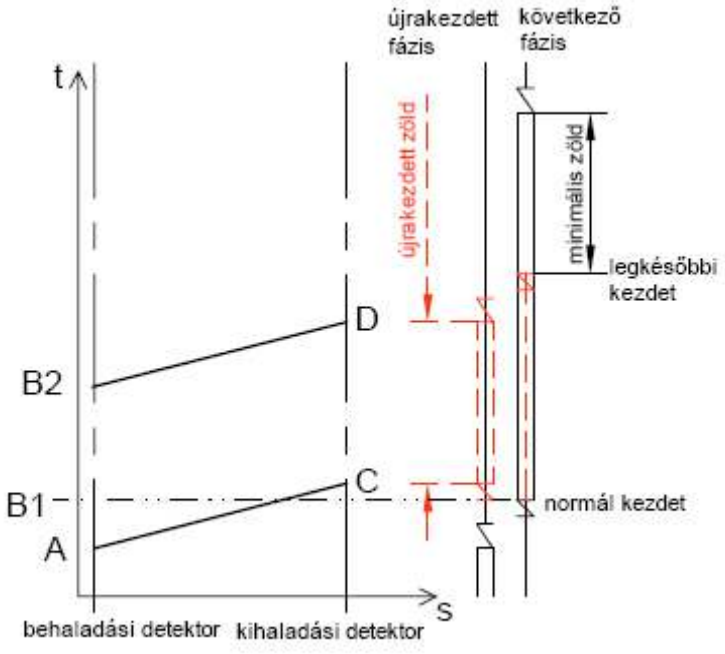
A 9. ábra alapján részletesen is megvizsgálható az intézkedés működése. Az ábra bal oldalán egy idő-út diagram található, amelyben két jármű mozgása van ábrázolva. A pontvonalakkal jelölt keresztmetszetekben a csomópont előtt behaladási detektor ill. a csomópont után kihaladási detektor van telepítve. A jobb oldalon a fázisterv részlete látható, mégpedig a nyújtott fázis, melyben a kitűntetett jármű érkezik, illetve az ezt követő fázis. Az ábrán az előnybiztosítás hatására a fázistervben történt változások vannak szaggatott piros vonalakkal jelezve. (Ez a későbbi hasonló struktúrájú ábrákra is vonatkozik.)

Azok a járművek, amelyek a C időpontig elhagyják a csomópontot, nem igényelnek fázisnyújtást, viszont a kereszteződést a C és a D időpont között elhagyók már igen. (A D pont a nyújtás végének időpontját jelöli.) A D időpont határozza meg, hogy a következő fázisban mikor kezdődhet egy keresztező irányban a szabad jelzés. A behaladási detektor általában a csomópont előtt 50-100m-rel helyezkedik el, ezért a bejelentkezés és a csomópontba behaladás között - a sebesség függvényében – bizonyos idő eltelik, emiatt a nyújtás időintervallumát el kell csúsztatni a sebességnek megfelelően. Ezt reprezentálja az A és a B időpont. Az előbbi fejtegetésben feltétel volt, hogy a jármű a behaladási detektor és a csomópont kezdete között állandó sebességgel haladt. Sajnos ez az ideális állapot ritkán áll fent, a jármű változó sebességgel halad, ezért az eltolás mértékét a forgalmi helyzet és egyéb paraméterek ismeretében változtatni kell.

3.2.2. Zöld újrakezdés

A második intézkedés a zöldidő nyújtásának egy speciális esete, azaz a forgalomirányító berendezés a már éppen befejeződött fázis zöldidejét ismét kiadja. Az egyszerűbb zöld

nyújtástól abban különbözik, hogy a jármű bejelentkezése nem az éppen futó zöldidőben történik meg, hanem a két követő fázis közti közbenső időben. Működése a 10. ábra alapján a következő:



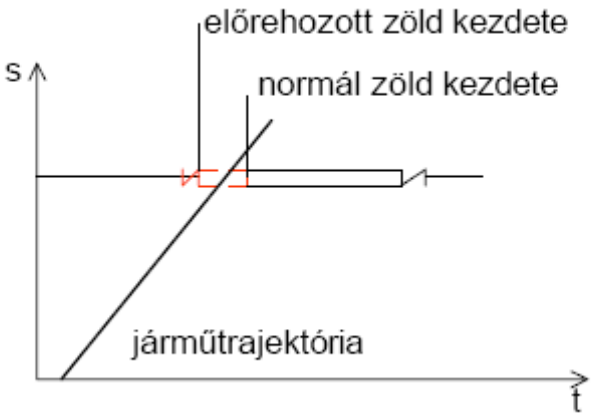
10. ábra A zöld újrakezdés megvalósítása a fázistervben ([2] alapján)

Az újrakezdett zöld a C és a D időpontok között valósítható meg. A C időpont a piros-sárga és a sárga jelzés hosszától függ. Piros jelzés kiadása elvileg nem lenne szükséges, azonban egy minimális hosszúságút célszerű kiadni, hogy a járművezetők ne gondoljanak a lámpa meghibásodására. A D időpontban be kell, hogy fejeződjön az újraindított fázis, mert a következő minimális zöldidejének kiadása csak így biztosítható. Az A pont jelzi azt a legkorábbi időpontot, ahonnan kezdve az újraindítás lehetséges. Ha az A pont előtt jelentkezik be a kitűntetett jármű, az előny a zöld nyújtásával biztosítható. A B1 időpont pedig a következő fázis kezdetét jelképezi. A B1 időpont elhagyása után a következőkben ismertetett stratégiák alkalmazhatók. Ha az előnyt kérő jármű után újabb közforgalmú járművek érkeznek, az újrakezdett fázis is hosszabbítható, aminek a B2 időpont szab határt, ugyanis a következő fázisban is meg kell, hogy legyen a minimális zöldidő. A B2 után érkező járművek előnyét szintén az alábbiak alapján lehet megadni. Ezt az intézkedési igen ritkán használják, ugyanis igen nagy feltartóztató hatása lehet a nem priorizált irányokban, emiatt a Java-programba sem építettem be.

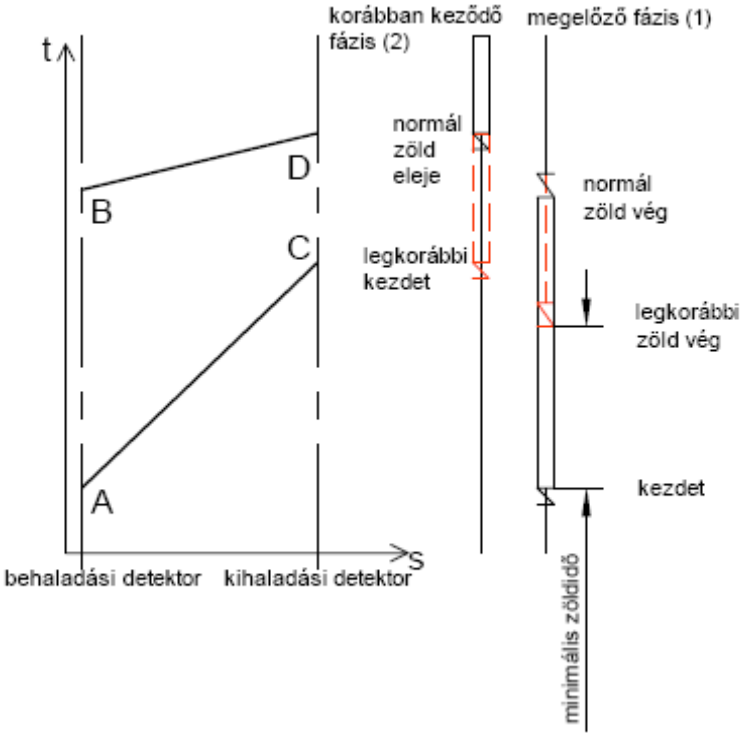
3.2.3. Zöld előrehozása

Ha a jármű tilos jelzéshez érkezik, két eset lehetséges: a következő fázisban az az irány is zöldet kap, amiben az előnyben részesítendő jármű is áll, ill. a következő fázisban sem tud áthaladni a csomóponton.

Az első esetben az éppen futó fázist (egy minimális zöldidőt kiadva) előbb befejezi a gép, és ezután átlép a következő fázisra, azaz előbb kezdődik a zöld jelzés, így a jármű hamarabb elhagyhatja a csomópontot. Ezt a 11. ábra szemlélteti.



11. ábra A zöld előrehozás elvi működése ([2] alapján)

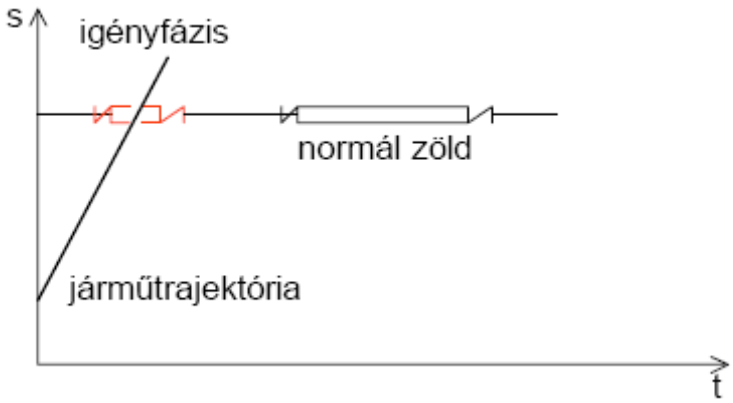


12. ábra A zöld előrehozás megvalósítása a fázistervben ([2] alapján)

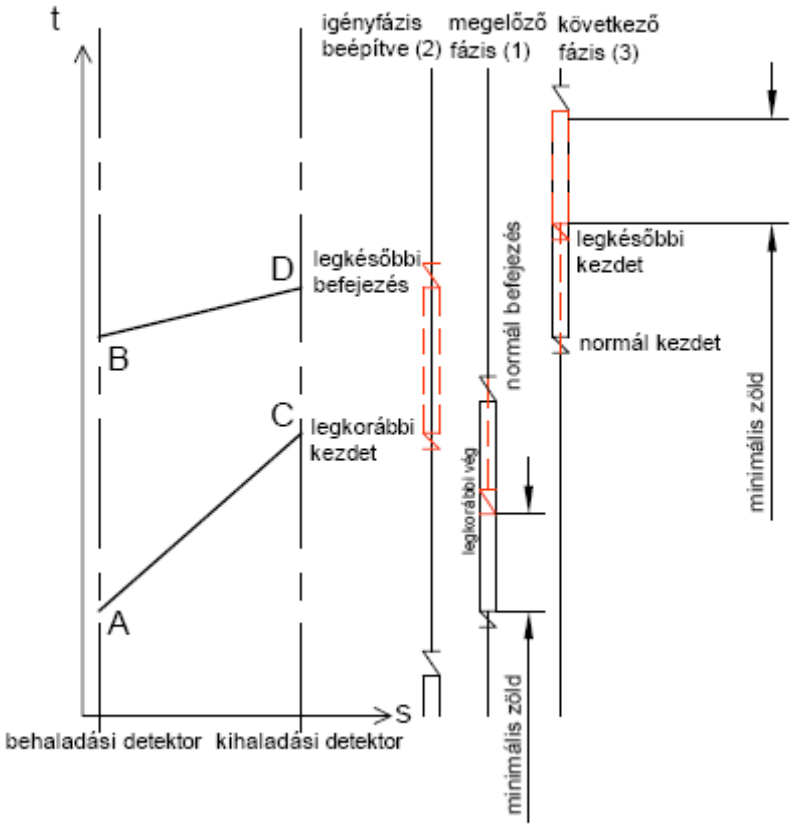
A pontos működés a 12. ábra alapján nyomon követhető. A zöldidő korábbi kezdése a vázlaton C és D pontokkal jelölt időintervallumban lehetséges. (A D pont a normál zöld kezdőpontját jelöli.). Az előny biztosításakor ebben az esetben tekintettel kell lenni az (1) jelű megelőző fázisra is, ugyanis itt is ki kell adni egy minimális zöldidőt: csak ennek letelte után lehet a csomóponton a kitűntetett járművet a következő fázisban átjuttatni. A minimális zöldidő hossza jelöli ki a (2) jelű fázisban a zöld legkorábbi kezdetét. A korábbi zöldet a jármű az A és a C időpontok között kérheti. Az A pont előtt ebben a stratégiában nem lehetséges a kérelem, mert akkor még az (1) jelű fázis sem kezdődött el. Ekkor igényfázist kell beiktatni. (lásd alább) Az A és a C pont között az (1) minimális zöldje miatt kell várakozni a korábbi kezdésre.

3.2.4. Fázisbeillesztés

A második esetben új fázist illesztünk be a jelzések közé. Ez a fázis lehet egy később következő fázis, vagy egy ún. igényfázis is. Utóbbi esetben például egy buszsávból kell balra kanyarodni. Ideális esetben ezt a sávot egyéb járművek nem használják, így csak akkor kell számára zöldet adni, ha egy érkező autóbusz ezt igényli. Elvi működését a 13. ábra szemlélteti.



13. ábra A fázisbeillesztés elvi működése ([2] alapján)



14. ábra A fázisbeillesztés megvalósítása a fázistervben ([2] alapján)

A 14. ábra alapján az új fázis beillesztése a C és a D időpontok között valósítható meg. Lehetséges kezdetét az (1) jelű megelőző fázis minimális zöldideje, legkésőbbi befejezését pedig a (3) jelű követő fázis minimális zöldideje határozza meg. Az előnykérés az A és a B időpontok között történhet meg. Az A pont előtt ez nem lehetséges a fázis korábbi kezdésénél meghatározott ok miatt. A B pont után bejelentkező járműnek a már ismertetett feltételek betartása mellett korábban kell kezdeni a zöldet.

3.2.5. Az intézkedések értékelése

A négy bemutatott intézkedés az 1. táblázatban foglalható össze:
Ha a jármű bejelentkezése az előnyben részesítendő fázis

előtti piros- és közbenső időben	történik, akkor	korábban kell zöldet kezdeni
zöldidejében		zöldet kell nyújtani
utáni közbenső időben		újra kell kezdeni a zöldet
utáni pirosidőben		új fázist kell beilleszteni

1. Táblázat Az előnybiztosítási intézkedések összefoglalása

Az aktív stratégiák tökéletesítése jelenleg is folyamatban van. Az előbb leírt intézkedések tovább finomíthatók, így három csoportba sorolhatók be: feltétel nélküli stratégiák, feltételes stratégiák ill. adaptív stratégiák.

Feltétel nélküli stratégiák: ebben az esetben az előnyt minden érzékelt jármű megkapja, függetlenül attól, hogy az a menetrendhez képest késik-e, vagy siet, szállít-e utast, vagy nem. Alapvető feltétel, hogy a menetrendhez képest sietők járművek nem kapnak előnyt a csomópontoknál, hiszen így még nagyobb menetrendi eltérés keletkezne.

A feltételt használó stratégiák leggyakrabban a már említett menetrendtől való eltérést vizsgálják. A sietők jármű nem kaphat előnyt, a késő viszont igen. A menetrendnek megfelelően közlekedő járatok sem élhetnek ilyen lehetőséggel. Emellett figyelembe vehető a szállított utasok száma, illetve a járat útvonalán távolabb detektált speciális forgalmi helyzetek. Ezen stratégiák továbbfejlesztésénél a fő cél az, hogy a minél jobb menetrendhez való igazodás mellett a beavatkozásoknak minél kisebb hatása legyen a többi közlekedőre. A témában számos szimuláció és terepi próba eredménye megtalálható a [2] jelű dokumentum hivatkozásai között.

Az adaptív stratégiák alapvetően abban különböznek az eddig bemutatottaktól, hogy mindig optimalizálást hajtanak végre, és eszerint határozzák meg a beavatkozás idejét és mikéntjét. Mivel adaptív forgalomirányító rendszereknél nincsenek előre definiált fázistervek, így a bemutatott négy aktív intézkedés megtételére sincs feltétlenül szükség. A többi csomóponti ágban lévő járművek számának figyelembe vételével a forgalomirányító berendezés ciklusonként újraalkotja a jelzéstervet, és ebbe bekalkulálja az érkező közforgalmú járművet. Korábban említettem, hogy az eltérő foglaltságú járművek eltérő súllyal szerepelhetnek a számításban.

Az adaptív rendszerek használata több problémát is felvet:

1. A számítás során az ilyen stratégiát alkalmazó berendezések általában hálózati szintű jellemzőket vesznek figyelembe. Ennek ellentmond az, hogy az előnyt többnyire lokálisan egy adott csomópontban kell biztosítani.
2. Az algoritmusok makroszkopikus mérési adatok alapján számolnak, hiszen ezek viszonylag egyszerűen mérhetőek, azonban az egyes járművek közlekedésének menetrendhez való viszonya mikroszkopikus jellemző. Emellett a megállók helyzete, a megállókhoz kapcsolódó problémák (lásd fentebb) szintén nem vehető figyelembe makroszkopikus szinten.
3. Kis és közepes forgalomnagyság esetén még kis ráhatással van egy előnyt biztosító intézkedés a többi közlekedőre. Nagy terheléseknél viszont már az optimalizálás is nehézkes, ezt tovább nehezítik az autóbuszra vagy villamosra vonatkozó feltételek. A számítási időigény jelentősen megnőhet ilyen esetekben.

4 Új, előnybiztosítást is tartalmazó program kifejlesztése

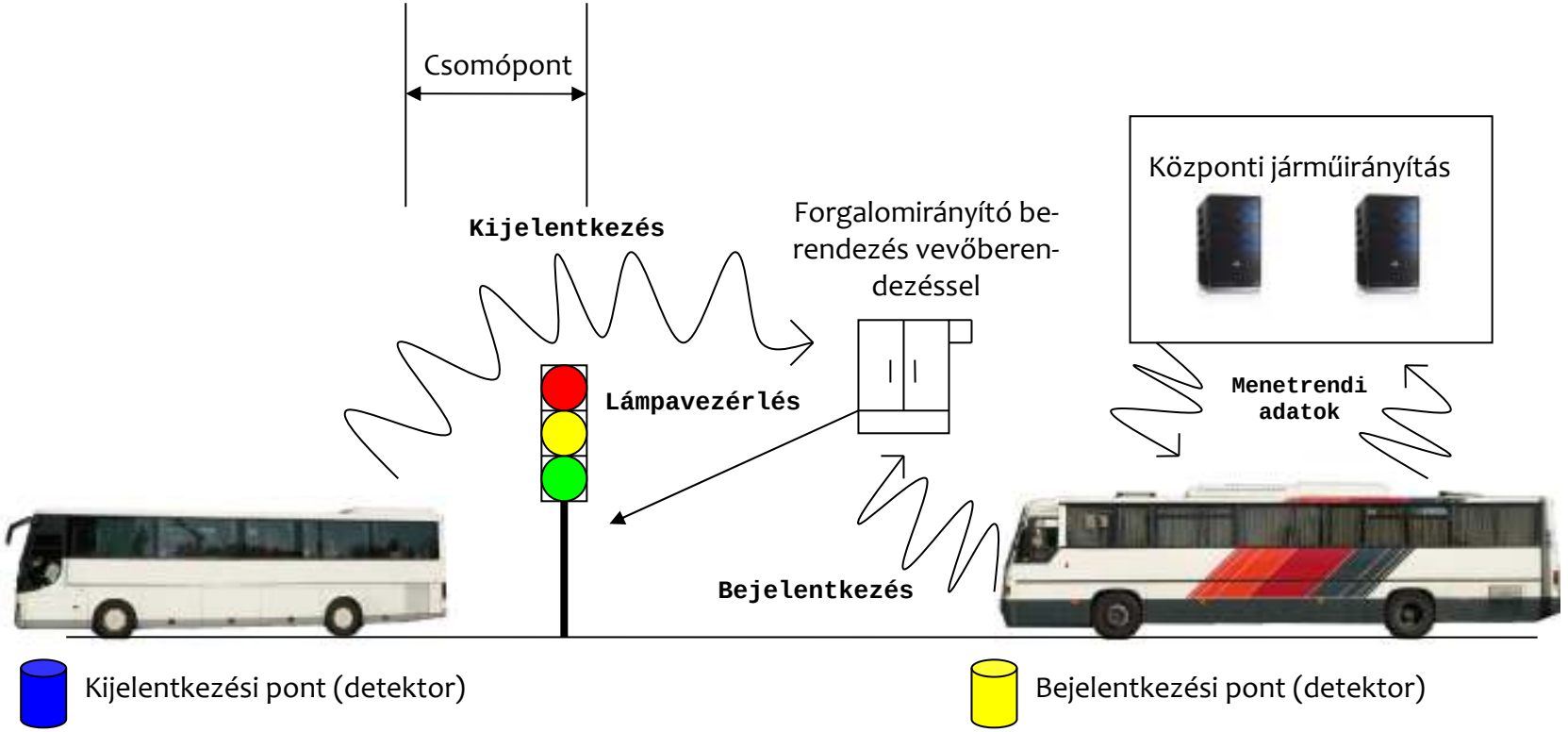
4.1 Fizikai követelmények

4.1.1 Általános felépítés

A bemutatott intézkedések végrehajtásához speciális hardverelemekre van szükség. A forgalomirányító berendezésnek képesnek kell lennie forgalomfüggő programok kezelésére. Ilyen például a Signalbau Huber vállalat Actros VTC 3000-es készüléke. Ez azonban nem elegendő, mert a berendezésben el kell helyezni egy olyan rádióvevővel ellátott elemet, amely képes a járművek adatainak fogadására és ezek megfelelő formában történő átadására a vezérlőlogikának. A jármű fedélzeti számítógépe egy kisteljesítményű rádióadó segítségével juttatja el a megfelelő adatokat a forgalomirányító berendezésbe. A menetrendi eltérés kiszámításához és tárolásához szükség van egy központi járműirányítás rendszerre (pl. a BKV AVM [Automatikus Vonali Megfigyelő] rendszere). Ezen rendszer adatbázisában lévő menetrenddel lehet összehasonlítani a jármű helyzetét.

Az autóbusz aktuális helyzete két bejelentkezési ponton ismert. Az egyik bejelentkezési pont a csomópont előtt 50-100 m-re van elhelyezve, míg a másik a csomópont után közvetlenül. Előbbit nevezik bejelentkezési, utóbbit kijelentkezési detektornak. ilyen detektorokat értelemszerűen a csomópont összes olyan ágában el kell helyezni, amelyet menetrend szerinti autóbuszok is használnak.

A rendszer működése a 15. ábrán szemléltethető. Az autóbusz és a járműirányítási rendszer kapcsolata utóbbi kiépítésétől függ. Elterjedtek az ún. helykódadós megoldások, de követve az általános trendeket egyre több GPS-alapú információs rendszert helyeznek üzembe. Az előnybiztosítás szempontjából gyakorlatilag irreleváns a menetrendi adatok lekérdezésének módja. Az a fontos, hogy a csomópont előtt ezek az adatok rendelkezésre álljanak. A bejelentkezési ponthoz érve az autóbuszon elhelyezett jeladó által sugárzott információkat befogja a forgalomirányító berendezésbe telepített vevő. A berendezés valamilyen algoritmus segítségével tárolja a bejelentkezést. Ha egyszerre több autóbusz is érkezik a csomópontba, akkor egy előre megírt logika szerint a berendezés dönt arról, hogy melyik jármű kap előnyt. A kiválasztott autóbusz a bemutatott aktív előnybiztosítási módszerek segítségével eléri a csomópontot, majd áthalad rajta. A kijelentkezési detektorhoz érve ismét „tudomást vesz” róla a forgalomirányító berendezés. Ennek következményeként be lehet fejezni a megkezdett előnybiztosítási intézkedést. Ha a továbbiakban nem jön több autóbusz, akkor a csomópont forgalmát a betáplált „autóbusz nélküli” jelzésterv szabályozza.



15. ábra Hardverelemek és kapcsolataik

4.1.2 Az ACTROS berendezés bemutatása

A BME Közlekedésautomatikai Tanszéke 2006 őszén szert tett egy ACTROS VTC 3000 típusú forgalomirányító berendezésre, melyet a Vilati - Signalbau Huber Forgalomtechnika KFT (jelenleg Swarco Traffic Hungary KFT) bocsátott az egyetem rendelkezésére. Az Actros berendezés napjaink legkorszerűbb technológiáját képviseli, a legújabb kutatási eredmények demonstrálását is lehetővé teszi. Jelenleg (2009. szeptember) még egyetlen ilyen berendezés sincs üzemben Budapest utcáin, azonban hamarosan megjelennek majd a Swarco jóvoltából.

A berendezés alábbi jellemzői jól mutatják, hogy egy felhasználóbarát, könnyen programozható gép kerül az üzemeltetőhöz:

- moduláris felépítésű, így bővíthető a funkcióknak megfelelően
- a belső kommunikáció nagy sebességű CAN-hálózaton keresztül történik
- több csomópontot (3) is tud egy gép kezelni, ezért csökkennek a telepítési és az üzemeltetési költségek
- TCP/IP ill. COM-felületen keresztül lehet a berendezéssel kommunikálni
- támogatja az OCIT-technológiát
- hibatárolási és hibakiolvasási funkciója is van
- Java nyelven programozható, így bonyolult logikák is megvalósíthatók benne

A berendezéshez kapcsolódik OCIT-protokollon keresztül egy 4 db detektort tartalmazó egység is. Ezek a detektorok ugyanúgy működnek, mint az úttestbe épített induktív hurokdetektorok, azonban sokkal kisebb fémtömeg hatására is elhangolódnak.

A Java a C nyelvhez hasonlóan objektumorientált programozási nyelv, így az Actros berendezés programja is objektumokból épül fel, amelyek nagy részét a gyártó készítette el, de ezek kiegészíthetők saját objektumokkal is. A „gyári” objektumok minden részlete meghatározott, azaz számos beépített függvény és eljárás tartozik hozzájuk. Ezen metódusokkal szinte minden jellemző lekérdezhető az objektum aktuális állapotáról.

A program alapegysége a class, amely magyarul osztályt jelent. A fázisok, programok, egyéb jellemzők is osztályként vannak definiálva, így több azonos osztályba tartozó objektumnak megegyező tulajdonságai is vannak. (pl. a metódusok minden a classba tartozó objektumnál azonosak.) Az egyes konkrét fázisok és programok ezen osztályokból eredeztethetőek. A forgalomirányító berendezés programozásakor a közbenső időket és az ezalatt elvégzendő feladatokat sajátosan kell definiálni, ugyanis ezeket is fázisként kezeli, ill. fázisátmenet néven ismeri a berendezés. Java-alapú forgalomtechnika programozásakor szokás, hogy az összes változót (variables) egy Var.java vagy V.java classban definiálják. Az így létrehozott változók „globális” változók, azaz az összes többi objektum értékükhöz hozzáfér, azt módosíthatja.

A berendezés telepítésekor nagyon sok az adott csomópontra jellemző konstans értéket kell felvenni. Ilyenek például a közbenső idők, a zöld és piros időkre vonatkozó minimumértékek, a fázisok, a jelzőfejek, fénypontok típusa, a detektorok, a berendezés belső kapcsolása, stb. Ezeket az értékeket a berendezés elindításakor egy ún. Init.java fájlból olvassa be. Ezt a folyamatot nevezik inicializálásnak. Az egyes programok is külön külön objektumnak tekinthetők. Az alábbi felsorolásban szereplő programok közül választ a berendezés a megadott időtervnek, vagy szituációknak megfelelően:

- bekapcsolási program (Einprogramm)
- sárga villogó (Blinkprogramm)
- fix zöldidős program (Festenzeitprogramm)
- forgalomfüggő program (VAProgramm)
- kézi vezérlés
- kikapcsoló program (Ausprogramm)

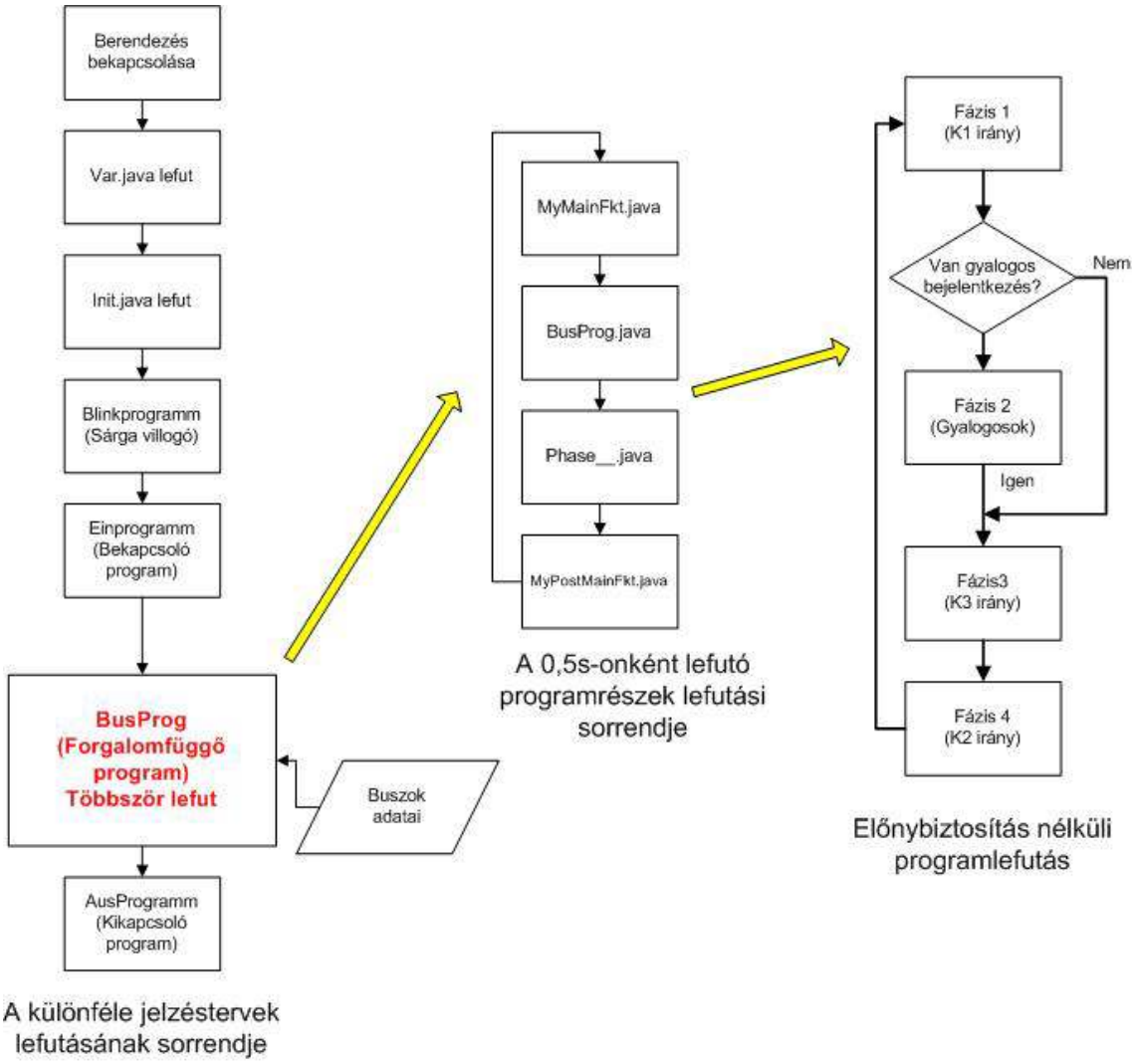
A berendezés üzembe helyezése és üzeme során az eddig említett programrészek az alábbi sorrendben futnak le: a bekapcsolást követően 10s-os inicializálás következik (Init.java lefutása), majd a sárga villogó programra vált a vezérlés. A sárga villogóból a bekapcsoló programot, majd ezután az előző kikapcsoláskor futó programot indítja el a berendezés.

A MyMainFkt.java minden ciklus elején lefut. A bekapcsolt jelzéstervet tartalmazó programrész 0,5s-onként fut le, az éppen aktuális fázis java fájlja szintén.

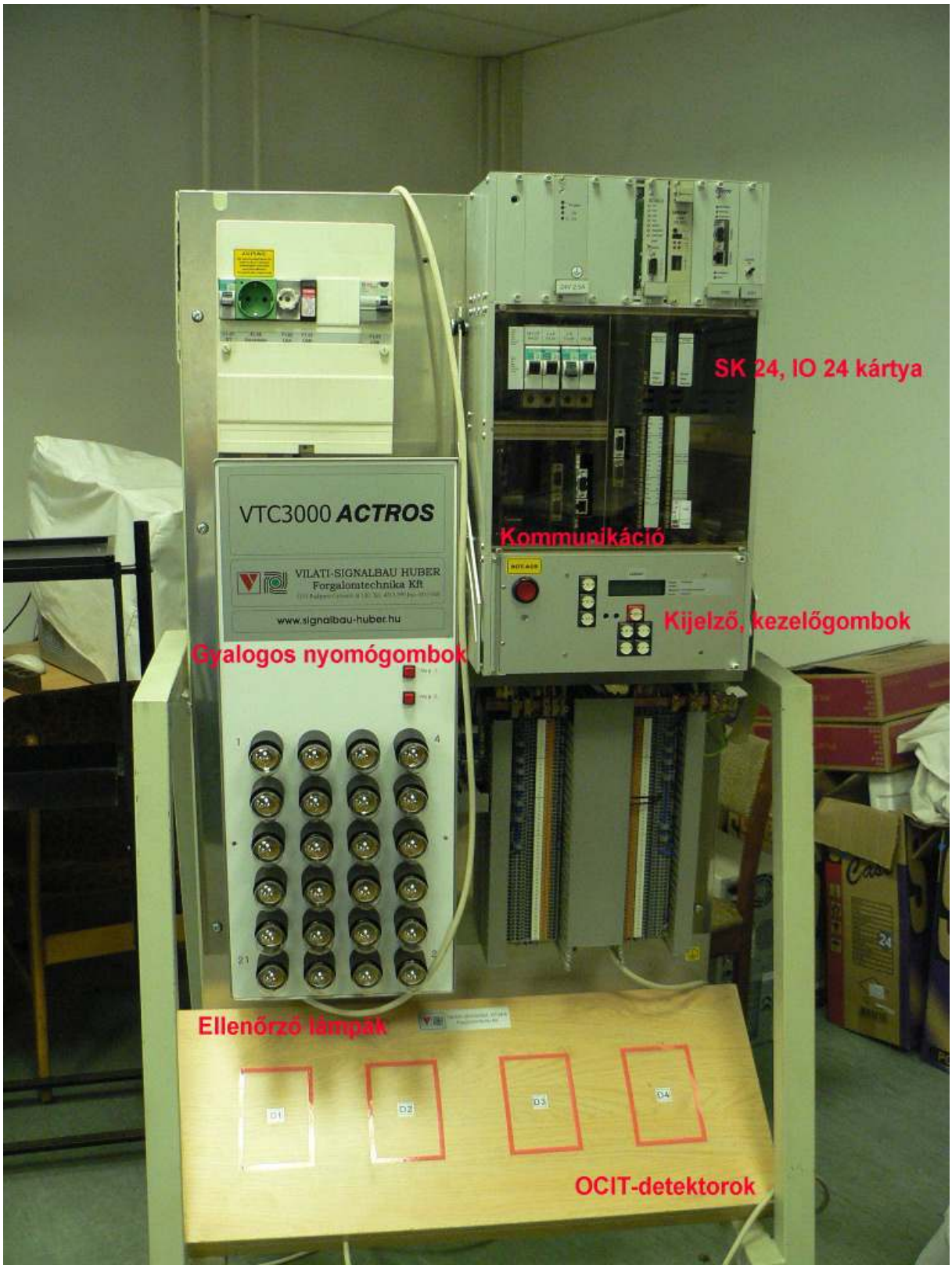
A fázisok közti átkapcsoláshoz szükséges feltételek a MyMainFkt.java classban találhatók meg.

Ha saját objektumot hozunk létre, akkor nekünk kell gondoskodnunk az abban lévő utasítások futtatásáról.

A Java program működését a 16. ábra foglalja össze. Az Actros a 17. ábrán vehető szemügyre, a berendezés hardver, ill. szoftverfelépítéséről bővebben a [4], [5] és [6] jelű dokumentumokban lehet olvasni.



16. ábra A program folyamatábrái

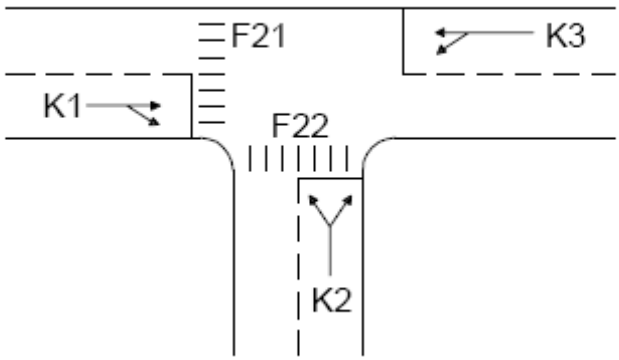


17. ábra Az ACTROS VTC 3000 forgalomirányító berendezés

4.2 A program elkészítésének lépései

4.2.1 A forgalomtechnikai keretrendszer

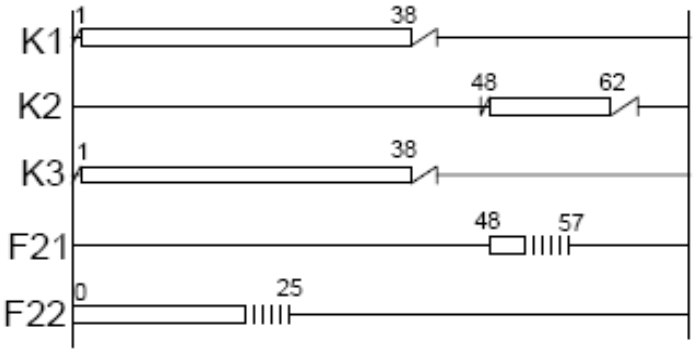
A bemutatott forgalomirányító berendezés már tartalmaz egy előre definiált csomópontra egy egyszerű jelzéstervet. Az irányított csomópont egy városi T-alakú kereszteződés, amelyet a 18. ábra szemléltet.



18. ábra A mintacsomópont

Az ábrán a programban használt jelöléseket alkalmaztam. A rövidítések jelentése a következő:

F – Fußgänger: gyalogos
K – Kraftfahrzeug: gépjármű



19. ábra A "gyári" program

A továbbiakban főiránynak tekintem a K1 és K3 irányokat, míg a mellékirány a K2 jelű. A gyárilag feltöltött program telezöldes irányítást valósít meg. Jelzésterve a 19. ábrán vehető szemügyre. A gyalogosjelzők csak akkor váltanak szabad jelzésre, ha a megfelelő beépített gyalogos nyomógombot megnyomták. Ezt az igényvezérelt működést az új programban is megvalósítottam.

A telezöldes irányítás miatt a csomópont közbensőidő mátrixa nem volt teljesen kitöltve. Az új forgalomfüggő program megvalósítása közben szükségessé vált az egyes fázisok elkülönítése az előnybiztosítási intézkedések demonstrálásához. Az irányok szétválasztása miatt teljesen ki kellett tölteni a közbensőidő mátrixot. A 2. táblázatban megtalálható mátrix pirossal jelölt értékeit én vettem fel a meglévő értékek figyelembe vételével. (Megjegyzés: a közbensőidő mátrix változtatásai nincsenek átültetve a Java-kódba, mivel az Actros berendezés sajátosságai miatt ehhez EP-ROM-ot kellene égetni. A program futás közben figyelembe veszi a megfelelő közbensőidő értékeket, de az ellenőrzési metódusoknál csak a benne lévő (gyári) közbensőidő mátrix tartalmát tudja kontrollálni. Eme hiányosság miatt a program „út menti” használatra alkalmatlan, ehhez ki kell egészíteni, és EP-ROM-ot kell égetni.

Irány	BE					
KI	program	K_1	K_2	K_3	F_{21}	F_{22}
	K_1	-	6	5	4	6
	K_2	6	-	6	8	4
	K_3	9	6	-	7	10
	F_{21}	10	7	10	-	--
	F_{22}	4	14	8	--	-

2. Táblázat A kiegészített közbensőidő mátrix

Az 4.1.1 fejezetben bemutatott fizikai modellben szerepel egy bejelentkezési és egy kijelentkezési detektor. A már működő rendszerek legtöbbjénél ez a valóságban is megvan, azonban a Tanszéken rendelkezésre álló hardverelemek között nincs. Az 5.2 fejezetben tárgyaltak szerint négy detektor áll rendelkezésre a programok kipróbálására. A csomóponti geometriát változatlanul hagyva, de feltételezve, hogy minden irányból minden irányba közlekedhetnek autóbuszok, hat detektorra lenne szükség. Mivel ez nem valósítható meg, így minden irányban csak a bejelentkező detektort építettem be a programba, és megbecsültem, hogy az autóbusznak mennyi időre van szüksége ahhoz, hogy a csomópontot elhagyja. A következő számítások ezen időtartamok meghatározását szolgálják.

A) eset: tételezzük fel, hogy a detektorok 100m-re helyezkednek el a csomóponttól.

Az autóbusz sebessége legyen $20\frac{km}{h}$ a főirányban, $24\frac{km}{h}$ a mellékirányban. (Mellékirányban kisebb forgalomnagyságot feltételezve nagyobb átjutási sebességgel lehet számolni.)

Főirányban $t_{fő} = \frac{100}{\frac{20}{3,6}} = 18s$ az átjutási idő, míg mellékirányban $t_{mellék} = \frac{100}{\frac{24}{3,6}} = 15s$ a csomópont eléréséhez szükséges idő. Ha a főirányban a stopvonal és a csomópont külső széle közötti táv 4s alatt, mellékirányban 5s alatt küzdhető le, akkor a főirányban 22s-os, a mellékirányban 20s-os átjutási időt kapunk. (Megjegyzés: a 4 és 5s idők már a közbensőidőbe számítanak.)

Ha ezeket az időtartamokat használnánk a program módosítására, jelentős növekedés következne be az előnyben nem részesített forgalmi irányokban. Emellett nagyon sok idő telne el az előnyadás és a tényleges áthaladás között. Ennek kiküszöbölésére tekintsük a B) esetet.

B) eset: a detektorok a csomóponttól 50m távolságra vannak telepítve.

A sebességeket változatlanul hagyva a következőket kapjuk:

$t_{fő} = \frac{50}{\frac{20}{3,6}} = 9s$ és $t_{mellék} = \frac{50}{\frac{24}{3,6}} = 7,5s \longrightarrow 8s$ a két átjutási idő

A csomópont eléréséhez szükséges időből kiindulva állapítottam meg a zöld nyújtás és a fázisbeillesztés időtartamát. A főirányban az esetleges forgalmi egyenlőtlenségek miatt 10s-ra választottam az előnyadás zöldidő szükségletét. Mivel a közúti forgalomirányításban a legkisebb időegység a másodperc, a mellékiránybeli zöld hosszát 8s-ra kerekítettem.

A zöldidő megadás és az átjutási idő kapcsolatából megállapíthatók a következők. Ha egy autóbusz bejelentkezik a detektoron keresztül, és a sorbanálló járművek miatt nem tud előrébb haladni, az előnybiztosítás zöldidő hossza miatt biztosan át tud jutni a csomóponton. Ha egymás után több busz is bejelentkezett ugyanabból az irányból, akkor mindegyik biztosan elérte a detektort is. Ha ez az irány előnyt kap, akkor az összes jármű át tud majd haladni a csomóponton. A meghatározott stopvonal eléréséhez szükséges időt célszerű a minimális zöldidőként is felhasználni, így egyszerűsödhet a készülő program, a két érték egyenlőségét jól fel lehet használni a program megalkotásakor.

A detektorok elhelyezése a csomópontban a következő: a K1 irányban található az 1-es számú, a K2 irányban a 2-es számú, így a K3 iránynak már csak a 3-as számú marad.

4.2.2 A forgalomfüggő program előnyadási stratégiája

4.2.2.1 Bemeneti paraméterek

A rendszer működésének bemutatásánál említettem, hogy a forgalomirányító berendezés valamilyen előre megadott logika szerint választ a bejelentkezések közül. Az általam elkészített program döntési mechanizmusa az alábbiakban kerül kifejtésre.
Minden egyes autóbuszt pontszám alapján rangsorol a gép. A pontszám több részből tevődik össze. Egy menetrend szerint közlekedő autóbusz négy jellemzőjét használom fel a pontszám kialakításában, nevezetesen:

1. Hány utast szállít az adott jármű?
2. Mekkora késéssel bír a menetrendhez képest?
3. Helyi- vagy helyközi járatról van-e szó?
4. „Lassú”, vagy gyorsjáratként közlekedik-e a jármű?

A valóságban az első jellemzőt a járművön elhelyezett utasszámláló berendezés határozza meg. Magyarországon elsősorban olyan berendezések működnek, amelyek a jármű légrugójának terhelése alapján határozzák meg az utasszámot. (Külföldön működnek más rendszerű mérőeszközök is, például lépcsőbe épített nyomásra érzékeny szenzorok; vagy ajtó mellé telepített fotocellás érzékelő.) A busz menetrendi eltérését a központi járműirányítási rendszer segítségével lehet meghatározni. Ez a BKV Zrt. esetében praktikusan az AVM[1] ill. DIR[2] rendszert jelentené, Volánok esetében a jelenleg több helyen kiépítés alatt álló GPS alapú Vultron[3] információs rendszert. A járat jellemzőit (lassú/gyors, helyi/helyközi) szintén a járműirányítási rendszer segítségével lehetne a forgalomirányító berendezés számára elérhetővé tenni.

A forgalomirányító berendezés programjának kidolgozásakor nem állt rendelkezésre sem igazi autóbusz valódi utasokkal, sem járatinformációs rendszer. Emiatt a program véletlenszám-generátorral állítja elő egy-egy bejelentkezés utasszám és késés adatait a következőképpen: az utasszám 0 és 150 között lehetséges. Az üres autóbusz azért került a programba, hogy minél jobban hasonlítson a valóságos működésre. A 150 fős utasterheltség pedig egy városi csuklós autóbusz összkapacitásának felel meg. A késést 0 és 20 perc közé sorsolja a gép. A valóságban 20 perc feletti késések csak ritkán fordulnak elő. Ha a valóságban is használni kívánjuk a programot, akkor egy feltétellel meg lehet határozni, hogy a 20 perc feletti késéseket is 20 percnek tekintse a forgalomirányító berendezés.

[1] Automatikus Vonali Megfigyelőrendszer
[2] Diszpécser Irányító Rendszer
[3] Részletesebb információk: www.vultron.hu

4.2.2.2 A pontszámok előállítása

Az utasszámot nem abszolút értékben veszi figyelembe a gép, - mivel a nagy utasszámokkal nem lenne összemérhető a késés, valamint az egyéb jellemzők által adott pontszámrész – hanem csak a tizedét. Késésként a generált számot használja a vezérlés. A járat jellemzőinél a helyközi gyorsjáratoknak adtam legnagyobb prioritást, a helyi lassújáratoknak pedig a legkisebbet. Véleményem szerint nagyvárosokban a helyközi járatok menetidejének betartása fontosabb, mint a helyieké, mivel általában ritkább követési idővel közlekednek, mint városi társaik. Ennek megfelelően a helyközi minőség 2 pontot ér, a helyi 1-et. A gyorsjáratok 4 ponttal gazdagodnak, a lassúak pedig 3-mal. (Azért volt szükség a két jellemzőnél más-más pontértékekre, mert így csökkenthető az azonos pontszámmal rendelkező autóbuszok száma.)

A kapott pontszámok súlyozva kerülnek összeadásra, a súlyok összege 1-et ad ki. A programban az utasszám tizedét 0,4-es szorzóval vettem figyelembe. Véleményem szerint annál fontosabb, hogy egy jármű hamar átjusson egy csomóponton, minél több utast szállít. Ha egy 100 embert szállító jármű 10 percet késik, az felfogható 1000 perc utaskésének is, mert minden, a járművet igénybe vevő utas késik. A menetrendi eltérés 0,3-as súlyt kapott. A helyi/helyközi jellemző 0,2-szeresen számít az összpontszámba, a járat gyorsasága pedig 0,1-el. A pontok súlyozott összegét akár költségfüggvénynek is tekinthetjük, mivel alakja az alábbihoz hasonló:

$$J = s_1 \cdot A_1 + s_2 \cdot A_2 + ...$$

(Sok esetben az *A* értékek mátrixok, az *s* értékek pedig oszlop, ill. sorvektorok.)

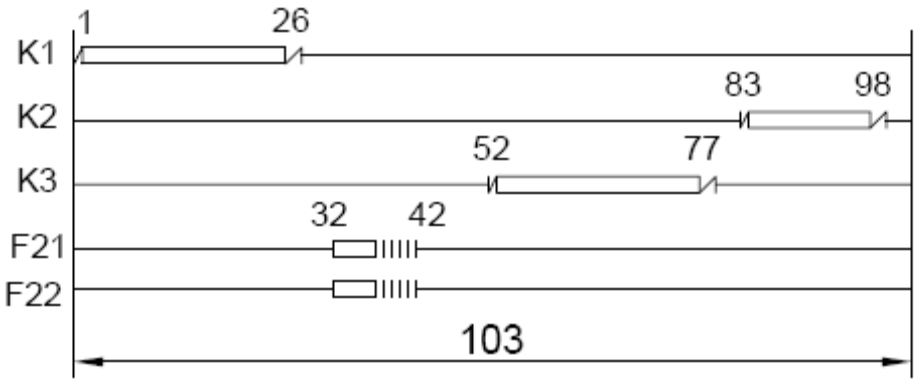
A súlyozott jellemzők összege lehetne a pontszám, azonban a leghamarabb prioritást igénylő (zsúfolt, sokat késő) jármű a legnagyobb pontszámot kapná. Irányítástechnikai feladatoknál az említett költségfüggvényt minimalizálni szokták. Ahhoz, hogy a legnagyobb prioritást igénylő jármű a legkisebb pontszámot kapja, nem kell mást tenni, mint az súlyozott összeg reciprokát venni. Az így kapott érték már ténylegesen pontszámnak tekinthető.

A bejelentkezések közül a gép egyszerű minimumkeresési eljárással választja ki a legkisebb pontértékkel rendelkezőt.

4.2.3 Az új program elemei

4.2.3.1 Jelzésterv

Az autóbuszok közlekedésétől függő program alapvetően egy fix fázisidős program, jelzésterve a 20. ábrán látható.



20. ábra A felhasznált jelzésterv

A fázissorrendet a közbenső idők összegének minimuma alapján határoztam meg. A gépben lévő alapjellemzőket nem változtattam meg, így az előkészítő idő 1s, az átmeneti idő pedig 2s nagyságú.

Az autóbuszok előnyben részesítésével ezen program jelentős mértékben módosulhat. Ha nagyon sok autóbusz érkezik a csomópontba, akkor nagyon sok változtatásra lehet szükség, így gyakorlatilag eltűnik a program ciklikussága, hiszen a bejelentkező autóbuszok irányítják annak lefolyását, ezért ciklusidőről nem beszélhetünk. Mivel a csomópont izoláltnak tekinthető, nem szükséges általánosan elterjedt periódusidőt választani. A program zöld nyújtást, fázisbeillesztést és zöld előrehozást képes az autóbuszoknak adni.

A módosításokat a program lefutásának logikai sorrendjében mutatom be, viszont bizonyos helyeken e fogok térni ettől. A 3. mellékletben a teljes forráskód megtalálható. Az általam írt vagy módosított programrészletek sárga színnel vannak kiemelve.

4.2.3.2 Var.java

Definiálni kellett a buszfüggő programot, mint forgalomfüggő programot. A fázisátmenetek itt deklarálandók, az itt található objektumok alapján tud majd az Init.java futásakor a gép inicializálni. Az egész programban globális változókat használok, így nem kell azzal törődni, hogy az egyes classok között egymásnak átadjak értékeket. Utóbbira lokális változók használata mellett szükség lenne. A következőkben leírt változókat használom:

utasszam (float): bejelentkezéskor ebbe a változóba generálja a gép a 0 és 150 közé eső utasszámot.
keses (float): ide kerül a generált menetrendi eltérés.
hhk(float): a helyijárat/helyközi járat jellemzőt tartalmazó változó.
lgy(float): A busz „gyorsaságát” tároló változó.
pontszam: a véletlenszám generátorral előállított értékek súlyozva, összeadva és reciprokot képezve kerülnek ide.
bejelentkezesek[][] mátrix: A mátrix összes eleme float típusú, így tárolhatók benne a float típusú változókból átadott elemek. Ebben a 7 sorból és 20 oszlopból álló tömbben tárolja a program a bejelentkezéseket. A hét sor a következő jelentéssel bír (indexek szerint):

0: két állapota lehet: 0 és 1. Az érték 0, ha az 1…6 indexű sorok üresek, 1, ha nem üresek.
1: két állapota lehet: 0 és 1. Az érték 1, ha az OCIT1 detektoron jelentkezett be a busz. Ellenkező esetben 0.
2: Hasonló az előzőhöz, de az OCIT2 detektorra vonatkozik.
3: Az előbbiekkel megegyező jellemzői vannak, az OCIT3 detektoron bejelentkező busz ír ide 1-et.
4: kiszámított pontszám kerül ide a *pontszam* változóból.
5: tartalék
6: Egyes kerül ide, ha ez a bejelentkezés előnyt kapott, (ld. a későbbiekben) ellenkező esetben 0.

vizsgalt[]: minimumkeresésnél ebbe a vektorba kerülnek be a pontszámok. (ld. lejjebb)
szamlalo, *szamlalo2*, *szamlalomin (int)*: ciklusszámlálók. Azért kell különválasztani a minimumkeresésnél használt változót, mert ez a programrész fut együtt a BusProg.java-val, amelyben a másik kettőt használtam.
min (int): a minimális pontszám oszlopindexét fogja tartalmazni. (ld. később)
elozofazis (int): fázis beillesztésénél tárolja az előző fázis számát. (1…4)
zoldk1k3=25: a főiránybeli zöldidő, nagysága 25s. Ez akkor valósul meg, ha az adott fázisban nem kerül sor semmilyen előnybiztosításra.
zoldk2=15: a 15s-os mellékiránybeli zöldidő. Ez is csak akkor igaz, ha nincsen beavatkozás a programba.
k1ben, *k2ben*, *k3ban (boolean)*: ezen változók igaz értékükkel reprezentálják azt az irányt, amelyből a minimális pontszámmal rendelkező jármű érkezett.
ocit1, *ocit2*, *ocit3 (boolean)*: a detektorok lekérdezéséhez szükségesek.
beillesztes (boolean): akkor kap igaz értéket, ha fázisbeillesztés történik.
Utóbbi változó szükségessége a fázisok bemutatásánál fog kiderülni.

4.2.3.3 Init.java

A gyári programban nem szerepelnek OCIT-protokollon kommunikáló detektorok, ezért őket létre kellett hozni. Nevük *d01, d02, d03.* Az `initialisiereProgs()` metódusban be kellett szúrni egy a buszfüggő program jellemzőit tartalmazó sort. A program abszolúte buszoktól függése miatt létre kellett hozni több új fázisátmenetet, hiszen egy fázisból az összes járműfázisba át kell tudni kapcsolni: *phUeb13; phUeb14; phUeb21; phUeb31; phUeb32; phUeb42; phUeb43.* A fázisok inicializálása nem változott, mert ebben a programban is négy fázis van, és az inicializálás során a fázisok programja nem fut le.

Az OCIT-detektorok kezeléséhez kimeneteket kell definiálni. (*out1, out2, out3*) Az IO-24 kártya kimeneteihez is hozzá kell rendelni a detektorokat a helyes működéshez.

4.2.3.4 Busprog.java

Ez a programrész tartalmazza a buszfüggő program stratégiájának keretrendszerét. A `programmFunktion` metódusban található a bejelentkezések kezelése. Mindhárom OCIT-detektorról egyforma elv szerint „veszi le” a gép az adatokat. Működését az 1-es jelű detektorhoz tartozó utasítássorozaton mutatom be.

Az utasítások csak akkor futnak le, ha az OCIT1 detektor foglaltságot jelez (`eingangGesetzt()` metódus). A program stratégiájának bemutatásánál leírt elv szerint a gép pontszámot képez a bejelentkezett autóbusz adataiból akkor, ha sem az utasszám, sem a késés nem nulla értékű. Ez a feltétel azért szükséges, mert üres autóbusz és menetrendet tartó autóbusz nem kap előnyt. Ezután egy ciklussal meg kell vizsgálni, hogy a bejelentkezéseket tároló tömbben melyik az első üres oszlop: ide kell beírni a bejelentkezés adatait. Ha a vizsgálat nem történne meg, felülíródhatnának korábbi bejelentkezések. Végül az adatok bekerülnek a *bejelentkezesek[][]* mátrixba.

Még ebben programrészben gondoskodni kell arról, hogy a *bejelentkezesek[][]* mátrixban mindig legyen hely az új bejelentkezések számára. Ha mind a húsz helyre került már bejelentkezés, akkor a gép törli az első tíz adatait, a második tíz adatait pedig a mátrix elejére mozgatja. Mivel tíz bejelentkezés minden törlés után rendelkezésre áll, mindig tud miből választani a program.

4.2.3.5 MinSearch.java

Ez egy általam létrehozott class, amely a bejelentkezések közül kiválasztja a legkisebb pontszámmal rendelkezőt. Egyetlen metódusa van: a `getBejelentkezes()`.

Elvileg a definiált *vizsgalt*`[]` tömbre nem lenne szükség, hiszen a minimumkeresési eljárás a bejelentkezéseket tartalmazó *bejelentkezesek[][]* mátrix megfelelő sorában is végrehajtható lenne. A különválasztás oka a programrészek futási időközében kere-

sendő. A *bejelentkezesek[][]* mátrix a BusProg.java-ban kap új értékeket, ami folyamatosan fut a gépben, így folyamatosan módosulhat. A *vizsgalt*[] tömb viszont csak akkor kap értékeket, ha a getBejelentkezes() metódus a program más részeiből meghívódik. A minimumkeresés, valamint a feltételek vizsgálata nem nulla idő alatt megy végbe, és ezalatt az időtartam alatt lehetséges az, hogy bejelentkezik egy újabb autóbusz. Ha a mátrixban közvetlenül keresnénk minimumot, a vezérlés nem biztos, hogy elviselné a mátrix keresés közbeni módosítását, és lefagyhat a program. Mivel a jelzőlámpás forgalomirányítás biztonságorientált rendszer, minimálisra kell csökkenteni a hibalehetőségeket, így az esetleges hibák helyett inkább definiáltam egy újabb tömböt, és megoldottam a változók közti adatátadást.

Először a *vizsgalt*[] vektor feltöltődik csupa 9999 értékkel. Ilyen magas pontszám biztosan nem fordul elő, így elkülöníthetők lesznek azok a helyek, ahová kerül pontszám, ill. ahová nem. A vezérlés hátulról előre haladva megvizsgálja, hogy a *bejelentkezesek[][]* mátrix [6] indexű sorában hol nincs egyes. Ahol van, az a bejelentkezés már kapott előnyt, így már nem kell vele a továbbiakban foglalkozni. Azon bejelentkezések pontszáma, ahol az utolsó sorban 0 van és az utolsó előnyt kapott bejelentkezés után keletkeztek, a *vizsgalt*[] vektorba kerülnek. Ezt egy **while** ciklus valósítja meg. Ebben a gép klasszikus minimumkeresési eljárással megkeresi a legkisebb értéket, és eltárolja a minimum sorindexét. Ha a minimum 9999, akkor a vektor üres, így nem jelentkezett be autóbusz. Az utolsó utasítássorozat csak akkor hajtódik végre, ha nem üres a *vizsgalt*[] vektor. A gép azt vizsgálja, hogy a minimum pontszámmal rendelkező bejelentkezés melyik irányból jött, és ezalapján teszi igazzá a (*k1ben, k2ben, k3ban*) változók egyikét.

A MinSearch.java a fázisokban van meghívva a getBejelentkezes() metódus segítségével. A Java alapú forgalomtechnika bemutatásánál szó volt róla, hogy az aktuális fázis programkódja fél másodpercenként fut le. Ebből adódóan a MinSearch.java is 0,5s-onként lefut, és megad egy előnyben részesítendő irányt. Ha egyéb kiegészítő feltételek nélkül csak ezen három logikai változó alapján döntenénk egy előny megadásáról, akár fél szekundumonként változhatna az előnyben részesített irány. Ezt kiküszöbölendő a fázisok változtatásához nem elég az egyik logikai változó igaz állapota, hanem a fázis minimális zöldidejének le kell telnie. A két feltétel ÉS kapcsolattal van beépítve a programba. Így a minimális zöld letelte előtt nem lehet beavatkozni a program futásába. A minimális zöld letelte után a meghívott getBejelentkezes() metódus által megadott minimális pontszámmal rendelkező irány valóban kaphat előnyt, mert a feltételek megengedik.

4.2.3.6 Phase_.java

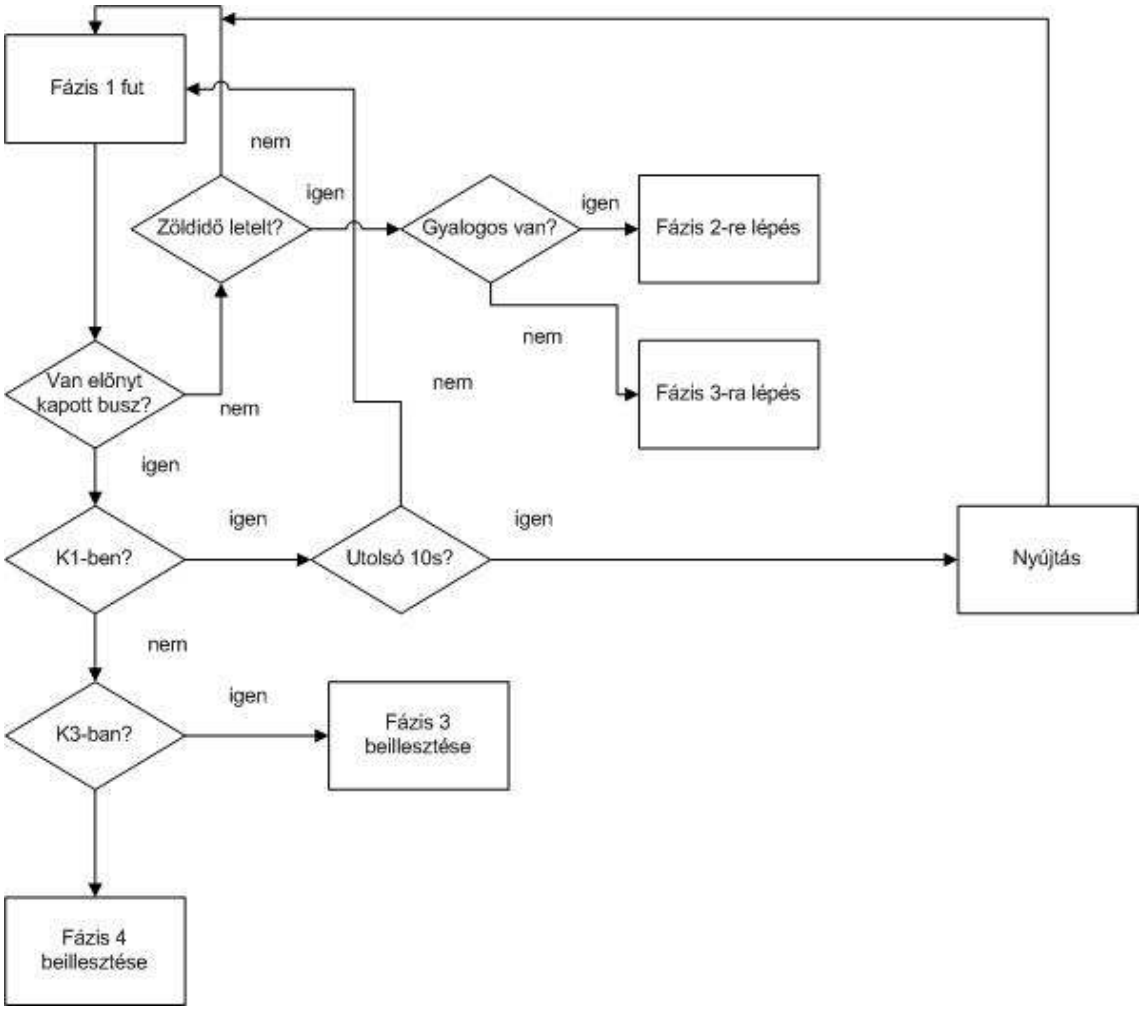
Egy-egy fázis futásakor végrehajtandó utasításokat tartalmazza. Mindegyik fázisban megadtam, hogy ott milyen intézkedéseket lehet tenni. Ezek végrehajtási feltételét részleteztem a MinSearch.java bekezdésében. Minden fázisban hasonló intézkedéseket lehet tenni, kivéve a 2. fázist, amely a gyalogos fázis. A hasonlóság miatt itt is csak az elsőt mutatom be részletesen.

A fázisnyújtás megvalósításának alapötlete a `getPhasenSek()` függvény és a `setPhasenSek()` eljárás használatán alapul. Az elsővel lekérdezhető, hogy az éppen futó fázis hány másodperce aktív. A másodikkal ezt az időt lehet módosítani. A nyújtás feltétele a 3.2.1 fejezet elméleti elvei alapján megérthető. Elve a következő: a zöld vége előtti utolsó 10s-ban lehet zöld nyújtást kezdeményezni, ugyanis ekkor lehetséges az, hogy a jármű a bejelentkezési ponttól nem ér el a csomópontig a még rendelkezésre álló zöldidő alatt. A fázis zöldideje az A-dik másodpercben van, mely teljesíti a fenti feltételt, nyújtás 10s hosszúságú. Ha a fázis zöldidejét 10-A-val állítjuk vissza, akkor a 10s eltelte után a busz éppen elhagyta a csomópontot. Pl. A=22. Ekkor 25-22=3s van vissza. 10-3=7s-mal kell visszaállítani a fázis idejét. 22-6=15, és ekkor 15+10s nyújtás=25, és innentől „rendben” folytatódik tovább a ciklus.

Fázisbeillesztésnél az alapelv a következőképpen magyarázható el. A jelenlegi fázis legyen az 1-es jelű. Ekkor egyetlen autóbusz érkezik a K3 irányból, így neki előnyt kell adni, ami a 3. fázis bekapcsolását jelenti. A beillesztett fázis mindig csak 10s (K2 irányban 8s) lehet, ennek betartásához szükséges a *`beillesztes`* logikai változó. A 10 másodperc elteltével a fázis befejeződik. Jelen programban a beillesztett fázis nem nyújtható. (Kevés számú módosítással alkalmassá tehető a program ilyen működésre is.) A következő fázis a 4. lenne, azonban beillesztésről lévén szó, az elvhez ragaszkodva a 2-es fázist kapcsolja be a program, ha van bejelentkezett gyalogos. Ha nincs, akkor ismételten a 3. fázist adja ki oly módon, hogy a már eltelt 10 másodpercet beleszámítja a zöldidőbe, így csak 15s-ig marad kint ezen felül a zöld jelzés ezen irányban. Ez praktikusan úgy oldható meg, hogy semmilyen utasítást nem kell végrehajtani. Ha nem pont a gyalogos fázis előtt illesztünk be másik fázist, akkor nincs ennyi variációs lehetőség. Az *`elozofazis`* változó segítségével tárolható az, hogy mely fázist elhagyva kapcsoltunk be egy másikat, így az elhagyott után az elvi jelzéstervben következő bekapcsolható. A fenti példában ez a 2-es fázis.

A fázis kezdetének előrehozása oldható meg legkönnyebben. Itt is figyelni kell, hogy letelt-e már a minimális zöld, ha igen, akkor egy egyszerű azonnali fázisátmenet meghívással lerövidíthető az aktuális zöldidő, és előbb bekapcsolható a következő.

A fázis mindenképpen úgy fejeződik be, hogy valamilyen feltételnek megfelelő fázisátmenetet hív meg a vezérlés.



21. ábra Az 1. fázis logikai összefüggései

A 21. ábrán látható az 1. fázis futásának logikai képe.

4.2.3.7 PhUeb__.java

Az Actros a közbenső idők alatt elvégzendő feladatokat ún. fázisátmenetekben hajtja végre. Programozás szempontjából ezek is fázisoknak tekinthetők, hiszen a `Phase` osztályból vannak származtatva. Például egy fix programos háromfázisú csomópont vezérlésekor a gép hat fázist kezel, mert van három fázisátmenet is.

A fázisátmenetben meg kell adni, hogy az előző fázis jelzőit azonnal tilosra kell állítani. A berendezés az átmeneti idő kiadását automatikusan végzi. A következő fázis jelzőit pedig a közbenső idő elteltével kell zöldre állítani. (az előkészítő idő kiadása is automatikus) Ezek a műveletek a `setSg` eljárással hajthatók végre. Ezután a fázisátmenet befejezési feltétele következik, amelyben a vezérlés azt vizsgálja, hogy a jelzőfejek a megfelelő jelzéseket mutatják-e, ill. letelt-e a közbenső idő. Csak utóbbi elteltével lehet a következő fázist bekapcsolni. Az autóbuszok bejelentkezésekor miatt mindegyik fázisból

szinte mindegyikbe át lehet lépni, így újabb fázisátmenetek létrehozása vált szükségessé: az 1-4; 2-1; 2-4; 3-1; 3-2; 4-2; 4-3 fázisok között.

Emellett további beállítások szükségesek a MyMainFkt.java és a MyPostMainFkt.java részekben, ahol az OCIT-detektorok pontos működését szükséges leírni.

4.2.4 A program továbbfejlesztési lehetőségei

1. A megvalósított program „végtelen” ciklusidejűnek is tekinthető, ez magában hordozza a változtatások igen nagy tárházát. Készítető fix ciklusidejű program is, ekkor azonban az előnybiztosítási intézkedések megtétele előtt ellenőrizni kell, hogy rendelkezésre áll-e megfelelő időtartam a végrehajtáshoz a kötelezően betartandó feltételek betartása mellett. (közbenső idők, minimális zöldidők megtartása) A ciklusban előrehaladva egyre kevesebb lehetőség kínálkozik a változtatásra, ez pedig eltérést eredményezhet az optimális megoldástól.

2. A program alapvetően egy fix fázisidejű jelzésterven alapul. Az ilyen jelzésprogram a XXI. században már nem tekinthető túl innovatívnak. Célszerű lenne egy alapvetően optimalizálásra alapuló forgalomfüggő logikát megvalósítani. Ehhez azonban megfelelő fejlesztőszoftver (pl. Matlab) szükséges, amely képes a bemenő forgalmi adatok alapján valós időben új fázisidőtervet alkotni. Valóságban működő rendszernél szükség lehet egy újonnan beépített célhardverre is, amelyben a helyszínen mehet végbe az optimalizációs eljárás lefutása. Ha ez nem állna rendelkezésre, a bemenő adatokat továbbítani kellene egy központba, és a kimenő adatokat (zöldidők kezdete, hossza) pedig vissza kellene küldeni a berendezéseknek. A program az előnybiztosítás miatt nem lenne optimális, hiszen az érkező autóbuszok „elhangolják azt.

3. A szakdolgozatban szereplő csomópont és jelzéstervének működése demonstrálható megfelelő szoftverek segítségével. (pl. Vissim) A Vissimben a csomópont felépítésén kívül szükség lenne a vezérlés COM-felületen történő átadására is, mivel a szoftver önmagától nem képes forgalomfüggő irányítás megvalósítására. A 2. pontban vázolt optimalizált forgalomfüggő jelzésterv is bemutatható Vissimben, ekkor azonban a Matlabot és a Vissimet össze kell kötni a COM-felületen keresztül. (A témában lásd a [7] jelű cikket.)

4. A forgalomirányító berendezés virtuálisan „áthelyezhető” egy másik csomópontba, azaz meg lehet változtatni a csomóponti geometriát. Ez szinte a teljes program megváltoztatásával jár, ugyanis további jelzőfejeket kell létrehozni, új alapparamétereket kell definiálni, stb.

5 Elért eredmények

5.1 Futtatás, szimuláció

Megvizsgálható, hogy az előnybiztosítási intézkedések hatására hogyan változik az egyes csomóponti irányokban a maximálisan várakozással eltöltött idő. A 3. táblázat alapján megállapítható, hogy a zöld előrehozással rövidül legjobban várakozási idő a következő szabad jelzésre. Ez érthető is, hiszen ezzel az intézkedéssel tulajdonképpen csökkent a csomópont „ciklusideje".A fázisbeillesztés produkálja a legnagyobb értékeket, mert azt feltételeztem, hogy a beillesztést megelőző fázis zöldideje nem rövid le a beillesztéssel. Mivel a beillesztés a minimális zöld letelte után bármikor végrehajtható, nem biztos, hogy ilyen magas érték adódik a futtatáskor.

Irányok	**normál**	**K1 nyújtása**	**K2 nyújtása**	**K3 nyújtása**	**K1 előrehozása**	**K2 előrehozása**	**K3 előrehozása**	**a) beillesztés**	**b) beillesztés**	**c) beillesztés**	**d) beillesztés**	**e) beillesztés**	**f) beillesztés**
K1	78	0	86	88	71	63	63	103	100	0	106	0	103
K2	88	98	0	98	81	73	73	113	0	113	0	113	113
K3	78	98	86	0	71	63	63	0	100	103	106	103	0

3. Táblázat maximális várakozási idők

A betűkkel jelölt fázisbeillesztések a következők:

Beillesztés jele	Beillesztés sorrendje (aláhúzott: beillesztett fázis)
a)	K1 → <u>K3</u> → GY
b)	K1 → <u>K2</u> → GY
c)	GY → <u>K1</u> → K3
d)	GY → <u>K2</u> → K3
e)	K3 → <u>K1</u> → K2
f)	K2 → <u>K3</u> → K1

A következőkben a program futásából származó részlelek láthatók, melyeket a Microsoft vállalat HyperTerminal programjával lehet az ACTROS berendezés soros portján keresztül kinyerni. Sajnos a képernyőrészletek nem túl nagyok, mert a HyperTerminal csak kb 25 sort tud normál tördeléssel megjeleníteni, utána már szétesnek a sorok és az információk.

```
 00010741: jniVT           : 1.fazis fut; fazis ideje: 22
 00010771: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010771: jniVT           : 1.fazis fut; fazis ideje: 23
 00010801: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010801: jniVT           : 1.fazis fut; fazis ideje: 23
 00010831: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010831: jniVT           : 1.fazis fut; fazis ideje: 24
 00010861: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010861: jniVT           : 1.fazis fut; fazis ideje: 24
 00010891: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010891: jniVT           : 1.fazis fut; fazis ideje: 25
 00010921: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010921: jniVT           : 1.fazis fut; fazis ideje: 25
 00010951: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00010951: jniVT           : 1.fazis fut; fazis ideje: 26
 00011311: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00011311: jniVT           : 3.fazis fut; fazis ideje: 0
 00011341: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00011341: jniVT           : 3.fazis fut; fazis ideje: 0
 00011371: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00011371: jniVT           : 3.fazis fut; fazis ideje: 1
 00011401: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00011401: jniVT           : 3.fazis fut; fazis ideje: 1
```

22. ábra A buszos program futása

A 22. ábrán egy 1-3. fázis közti átmenet látható. Nem volt sem gyalogos bejelentkezés, sem autóbusz bejelentkezés.

```
 00015661: jniVT           : 1.fazis fut; fazis ideje: 16
 00015691: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00015691: jniVT           : 1.fazis fut; fazis ideje: 17
 00015721: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
 00015721: jniVT           : 1.fazis fut; fazis ideje: 17
 00015751: jniVT           : CiklusvÃ¨ltozÃł: 0
 00015751: jniVT           : ellenorzes: 0
 00015753: jniVT           : 3-as detektoron bejelentkezes; pontszam: 0.03069112
8 Bejelentkezesek matrixban elfoglalt oszlop szama: 0
 00015753: jniVT           : MS lef; Ir: K1: false; K2: false; K3: true
 00015753: jniVT           : Fazis 3 beillesztese
 00015781: jniVT           : CiklusvÃ¨ltozÃł: 0
 00015781: jniVT           : CiklusvÃ¨ltozÃł: 1
 00015781: jniVT           : ellenorzes: 1
 00015781: jniVT           : 3-as detektoron bejelentkezes; pontszam: 0.05058703
 Bejelentkezesek matrixban elfoglalt oszlop szama: 1
```

23. ábra Fázisbeillesztés a program futásakor

A 23. ábrán látható részlet szerint az 1. fázis futása közben bejelentkezett egy autóbusz a K3 irányból, ekkor a gép –mivel a fázis minimális zöldidején túljutott- beillesztette a 3. fázist. Ezek után ismét érkezett egy bejelentkezés a K3 irányból, de ennek hatására már semmit nem kell tenni, hiszen már elindult a 3. fázis beillesztése.

```
00022381: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022381: jniVT           : 3.fazis fut; fazis ideje: 17
00022411: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022411: jniVT           : 3.fazis fut; fazis ideje: 18
00022441: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022441: jniVT           : 3.fazis fut; fazis ideje: 18
00022471: jniVT           : CiklusvÃ¯ltozÃł: 0
00022471: jniVT           : CiklusvÃ¯ltozÃł: 1
00022471: jniVT           : CiklusvÃ¯ltozÃł: 2
00022471: jniVT           : CiklusvÃ¯ltozÃł: 3
00022471: jniVT           : CiklusvÃ¯ltozÃł: 4
00022471: jniVT           : CiklusvÃ¯ltozÃł: 5
00022471: jniVT           : ellenorzes: 5
00022471: jniVT           : 2-es detektoron bejelentkezes; pontszam: 0.01934196
Bejelentkezesek matrixban elfoglalt oszlop szama: 5
00022471: jniVT           : MS lef; Ir: K1: false; K2: true; K3: false
00022471: jniVT           : Fazis 4 elorehozas
00022891: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022891: jniVT           : 4.fazis fut; fazis ideje: 0
00022921: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022921: jniVT           : 4.fazis fut; fazis ideje: 0
00022951: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00022951: jniVT           : 4.fazis fut; fazis ideje: 1
```

24. ábra Fázis előrehozása a program futásakor

A 24. ábrán látható programállásban a 3. fázis futott éppen, amikor a K2 irányból autóbusz érkezett. A közbenső idő beiktatása után megtörtént a 4. fázis előrehozása.

```
00018871: jniVT           : 2-es detektoron bejelentkezes; pontszam: 0.03133712
Bejelentkezesek matrixban elfoglalt oszlop szama: 3
00018871: jniVT           : MS lef; Ir: K1: false; K2: true; K3: false
00018871: jniVT           : 4.fazis fut; fazis ideje: 7
00018901: jniVT           : MS lef; Ir: K1: false; K2: true; K3: false
00018901: jniVT           : Fazis 4 nyujtas
00018901: jniVT           : 4.fazis fut; fazis ideje: 7
00018931: jniVT           : MS lef; Ir: K1: false; K2: false; K3: false
00018931: jniVT           : 4.fazis fut; fazis ideje: 7
```

25. ábra Zöld nyújtása a program futásakor

A 25. ábra egy zöld nyújtást mutat a 4. fázisban. A bejelentkezés a zöldidő első nyújtható másodpercében érkezett, ezért van minden időértéknél 7 másodperc.

```
00024751: jniVT             : 1.fazis fut; fazis ideje: 8
00024781: jniVT             : MS lef; Ir: K1: false; K2: true; K3: false
00024781: jniVT             : 1.fazis fut; fazis ideje: 8
00024811: jniVT             : MS lef; Ir: K1: false; K2: true; K3: false
00024811: jniVT             : 1.fazis fut; fazis ideje: 9
00024841: jniVT             : MS lef; Ir: K1: false; K2: true; K3: false
00024841: jniVT             : 1.fazis fut; fazis ideje: 9
00024871: jniVT             : MS lef; Ir: K1: false; K2: true; K3: false
00024871: jniVT             : Fazis 4 beillesztes
00025291: jniVT             : MS lef; Ir: K1: false; K2: false; K3: false
00025291: jniVT             : 4.fazis fut; fazis ideje: 0
```

26. ábra A minimális zöld hatása

A minimális zöldidő „védő” hatását mutatja a 26. ábra. A bejelentkezés és a pontszám kiértékelése már korábban megtörtént, de ekkor a minimális zöldidőn még belül volt a program. Az ábrán látható időben ért el az 1. fázis minimális zöldjének végéhez, így megtörténhetett a 4. fázis beillesztése.

5.2 A csomópont externális költségeinek monetarizálása [3]

A közúti közlekedés költségszerkezete igen szerteágazó, többek között externális költségelemeket is tartalmaz. (A közúti közlekedésben externális [külső] költségnek nevezik a közlekedés direkt költségei között meg nem jelenő ráfordításokat: a légszennyezés, zajhatás, baleset, késés költségeit.) Ezeket a költségelemeket nagyon sokáig nem vették figyelembe, így viselőjük részére nem kerültek megtérítésre. A közlekedési teljesítmények növekedésével azonban egyre magasabbak lesznek az externális költségek, így egyre szükségesebb ezek pontos megállapítása és beépítése a pénzügyi rendszerbe.

A kibocsátott szennyezőanyagok mennyisége valamelyest csökkenthető, ha a közlekedés „folyamatos". Ez azt jelenti, hogy a lehető legminimálisabb számú gyorsítással és lassítással megoldható a cél elérése egy adott útvonalon. Ez vonali jelzőlámpás forgalomirányítással oldható meg, amely praktikusan a „zöldhullám" kialakítását jelenti. Azért célszerű erre törekedni, mert a járművek fogyasztása gyorsításkor a legnagyobb, egyenletes haladáskor a legkisebb. A szennyezőanyag-csökkentés másik kézenfekvő módja a modal split javítása a közforgalmú közlekedés javára. A fenti két módszert ötvözi az autóbuszok csomóponti előnybiztosítása.

5.2.1 Kiindulás

A járművek károsanyag-kibocsátásának monetarizálásához szükség van a csomópontban egy ciklus alatt irányonként feltartóztatott járművek számára. A következőkben ismertetett módszerrel egy másik jellemző mennyiséget, az egy járműre jutó feltartóztatási időt is ki lehet számítani. Ezen számítási metódus segítségével értékelhető egy jelzőlámpával irányított csomópont forgalomlefolyása is.

A számítási módszer **bemenő paraméterei** a következők:

- érkező járművek száma óránként az összes irányban (Q_{be})
- a csomópontból kihaladó járművek száma: jelen esetben a maximális, $\mu = 0{,}5\frac{J}{s}$ értékkel számoltam
- a csomópont jelzésterve (a számítást a 17. ábrán látható jelzésterv alapján végeztem el.)

Megjegyzés: egy csomópont jelzéstervének kialakítása és a bemenő forgalomnagyságok között összefüggés van, ugyanis a program periódusidejét és a jelzéstervben szereplő zöldidőket a bemenő forgalomnagyságok alapján állapítják meg. Emiatt egy jelzésterv „felső kapacitással" rendelkezik, mert ha a tervezés során figyelembe vett forgalomnál jelentősen nagyobb igények jelennek meg a csomópontban, akkor a jelzésterv nem képes ezek levezetésére, torlódás alakul ki.

1. lépés: forgalmi (ρ) intenzitás számítása:

Ehhez szükség van arra, hogy az érkező járműszámot (Q_{be}) $\frac{J}{s}$ mértékegységben ismerjük (1):

(1) $$\lambda = \frac{Q_{be}}{3600}$$

.

A forgalmi intenzitás (2) mértékegység nélküli viszonyszám.

(2) $$\rho = \frac{\lambda}{\mu}$$

2. lépés: a sor kiürülési idejének - t_c (s) - meghatározása (3):

A jelzéstervből meg kell állapítani, hogy mekkora a piros idő - r (s) - egy-egy irányban, amely felhasználásával:

(3) $$t_c = \frac{\rho \cdot r}{1-\rho}$$

3. lépés: a csomóponti időarány - P_q (-) - a kiürülési idő, valamint a pirosidő és a periódusidő aránya (4):

A csomópont ciklusideje C (s). A következő ezt képletben felhasználva kapjuk:

(4) $$P_q = \frac{r + t_c}{C}$$

amely az (5) összefüggés szerint megegyezik a megállított járművek arányával (P_s):

(5) $$P_s = \frac{\lambda \cdot (r + t_c)}{\lambda \cdot (r + g)} = \frac{r + t_c}{C} = P_q$$

ahol g az effektív zöldidő, ami magában foglalja az előkészítő időt és az átmeneti időt is.

4. lépés: a megállt járművek száma periódusonként (6): N (db)

(6) $$N = \lambda \cdot C \cdot P_s$$

5. lépés: egy járműre jutó késés – d_{avg} (s) - (8) alapján:

(8) $$d_{avg} = \frac{r^2}{2 \cdot C \cdot (1-\rho)}$$

5.2.2 A metódus kibővítése

A eddigiekben ismertetett lépéseket **ki kellett egészíteni** a társadalmi költségek meghatározásához. A további számításokhoz szükség van a „járműszekundum” nevű képzett mennyiségre. Nevét a mértékegysége alapján ($db \cdot s$) kapta. Kiszámítása a (9) képlet szerint történik.

(9) $$JS = N * d_{avg}$$

Meg kell határozni az egyes járműfajták tüzelőanyag-fogyasztását is. A piros lámpánál állva a járművek motorja alapjáraton üzemel. Ekkor egy személygépkocsi átlagosan $F_{car} = 0{,}8\frac{l}{h}$ tüzelőanyagot fogyaszt. Ez az érték a benzines és a dieselüzemű gépkocsikra egyaránt igaz. Egy autóbusz $F_{bus} = 1{,}8\frac{l}{h}$ gázolajat fogyaszt alapjáraton. A mértékegységek konzisztenciája érdekében a képletekben $\frac{l}{s}$ egységre átszámítva kell ezeket az értékeket felhasználni.

A járművek **klimatikus költségét** leggyakrabban a szén-dioxid kibocsátás alapján szokták meghatározni. Az erre vonatkozó alapértékek igen nagy szórást mutatnak, ugyanis különféle kutatások eredményeként 14 és 200 $\frac{Euro}{t\_CO_2}$ közti értékek állnak rendelkezésre a klimatikus költségek becslésére. [3] Dolgozatomban három különböző értékkel számoltam. [3] szerint Magyarországon $C_{CO_2} = 5\frac{Euro}{t\_CO_2}$ értékkel célszerű számolni, míg az Európai Szénkereskedelmi rendszerben a széndioxid tonnánkénti ára $C_{CO_2} = 12\frac{Euro}{t\_CO_2}$-os értéket alkalmazták (2009.10.16.), amely a széndioxid-kereskedelmi piacokon elfogadott ár. Nemzetközi összehasonlításban a $C_{CO_2} = 20\frac{Euro}{t\_CO_2}$ érték a megszokott.

Tökéletes égést feltételezve meghatározható, hogy 1 liter benzinből kb. 2,25kg, 1 liter gázolajból pedig 2,5kg CO_2 keletkezik. ($m_{CO_{2benzin}} = 2{,}25\frac{kg}{l}$; $m_{CO_{2diesel}} = 2{,}5\frac{kg}{l}$) Magyar viszonyok között feltételezhető, hogy a forgalomban lévő személyautók 60% benzinüzemű, a maradék 40% pedig dieselolajjal működik. A személygépkocsik összesített CO_2 kibocsátását ezen arány figyelembe vételével határoztam meg.(10)

(10) $$m_{CO_{2car}} = 0{,}6 \cdot 2{,}25 + 0{,}4 \cdot 2{,}5 = 2{,}35\frac{kg}{l}$$

A csomópont társadalmi költségének növelő tényezői:

Személygépkocsi klimatikus költségének - $C_{cl_{car}}$ ($\frac{Euro}{periódus}$) - meghatározása:

(11) $$C_{cl_{car}} = JS \cdot F_{szgk} \cdot m_{CO_{2_{car}}} \cdot C_{CO_2}$$

A klimatikus költség a (10) képletben kiszámított „járműszekundumnak", a személygépkocsik fogyasztásának, a fogyasztásból eredő CO_2 kibocsátás mennyiségének és a CO_2-kibocsátás költségének szorzata. Ezek alkotják a (11) képletet.

A számítás során azt feltételeztem, hogy minden sávban egyszerre érkezik 1-1 autóbusz, így az előnyt nem kapott irányokban lévő autóbuszok is kénytelenek többletkésést elszenvedni. Ezzel a feltételezéssel kiegyenlítettebbé válik a számítás.

Az autóbuszok klimatikus költsége a megfelelő értékek kicserélésével a (11) képlet módosításával számítható ki. (12) Mivel egy sávban csak egy autóbusz érkezését feltételeztem, ezért nem a JS értékkel, hanem az egy járműre jutó feltartóztatással kell számolni (d_{avg}). Ezen érték nulla minden előnyt kapott sávban, hiszen az előnyt kapott autóbusz nem áll a sorban.

(12) $$C_{cl_{bus}} = d_{avg} \cdot F_{busz} \cdot m_{CO_{2_{diesel}}} \cdot C_{CO_2}$$

A személygépjárművek késésből adódó társadalmi költségének meghatározásához (13) szükség van az egy órai **késésből származó externális költség** értékére. Jelen esetben $1781\frac{Ft}{\frac{késett\_óra}{személy}}$ mennyiséggel[4] számolva, amely - a 2009.10.15.-i MNB árfolyammal számolva – $C_{d_{dev}} = 6{,}67\frac{Euro}{\frac{késett\_óra}{személy}}$-nek felel meg. A számításban ezen értéket is másodperc időalapra kell vonatkoztatni. A személyautók kihasználása igen csekély, mindössze $N_{pass_{car}} = 1{,}1\frac{utas}{jármű}$.

(13) $$C_{d_{car}} = JS \cdot N_{pass_{car}} \cdot C_{d_{dev}}$$

A sorban álló autóbuszok késésből adódó költségét átlagos utasterheltségű járműre adtam meg. A járművek befogadóképessége igen széles skálán mozoghat a 20-tól egészen a 190-ig. Mivel a számítást egy városi csomópont adataival végeztem el, így figyelembe vehető, hogy a csomóponton helyi és helyközi kivitelű szóló és csuklós autóbuszok

[4] HEATCO (Developing Harmonised European Approaches for Transport Costing and Project Assessment - EU 6. Kutatási Keretprogram által finanszírozott projekt, magyar részről a BME Közlekedésgazdasági Tanszék vesz részt a kutatásban, témavezető: Dr. Tánczos Lászlóné, egyetemi tanár): HEATCO WP3: Current practice in project appraisal in Europe – Analysis of country reports (Deliverable 1). EU project funded by the EC – DG TREN, 6th Framework Programme. – Odgaard, T., Kelly, C., Laird, J. (2005),

egyaránt áthaladnak. Ezek férőhely-kapacitásbeli kapacitása igen nagy szórást mutat, ezért egy 100 fős befogadóképesség-egyenértéket vettem fel. A járművek napközbeni kihasználtsága is ingadozik, ezért 70%-ra választottam a járművek eme mutatót. Az előbbi két értékből következik, hogy egy autóbuszra $N_{pass_{bus}} = 70$ utas jut. Ennek segítségével felírható a (14) egyenlet:

$$C_{d_{bus}} = d_{avg} \cdot N_{pass_{bus}} \cdot C_{d_{dev}} \quad (14)$$

A következőkben bemutatott tényezők a számítás során **negatív előjellel** veendők figyelembe, ugyanis megtakarításnak tekinthetők. Abból adódnak, hogy az előnyben részesített járműnek nem kell megvárni a következő ciklus zöldidejét, hanem a zöldidőnyújtásnak köszönhetően még az aktuális ciklusban elhagyhatják a csomópontot. A sorban nem álló autóbuszok klimatikus költsége ($C_{ac_{bus}}$) a (15) képlet szerint alakul.

$$C_{acl_{bus}} = r_a \cdot F_{bus} \cdot m_{CO_{2_{diesel}}} \cdot C_{CO_2} \quad (15)$$

ahol r_a a „megspórolt” pirosidő, amit a járműnek nem kellett kivárnia. Ez ha zöld nyújtáson kívül más előnybiztosítási intézkedést nem foganatosíthatunk, akkor a teljes pirosidő, ha fázisbeillesztés és zöld előrehozás is lehetséges, akkor a közbenső idők és a következő fázis minimális zöldjének összege.

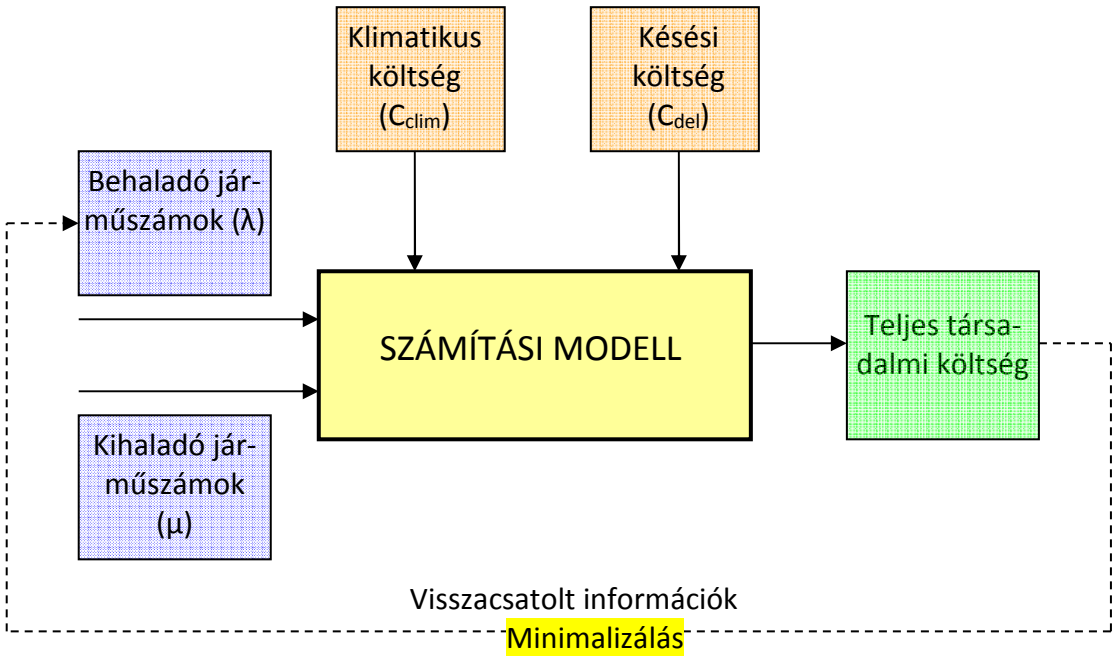
Az előnyt kapott autóbusz „nem késéséből” adódó költségtömeg ($C_{ad_{bus}}$) a (16) összefüggéssel határozható meg.

$$C_{ad_{bus=}} = C_{d_{dev}} \cdot N_{pass_{busz}} \cdot r_a \quad (16)$$

A **teljes társadalmi költség** a (11) – (16) összefüggések eredményéből származtatható (17):

$$C_{s_{sum}} = C_{cl_{car}} + C_{cl_{bus}} + C_{d_{car}} + C_{d_{bus}} - C_{acl_{bus}} - C_{ad_{bus}} \quad (17)$$

A számítási modell működése szemléltethető a 27. ábrán látható blokkvázlattal is. Bemenő paraméternek tekinthetők a be- és kihaladó járműszámok, valamint a társadalmi költség alapértékei. A rendszer kimenő paramétere a csomópontra meghatározott teljes társadalmi költség.



27. ábra A számítási módszer blokkvázlata

Az ábrán megjelöltem a fejlesztési lehetőségek egyikét egy szaggatott vonallal. Ezzel a kiegészítéssel a modellt visszacsatolt rendszerré sikerült tenni, így a vezérlés követelményei mellett a szabályozás követelményeinek is eleget tesz.

Jelenleg is folynak kutatások a közúti közlekedés gazdasági mutatókon is alapuló optimális irányításának kidolgozására[5]. Ebben az esetben a programalkotó üzemmódra is alkalmas forgalomirányítás nemcsak a közlekedési folyam jellemző forgalomtechnikai paramétereit veszi figyelembe, hanem a járműfolyam által generált az externális költségtömeget is tartalmazó gazdasági paramétereket is

További fejlesztési iránynak tekinthető az, hogy a modell a bemenő paraméterek változásához igazítja a csomópont jelzéstervének periódusidejét, ill. a zöldidőket. (A számításban ezek állandó értékek, így a teljesítőképességnek nemcsak fizikai felső korlátja van.)

5.2.3 A számítási metódus alkalmazása egy mintacsomópontra

A 18. ábrán látható csomópontot tekintettem a számítás alapjának, jelzésterve a 20. ábra szerinti.

[5] OTKA CNK 78186 projekt: A közúti járműforgalom modellezése és többkritériumú optimalizáláson alapuló irányítása a társadalmi és gazdasági hatékonyság figyelembe vételével (Vezető: Prof Dr. Bokor József)

A számításban az előnybiztosítási intézkedések közül a zöld nyújtást vizsgáltam meg, ugyanis ezzel spórolható meg a legtöbb várakozás. A zöld előrehozás és a fázisbeillesztés csak a minimális zöld kiadása után lehetséges, míg ez az intézkedés a zöld utolsó másodperceiben azonnal megtehető.

Három nyújtási változatot tekintettem át:

A. eset: a K3 irányban 10s nyújtás, ezt a 10s-ot a K2 irány zöldidejéből vesszük el.
B. eset: K1 irányban 10s nyújtás, K3 irányban -10s zöldidő.
C. eset: A K2 irány kap 8s többlet zöldidőt, és ezt a K1 zöldidejéből vesszük el.

A klimatikus költség, mint paraméter szerint a számított eredmények közti különbségek gyakorlatilag elhanyagolhatók, ugyanis a klimatikus költségek nagysága töredéke a késésből adódó költségeknek. (Legjobb esetben is $20\frac{Euro}{t\_CO_2}$ áll szemben $6{,}67\frac{Euro}{\frac{késett\_óra}{személy}}$-el.)

A bemenő járműszámot a K1, K2, K3 irányokban rendre 300, 150, 300 $\frac{J}{h}$ nagyságúra választva a 4. táblázatban látható számítási eredményeket kaptam:

	Társ ktg. (€)		
Intézkedés CO2 kibocsátás (€/t)	5	12	20
normál eset	14,77	14,77	14,77
A eset	1,97	1,97	1,98
csökk. az eredetihez képest (%)	**86,66**	**86,64**	**86,63**
B eset	2,05	2,05	2,06
csökk. az eredetihez képest (%)	**86,10**	**86,09**	**86,08**
C eset	-0,16	-0,16	-0,16
csökk. az eredetihez képest (%)	**101,11**	**101,10**	**101,08**

4. Táblázat Teljes társadalmi költség egy periódusra vonatkozóan a mintacsomópontban

A bemenő értékek aránya miatt az előnybiztosítás **hatása igen nagy**, 86%-kal csökkent a társadalmi költség. A (C) jelű intézkedésnél egyenesen **társadalmi haszna** van annak, hogy az autóbusz nem áll a piros lámpánál. (Ebben az esetben 85s lenne a pirosidő, azért adódott ilyen kedvező eredmény.)

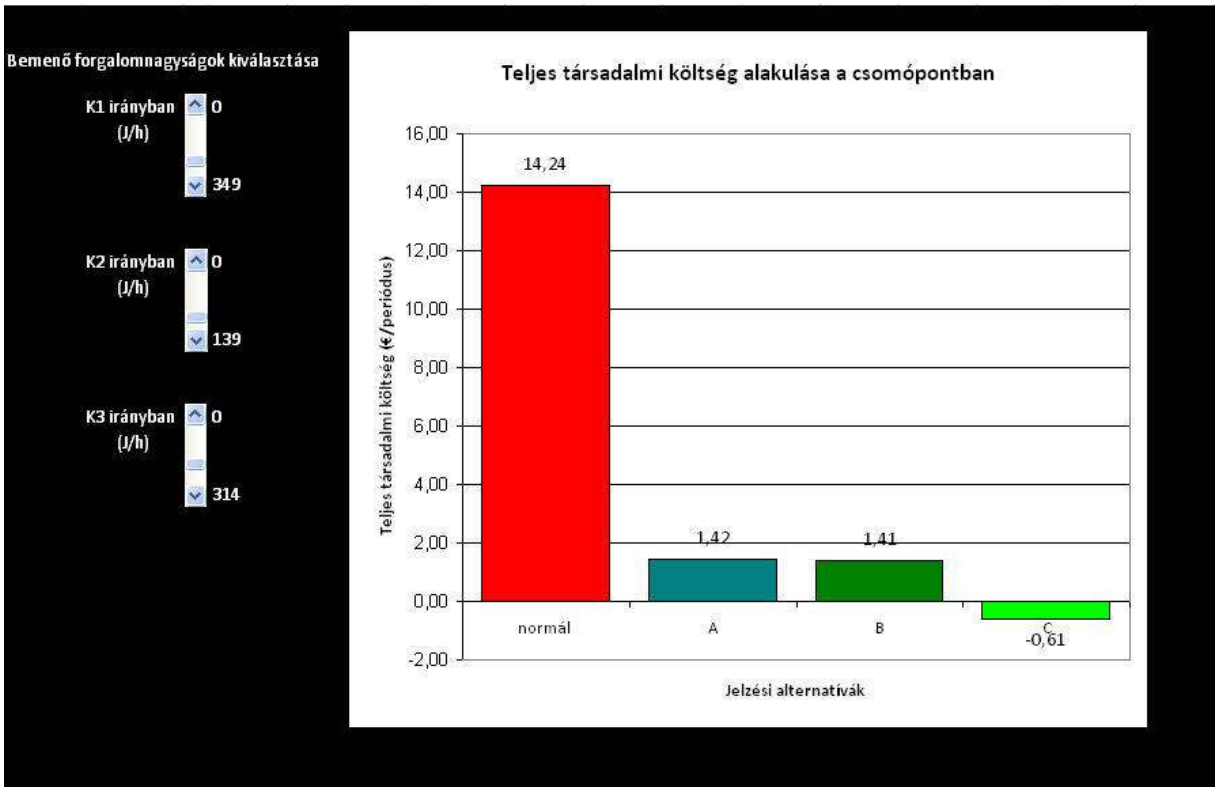
Egy másik eset is vizsgálható (5. táblázat), ebben azt feltételeztem, hogy óránként 2-2 autóbusz halad el minden irányban az előző esetnél említett forgalomnagyság mellett. Az óránként két járműben benne foglaltatik az, hogy éjszaka nincs, vagy nagyon ritka követési idejű az autóbusz-közlekedés. Ebben az esetben azt számítottam ki, hogyha az év minden napján ugyanazt az előnybiztosítási intézkedést hajtanák végre, akkor hogyan alakulna a társadalmi költség.

Társadalmi költség (€)			
Intézkedés CO2 kibocsátás (€/t)	5€/t	12 €/t	20€/t
normál eset	635354,24	636085,72	636921,70
A eset	395105,83	395791,77	396575,72
csökk. az eredetihez képest (%)	**37,81**	**37,78**	**37,74**
B eset	428013,96	428759,69	429611,95
csökk. az eredetihez képest (%)	**32,63**	**32,59**	**32,55**
C eset	613500,67	614269,66	615148,51
csökk. az eredetihez képest (%)	**3,44**	**3,43**	**3,42**

5. Táblázat Teljes társadalmi költség egy évre vonatkozóan a mintacsomópontban

A táblázat alapján megállapítható, hogy az (A) jelű intézkedés esetén csökken legjobban a társadalmi költség. Érdemes belegondolni, hogy éves szinten a kb. 400.000€ összeget mi mindenre lehetne fordítani, illetve hogyan lehetne ezt a pénzt megfizettetni a közlekedőkkel.

A bemenő járműszám változtatására elkészítettem egy demonstráló felületet az Excel programban. Ennek segítségével 10-es lépésközzel változtatható a bemenő járműszám az egyes irányokban, és a társadalmi költség aktuális értékét egy diagramban jelenik meg, egy ilyen beállítás eredménye látható a 28. ábrán.



28. ábra Képernyőrészlet a saját készítésű programból

6 Összefoglalás

6.1 Az előnybiztosítás és a közlekedéspolitika

Ha szigorúan csak környezetvédelemi vagy mérnöki szemszögből vizsgáljuk az előnybiztosítás hasznosságát, valószínűleg eltérő eredményt kapunk, mintha közlekedéspolitikai oldalról tekintünk a kérdéskörre. Látható, hogy az egyéni közlekedők egy részére nézve ezek az intézkedések többlet időráfordítást okoznak az eljutási időben, viszont összességében társadalmilag hasznos intézkedésről van szó. Jelen kutatásom célja egy közlekedéspolitikai döntéstámogató eljárás kidolgozása volt, amely az egyéni közúti közlekedés és a közforgalmú közösségi közúti közlekedés között feszülő társadalmi ellentétet próbálja feloldani a forgalomtechnika és szabályozás eszközrendszerével.

6.2 Az előnybiztosítás korlátai

A vázolt előnybiztosítási intézkedések **nem mindenhol és nem minden időben** foganatosíthatók ésszerűen. Vonali irányítás alkalmazásakor nem lehet sem a ciklusidőtől, sem a zöldidők előre definiált hosszától eltérni. Ha ez megtörténne, a „zöldhullám" nem jönne létre, így az egész irányítási cél elvész. A nagy terheltségű irányban is csak akkor lehet a zöldidőt változtatni, ha ez a csomópontok egymástól való nagy távolsága miatt nem befolyásolja jelentősen a forgalmi áramlat mozgását.

Az aktív előnybiztosító intézkedések kis és közepes forgalomnagyság mellett viszonylag kevés kihatással vannak a környező forgalomra, azonban telített forgalmi áramlatok esetén már nem igaz. Ilyen állapotban már nincs időtartalék ahhoz, hogy a kitűntetett járműveknek előnyt lehessen biztosítani.

Az intézkedések azonban korlátozottan alkalmazhatók akkor is, ha a csomópont közvetlen közelében megállóhely is van. Városi környezetben ez igen gyakran előfordul. Ilyen esetekben egy logikával meg kell vizsgálni azt, hogy a jármű a megállóhely előtt van, meg fog-e állni a megállóban, a megállóban van-e ill. elhagyta-e azt.

Torlódás esetén lenne igazán nagy jelentősége ezen intézkedéseknek, azonban a valódi segítséget buszsáv kijelölése, esetleg BRT[6]-rendszerű közlekedés bevezetése, illetve nagy forgalmú vonalakon az európai gyakorlatnak megfelelően az autóbuszok villamossal való kiváltása jelentené, de a legtöbb esetben erre nincs szabad vagy szabaddá tehető terület.

Olyan csomópontokban sem érdemes az autóbuszokat megkülönböztetni, ahol a csatlakozó irányokból igen sok menetrend szerinti járat érkezik. A buszforgalomtól függő program jelentős javulást nem hozna a várakozási időkben, viszont kiszámíthatatlanná

[6] BRT=Bus Rapid Trans: zárt pályás autóbusz-közlekedés

teszi a fázisok hosszát és sorrendjét. Ilyen esetekben célszerűbb más irányítási célt keresni. További lehetőség lenne a bejövő adatmennyiség csökkentésére ill. a változási lehetőségek csökkentésére, ha csak egy bizonyos vonal autóbuszait látnák el a bejelentkezéshez szükséges hardverelemekkel, így kevesebb alkalom kínálkozik a program módosítására. Ügyelni kell arra, hogy ez a kiválasztott vonal nagy utasforgalmat bonyolítson és/vagy igen nagy menetrendi eltéréseket produkáljon.

6.3 A dolgozat összegzése

A jelzőlámpás csomópontok csoportosításából kiderül, hogy az előnybiztosítás alkalmazásához legalább programválasztásra képes forgalomirányító berendezés kell.

Bemutattam, hogy egy csomópontban passzív és aktív intézkedések segítségével részesíthető előnyben egy autóbusz. Előbbiek hatékonysága csekély, utóbbiak viszont valóban hasznosak. Alapvetői fajtái a zöldidő nyújtás, a zöld újrakezdés, a zöld előrehozás ill. a fázisbeillesztés. A csomóponti előnybiztosítás, mint elv megvalósításához megfelelő és együttműködni képes hardverelemek, ill. informatikai rendszerek szükségesek.

Egy modern forgalomirányító berendezés felhasználásával konkrét programot is sikerült megvalósítani. A programírás előtt bizonyos feltételeket figyelembe kellett venni, illetve ki kellett dolgozni egy előnyadási stratégiát. A program működése során figyelembe veszi a csomópontba érkező autóbuszokat, ezek közül kiválasztja azt, amelyik a legtöbb utast szállítja, ill. a legnagyobb késéssel közlekedik. A program tulajdonképpen olyan készültségű, hogy akár a valóságban is lehetne alkalmazni. Összegyűjtöttem, milyen további fejlesztési lehetőségek kínálkoznak a programmal kapcsolatban.

A program megvalósítja a közforgalmú közlekedés előnyben részesítését, és mindezt úgy teszi, hogy az autóbuszok közlekedése folyamatosabbá válik, így csökken a tüzelőanyag-fogyasztás, így a károsanyag-kibocsátás. Emellett az utasok oldalán is számottevő eredmények keletkeznek, mivel csökken az utazási idő, a járművek a menetrendi eltérései csökkennek, így kiszámíthatóbbá válik a közlekedés.

Egy számítási módszer segítségével gazdasági alapra helyezve is megmutathatók az elképzelés előnyös tulajdonságai.

1. Melléklet: fogalommagyarázatok

Átmeneti jelzés: a szabd jelzést követő jelzés. Járműveknél $50\frac{km}{h}$-ig 3s, efelett 5s. Gyalogosoknál a szabad jelzés utolsó 5s-a villog.

Behaladási idő: A behaladási távolság (a helyzetjelző vonaltól a konfliktuspontig tart) és a behaladási sebesség hányadosa.

CAN-hálózat: A CAN-hálózat (Control Area Network) egy kétvezetékes félduplex nagysebességű soros busz. Az üzenetekben nincs megcímzett állomás, hanem tartalmát és prioritását egy azonosító írja le. Az azonosítótól függ az, hogy mely hálózati elemnek szól egy üzenet.

COM-felület: szoftverek közti kommunikációt megvalósító felület. Az egyes programokban előre definiáltak azok a paraméterek, amelyeket COM-felületen keresztül le lehet kérdezni, ill. változtatni lehet.

Előkészítő jelzés: a szabad jelzést megelőző jelzéskép. Járműveknél és kerékpároknál 2s, gyalogosoknál 0s.

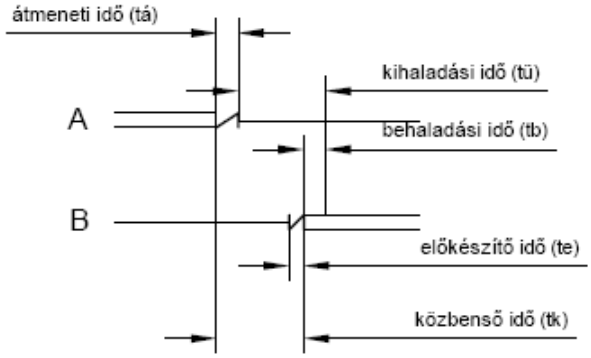
Fázis: az egyidejűleg megengedett csomóponti mozgások csoportja.

Forgalomirányító berendezés: a vezérelt fényjelző készülékkel és a csatlakozó illesztőberendezésekkel együttműködő közúti fényjelző vezérlőberendezés.

Forgalomfüggő irányítás: a forgalomirányítás, ill. a jelzések egymás után következésének azon módja, ahol a – helyszíni adottságok és a biztonsági előírások miatt megkötött jelzéseken kívül – egy vagy több vagy az összes jelzés közvetve ill. közvetlenül függ a forgalom nagyságától.

A kihaladási távolság (a helyzetjelző vonaltól a konfliktuszóna széle+6m; gyalogosoknál a gyalogátkelő tengelyben mért hossza) osztva a kihaladási sebességgel.

Közbenső idő: biztonsági szempontból az egyidejűleg tiltott szabad jelzések között biztosítandó legrövidebb idő.



29. ábra A közbenső idő grafikus magyarázata ([1] alapján)

OCIT-protokoll: Az OCIT-protokoll (Open Communication Interface for Road Control Traffic Systems) internet technológián alapuló nyílt interfészek megvalósítására szolgáló protokoll, amit kizárólag a közúti forgalomirányításban használnak.

Periódusidő: a jelzésképek egyszeri ismétlődése közötti idő. Általában 60, 90, 120s nagyságú.

Program (jelzésterv): egy csomópont összes fényjelző készülékén megjelenő valamennyi jelzési kép időtartama (azok kezdésének és befejezésének időpontjával együtt), vagy az ennek meghatározására vonatkozó feltételek. Forgalomtechnikai megjelenése a fázisidőterv, fázisterv.

TCP/IP protokoll: A TCP/IP (Transmission Control Protocol/Internet Protocol) egy több rétegből álló általános használatú protokollkészlet. Hálózatok közötti és az interneten keresztül történő adatátvitel szabványa.

2. melléklet: jelölésjegyzék

$\lambda\left(\frac{J}{s}\right)$: egy sávban érkező járművek száma 1s alatt

$\mu\left(\frac{J}{s}\right)$: maximális kihaladó járműszám másodpercenként

$\rho(-)$: forgalmi intenzitás

$r(s)$: pirosidő egy fázisban

$t_c(s)$: sor kiürülésének ideje

$C(s)$: a csomópont jelzéstervének periódusideje

$P_s(\%)$: a megállított járművek aránya egy periódusban egy irányban

$g(s)$: effektív zöldidő

$N(db)$: periódusonként megállt járművek száma

$d_{avg}(s)$: egy járműre jutó késés egy irányban

$F_{car}\left(\frac{l}{s}\right)$: személygépkocsik tüzelőanyag-fogyasztása alapjáraton

$F_{bus}\left(\frac{l}{s}\right)$: autóbuszok tüzelőanyag-fogyasztása alapjáraton

$C_{CO_c}\left(\frac{Euro}{t\_CO_2}\right)$: CO_2 kibocsátás költsége

$m_{CO_{2_{benzin}}}\left(\frac{kg}{l}\right)$: 1 liter benzin elégetéséből származó szén-dioxid tömege

$m_{CO_{2_{diesel}}}\left(\frac{kg}{l}\right)$: 1 liter gázolaj elégetéséből származó szén-dioxid tömege

$m_{CO_{2_{car}}}\left(\frac{kg}{l}\right)$: személygépkocsik egyenérték-kibocsátása

$N_{pass_{car}}(db)$: utazók átlagszáma egy személygépkocsiban

$N_{pass_{bus}}(db)$: utazók átlagszáma egy autóbuszon

$C_{d_{dev}}\left(\frac{Euro}{\frac{késett\_óra}{személy}}\right)$: késésből adódó externális költség

$C_{cl_{car}}(Euro)$: személygépkocsik klimatikus költsége

$C_{cl_{bus}}(Euro)$: autóbuszok klimatikus költsége

$C_{d_{car}}(Euro)$: személygépkocsik utasai által a késés miatt elszenvedett költség

$C_{d_{bsu}}(Euro)$: autóbuszok utasai által a késés miatt elszenvedett költség

$C_{acl_{bus}}(Euro)$: az előnybiztosítás miatt az autóbuszok meg nem termelt klimatikus költsége

$C_{ad_{bsu}}(Euro)$: az előnybiztosítás miatt az autóbuszok utasai által el nem szenvedett késési költség

$C_{s_{sum}}(Euro)$: a csomópont teljes társadalmi költsége

3. melléklet: a program kódja

BusProg.java

```
package budapest;
import vt.*;

public class BusProg extends LogikProg{

		public BusProg(TeilKnoten kn, String name, int num,
				int umlZeit, int gwpa, int gwpb, int warteZeit, int versatzZeit) {
		    super(kn, name, num, umlZeit, gwpa, gwpb, warteZeit, versatzZeit);
		}



		public void programmFunktion()


		{
			//bejelentkezés ocit detektorokról

		   Var.ocit1=Var.d04.eingangGesetzt();
		 if (Var.ocit1==true) {
		    Var.utasszam=(int) (151*Math.random());
		    Var.utasszam=(Var.utasszam/10)*(4/10);
		    Var.keses=(int) (21*Math.random());
		    Var.keses=Var.keses*(3/10);
		    Var.hhk=(int) (2*Math.random()+1);
		    Var.hhk=Var.hhk*(2/10);
		    Var.lgy=(int) (2*Math.random()+3);
		    Var.lgy=Var.lgy/10;

		   //pontosan közlekedő,és/vagy 0 utasú busz nem kap előnyt

		    if ((Var.utasszam!=0) &&(Var.keses!=0)){
				Var.pontszam=1/(Var.utasszam+Var.keses+Var.hhk+Var.lgy);


				for (Var.szamlalo=0; Var.szamlalo<20; Var.szamlalo++){
				       System.out.println("Ciklusvaltozo: "+Var.szamlalo);
				        if (Var.bejelentkezesek[0][Var.szamlalo]==0){
				                break;
				        }
				}

				System.out.println("ellenorzes: "+Var.szamlalo);
				Var.bejelentkezesek[0][Var.szamlalo]=1;
				Var.bejelentkezesek[1][Var.szamlalo]=1;
				Var.bejelentkezesek[4][Var.szamlalo]=Var.pontszam;
				Var.bejelentkezesek[5][Var.szamlalo]=Var.tk1.getProgSek();
				System.out.println("4-es detektoron bejelentkezes; pontszam:
"+Var.bejelentkezesek[4][Var.szamlalo]+" Bejelentkezesek matrixban elfoglalt oszlop szama: "+
Var.szamlalo);
		    }
		     Var.ocit1=false;
		     }


		 Var.ocit2=Var.d02.eingangGesetzt();
		 if (Var.ocit2==true) {
		    Var.utasszam=(float) (151*Math.random());
		    Var.utasszam=Var.utasszam*4/10;
		    Var.keses=(float) (21*Math.random());
		    Var.keses=Var.keses*3/10;
		    Var.hhk=(int) (2*Math.random()+1);
		    Var.hhk=Var.hhk*2/10;
		    Var.lgy=(int) (2*Math.random()+3);
		    Var.lgy=Var.lgy/10;

		    if ((Var.utasszam!=0) &&(Var.keses!=0)){
				Var.pontszam=1/(Var.utasszam+Var.keses+Var.hhk+Var.lgy);

				for (Var.szamlalo=0; Var.szamlalo<20; Var.szamlalo++){
```

```
                        System.out.println("Ciklusvaltozo: "+Var.szamlalo);
                         if (Var.bejelentkezesek[0][Var.szamlalo]==0){
                                break;
                         }
                    }

                    System.out.println("ellenorzes: "+Var.szamlalo);
                    Var.bejelentkezesek[0][Var.szamlalo]=1;
                    Var.bejelentkezesek[2][Var.szamlalo]=1;
                    Var.bejelentkezesek[4][Var.szamlalo]=Var.pontszam;
                    Var.bejelentkezesek[5][Var.szamlalo]=Var.tk1.getProgSek();
                    System.out.println("2-es detektoron bejelentkezes; pontszam:
"+Var.bejelentkezesek[4][Var.szamlalo]+" Bejelentkezesek matrixban elfoglalt oszlop szama: "+
Var.szamlalo);

                }
                Var.ocit2=false;
            }

                Var.ocit3=Var.d03.eingangGesetzt();
            if (Var.ocit3==true) {
                Var.utasszam=(float) (151*Math.random());
                Var.utasszam=Var.utasszam*4/10;
                Var.keses=(float) (21*Math.random());
                Var.keses=Var.keses*3/10;
                Var.hhk=(int) (2*Math.random()+1);
                Var.hhk=Var.hhk*2/10;
                Var.lgy=(int) (2*Math.random()+3);
                Var.lgy=Var.lgy/10;

                if ((Var.utasszam!=0) &&(Var.keses!=0)){
                        Var.pontszam=1/(Var.utasszam+Var.keses+Var.hhk+Var.lgy);

                        for (Var.szamlalo=0; Var.szamlalo<20; Var.szamlalo++){
                               System.out.println("Ciklusvaltozo: "+Var.szamlalo);
                                if (Var.bejelentkezesek[0][Var.szamlalo]==0){
                                        break;
                                }
                        }

                        System.out.println("ellenorzes: "+Var.szamlalo);
                        Var.bejelentkezesek[0][Var.szamlalo]=1;
                        Var.bejelentkezesek[3][Var.szamlalo]=1;
                        Var.bejelentkezesek[4][Var.szamlalo]=Var.pontszam;
                        Var.bejelentkezesek[5][Var.szamlalo]=Var.tk1.getProgSek();
                        System.out.println("3-as detektoron bejelentkezes; pontszam:
"+Var.bejelentkezesek[4][Var.szamlalo]+" Bejelentkezesek matrixban elfoglalt oszlop szama: "+
Var.szamlalo);
                }
                Var.ocit3=false;
            }


                // a régi (első 10) bejelentkezések törlése

            if (Var.bejelentkezesek[0][19]==1)
            {for (Var.szamlalo=0; Var.szamlalo<7; Var.szamlalo++){
                for (Var.szamlalo2=0; Var.szamlalo2<10; Var.szamlalo2++){
                        Var.bejelentkezesek[Var.szamlalo][Var.szamlalo2]=0;
                }
            }

            for (Var.szamlalo=0; Var.szamlalo<7; Var.szamlalo++){
                for (Var.szamlalo2=0; Var.szamlalo2<10; Var.szamlalo2++){

       Var.bejelentkezesek[Var.szamlalo][Var.szamlalo2]=Var.bejelentkezesek[Var.szamlalo][Var.
szamlalo2+10];
                }
            }
            for (Var.szamlalo=0; Var.szamlalo<7; Var.szamlalo++){
                for (Var.szamlalo2=10; Var.szamlalo2<20; Var.szamlalo2++){
                        Var.bejelentkezesek[Var.szamlalo][Var.szamlalo2]=0;
            }}}}}
```

Init.java

```
package budapest;

import det.*;
import hw.*;
import hw.hwlmp.*;
import hw.nkk.*;
import sg.*;
import uhr.*;
import vt.*;
import vtvar.*;

public class Init implements enp.Initialisierung {

    //ZwischenzeitenMatrix
    public ZwzMatrix zwz;

    /**
     * Main-Methode; Muss nicht geaendert werden!!!
     */
     public static void main(String[] args) {
         System.out.println("Budapest - Version 1.0, 21.06.2006; Kraushaar, VT-S");

         Init init1 = new Init();
         enp.Actros.registerInitialisierung(init1);
         init1.initialisiereSg();
         init1.initialisiereZwz();
         init1.initialisiereDet();
         init1.initialisiereProgs();
         init1.initialisierePhasen();
         init1.initialisiereParameter();
         init1.initialisiereZentrale();
         init1.initialisiereDebug();
         init1.initialisiereUhr();
         init1.initialisiereHW();
     }

     protected void initialisiereSg(){

         //Knoten initialisieren
         Var.tk1 = new TeilKnoten("Budapest"); //Name auf Anlage anpassen!!

         //Signalgruppen
         //Sg-Def:
           Teil-knoten|Sg-Name |Sg-Typ |min. Frei|min. gesp.|Richtung|Sg-Nr.|

 Var.k1=new Sg(Var.tk1,"K1"    ,Var.kfz,   10    ,    2     ,Sg.HR   ,   1  );
 Var.k2=new Sg(Var.tk1,"K2"    ,Var.kfz,    5    ,    2     ,Sg.NR   ,   2  );
 Var.k3=new Sg(Var.tk1,"K3"    ,Var.kfz,   10    ,    2     ,Sg.HR   ,   3  );
 Var.f21=new Sg(Var.tk1,"F21"  ,Var.fg,    5     ,    2     ,Sg.HR   ,   4  );
 Var.f22=new Sg(Var.tk1,"F22"  ,Var.fg,    7     ,    2     ,Sg.HR   ,   5  );

         //Wiederholer initialisieren
         Var.k1a = new Wiederholer("K1a");
         Var.k1.initWh(Var.k1a);

         Var.k2a = new Wiederholer("K2a");
         Var.k2.initWh(Var.k2a);

         Var.k3a = new Wiederholer("K3a");
         Var.k3b = new Wiederholer("K3b");
         Var.k3.initWh(Var.k3a,Var.k3b);

         Var.f21a= new Wiederholer("F21a");
         Var.f21.initWh(Var.f21a);

         Var.f22a= new Wiederholer("F22a");
         Var.f22.initWh(Var.f22a);

         //Abschaltcode initialisieren
         Var.k1.initAbschUGA(Var.lrot, "UGA 1");
         Var.k2.initAbschUGA(Var.lrot, "UGA 1");
         Var.k3.initAbschUGA(Var.lrot, "UGA 1&(2#3)");
         Var.f21.initAbschUGA(Var.lrot, "Keine");
         Var.f22.initAbschUGA(Var.lrot, "Keine");
```

```
        //Gibt das Ende der Signalgruppeninitialisierung an
        Sg.endInit();
        System.out.println("Signalgruppen sind initialisiert");
    }

    protected void initialisiereZwz(){
        //Zwischenzeiten
        zwz = new ZwzMatrix();

        //          Raeumer   | Einfahrer| Tz  | Gegenkonflikt
        zwz.setzeZwz( Var.k1  , Var.k2   ,   6 ,       6 );
        zwz.setzeZwz( Var.k1  , Var.f21  ,   4 ,      10 );

        zwz.setzeZwz( Var.k2  , Var.k3   ,   6 ,       6 );
        zwz.setzeZwz( Var.k2  , Var.f22  ,   4 ,      14 );

        zwz.setzeZwz( Var.k3  , Var.f21  ,   7 ,      10 );

        //Ende der Zwischenzeiteninitialisierung
        zwz.endInit();
        System.out.println("Zwischenzeiten sind initialisiert");
    }

    protected void initialisiereDet(){
//Detektorinit:
                    Knoten-name|Name |T-DB|TZAPPL| T-NB |Det.Nr.| Sg     |Anf-zeit |Anf/Bel
Var.t21=new Detektor(Var.tk1    ,"T21",200 ,10 ,Detektor.KEINE_UEBERWACHUNG,   5 ,
Var.f21,0,0);
        Var.t22  = new Detektor( Var.tk1   , "T22"  ,  200 ,   10
,Detektor.KEINE_UEBERWACHUNG,   6 , Var.f22,0,0);
        //Var.d01  = new Detektor( Var.tk1   , "OCIT_DET_01"  ,  200 ,   10
,Detektor.KEINE_UEBERWACHUNG,   1 , Var.k1,0,0);
        Var.d02  = new Detektor( Var.tk1   , "OCIT_DET_02"  ,  200 ,   10
,Detektor.KEINE_UEBERWACHUNG,   2 , Var.k2,0,0);
        Var.d03  = new Detektor( Var.tk1   , "OCIT_DET_03"  ,  200 ,   10
,Detektor.KEINE_UEBERWACHUNG,   3 , Var.k3,0,0);
        Var.d04  = new Detektor( Var.tk1   , "OCIT_DET_04"  ,  200 ,   10
,Detektor.KEINE_UEBERWACHUNG,   4 , Var.k1,0,0);


        // Ausgänge
        Var.sk21  = new Ausgang(Var.tk1, "SK31" , 22);
        Var.sk22  = new Ausgang(Var.tk1, "SK32" , 23);
        Var.out1  = new Ausgang(Var.tk1, "OUT1" , 19);
        Var.out2  = new Ausgang(Var.tk1, "OUT2" , 20);
        Var.out3  = new Ausgang(Var.tk1, "OUT3" , 21);
        // Funktionen aktivieren

        //Ende der Detektor- und Ausgangsinitialisierung
        Det.endInit();
        System.out.println("Detektoren sind initialisiert");
    }

    protected void initialisiereProgs(){
        //Systemprogramme
        //                  |Ref. auf TK| Prog Name |Prog Nr|Umlaufzeit
Var.einprog=new MyEinProgramm(  Var.tk1  ,"EinProg"  ,  30   ,   24    );
Var.ausprog=new MyAusProgramm(  Var.tk1  ,"AusProg"  ,  31   ,   24    );
Var.blinkprog=new MyBlinkProgramm(Var.tk1,"BlinkProg",  29   ,   10    );

        //VT-Programme, Festzeit-, Handschalt- oder Logikprogramme
                    Ref.auf.TK|  Name  |Prog Nr.| UZ |GWPA|GWPB|Wzeit|Vz
Var.BusProg1=new BusProg(Var.tk1,"Busprg",   12   ,9999, 0  ,  0 , 999 ,0);

        //Main- und PostMainProgramm
        Var.main  = new MyMainFkt(Var.tk1);
        Var.postmain  = new MyPostMainFkt(Var.tk1);
        Var.umschalt  = new MyUmschaltFkt(Var.tk1);

        Vt.endInit();//Ende der Programminitialisierung
        System.out.println("Programme sind initialisiert");

    }

    protected void initialisierePhasen(){
        //Phasen
```

```
        Var.phase1 = new Phase1("Phase1", 1);
        Var.phase2 = new Phase2("Phase2", 2);
        Var.phase3 = new Phase3("Phase3", 3);
        Var.phase4 = new Phase4("Phase4", 4);

        //Phasenuebergaenge
        Var.phUeb12= new PhUeb12("Phue1-2", 12);
        Var.phUeb23= new PhUeb23("Phue2-3", 23);
        Var.phUeb24= new PhUeb24("Phue2-4", 24);
        Var.phUeb34= new PhUeb34("Phue3-4", 34);
        Var.phUeb41= new PhUeb41("Phue4-1", 41);

        Var.phUeb13= new PhUeb13("Phue1-3", 13);
        Var.phUeb14= new PhUeb14("Phue1-4", 14);
        Var.phUeb21= new PhUeb21("Phue2-1", 21);
        Var.phUeb31= new PhUeb31("Phue3-1", 31);
        Var.phUeb32= new PhUeb32("Phue3-2", 32);
        Var.phUeb42= new PhUeb42("Phue4-2", 42);
        Var.phUeb43= new PhUeb43("Phue4-3", 43);

        System.out.println("Phasen./Phasenuebergaenge sind initialisiert");
    }

    protected void initialisiereParameter()  {
        // globale Parameter:

        // satzgebundene Parameter, Templates
        ParSets.mingruen = new ParZeit("gruenA");

        // satzgebundene Parameter fuer VA-Programm 1:
        int[] gnA_P1 = new int[] {10,  5, 10,  9, 14};
      //  int[] gnA_P2 = new int[] {10,  5, 10,  9, 14};
         ParameterDaten.assign(Var.BusProg1, ParSets.mingruen, gnA_P1);

        VtVar.endInit();
        System.out.println("Variablen sind initialisiert");
    }

    protected void initialisiereZentrale(){
        //Zentrale
//      System.out.println("Zentrale ist initialisiert");
    }

    protected void initialisiereDebug() {
      //Debugmodul
    }

    protected void initialisiereUhr(){
        //Uhrenautomatik (Tages- und Wocheplaene)
        TagesPlan mo_do = new TagesPlan("Mo_Do", Var.blinkprog);
        mo_do.initProgWunsch( 6 ,  0,  Var.BusProg1);
        //mo_do.initProgWunsch(22 ,  0,  Var.logikProg1);

        TagesPlan fr = new TagesPlan("Fr", Var.blinkprog);
        fr.initProgWunsch( 6 ,  0,  Var.fzProg11);
        fr.initProgWunsch(22 ,  0,  Var.blinkprog);

        TagesPlan sa = new TagesPlan("Sa", Var.blinkprog);
        sa.initProgWunsch( 6 ,  0,  Var.fzProg11);
        sa.initProgWunsch(22 ,  0,  Var.blinkprog);

        TagesPlan so = new TagesPlan("So", Var.blinkprog);
        so.initProgWunsch( 6 ,  0,  Var.fzProg11);
        so.initProgWunsch(22 ,  0,  Var.blinkprog);

        WochenPlan wp1 = new WochenPlan("Wochenplan_1", mo_do, mo_do, mo_do, mo_do, fr, sa,
so);

        System.out.println("Wochenplan ist initialisiert");
    }


    protected void initialisiereHW(){
        //Anlage initialisieren
        Var.anlage = new Anlage(Var.tk1);
        Var.anlage.setName("Budapest, Testanlage");
```

```
        //Zykluszeit auf  500 ms setzen
        Var.anlage.setZyklZeitHalbeSek();

        //Hardwarelampentypen
        HwLmpTyp lmp200Rot          = new LmpOCITRotLed40V();
        HwLmpTyp lmp200Gelb         = new LmpOCITGelbGruenLed40V(Hw.GELB);
        HwLmpTyp lmp200Gruen        = new LmpOCITGelbGruenLed40V(Hw.GRUEN);

        //NKK
        Nkk nkk = new Nkk40V50Hz();

        //Schalterkarten
        SchalterKarte sk1 = new SchalterKarte();

        //IO24-Karten
        IoKarte io1 = new IoKarte();

        //Schaltkanaele
       SchaltKanal sk1_1=new SchaltKanal(Var.k1,Var.lrot,lmp200Rot,Hw.HF,sk1, 1,Hw.SK);
       SchaltKanal sk1_2=new SchaltKanal(Var.k1a,Var.lrot,lmp200Rot,Hw.HF,sk1,2,Hw.SK);
       SchaltKanal sk1_3=new SchaltKanal(Var.k1,Var.lgelb,lmp200Gelb,Hw.HF,sk1, 3,Hw.KK);
       SchaltKanal sk1_4=new SchaltKanal(Var.k1,Var.lgruen,lmp200Gruen,Hw.HF,sk1, 4,Hw.KK);
       SchaltKanal sk1_5=new SchaltKanal(Var.k2,Var.lrot,lmp200Rot,Hw.HF,sk1,5,Hw.SK);
       SchaltKanal sk1_6=new SchaltKanal(Var.k2a,Var.lrot,lmp200Rot,Hw.HF,sk1, 6,Hw.SK);
       SchaltKanal sk1_7=new SchaltKanal(Var.k2,Var.lgelb,lmp200Gelb,Hw.HF,sk1, 7, Hw.KK);
       SchaltKanal sk1_8=new SchaltKanal(Var.k2,Var.lgruen,lmp200Gruen,Hw.HF,sk1, 8,Hw.KK);
       SchaltKanal sk1_9=new SchaltKanal(Var.k3,Var.lrot,lmp200Rot,Hw.HF,sk1, 9,Hw.SK);
       SchaltKanal sk1_10=new SchaltKanal(Var.k3a,Var.lrot,lmp200Rot,Hw.HF,sk1,10,Hw.SK);
       SchaltKanal sk1_11=new SchaltKanal(Var.k3b,Var.lrot,lmp200Rot,Hw.HF,sk1,11,Hw.SK);
       SchaltKanal sk1_12=new SchaltKanal(Var.k3,Var.lgelb,lmp200Gelb,Hw.HF,sk1,12,Hw.KK);
       SchaltKanal sk1_13=new SchaltKanal(Var.k3,Var.lgruen,lmp200Gruen,Hw.HF,sk1,13,Hw.KK);
       SchaltKanal sk1_14=new SchaltKanal(Var.f21,Var.lrot,lmp200Rot,Hw.HF,sk1,14,Hw.SK);
       SchaltKanal sk1_15=new SchaltKanal(Var.f21a,Var.lrot,lmp200Rot,Hw.HF,sk1,15,Hw.SK);
       SchaltKanal sk1_16=new SchaltKanal(Var.f21,Var.lgruen,lmp200Gruen,Hw.HF,sk1,16,Hw.KK);
       SchaltKanal sk1_17=new SchaltKanal(Var.f22,Var.lrot,lmp200Rot,Hw.HF,sk1,17,Hw.SK);
       SchaltKanal sk1_18=new SchaltKanal(Var.f22a,Var.lrot,lmp200Rot,Hw.HF,sk1,18,Hw.SK);
       SchaltKanal sk1_19=new SchaltKanal(Var.f22,Var.lgruen,lmp200Gruen,Hw.HF,sk1,19,Hw.KK);

        VirtKlemme  vk1    = new VirtKlemme(Var.k1a,Var.lgelb);
        VirtKlemme  vk2    = new VirtKlemme(Var.k1a,Var.lgruen);
        VirtKlemme  vk3    = new VirtKlemme(Var.k2a,Var.lgelb);
        VirtKlemme  vk4    = new VirtKlemme(Var.k2a,Var.lgruen);
        VirtKlemme  vk5    = new VirtKlemme(Var.k3a,Var.lgelb);
        VirtKlemme  vk6    = new VirtKlemme(Var.k3b,Var.lgelb);
        VirtKlemme  vk7    = new VirtKlemme(Var.k3a,Var.lgruen);
        VirtKlemme  vk8    = new VirtKlemme(Var.k3b,Var.lgruen);
        VirtKlemme  vk9    = new VirtKlemme(Var.f21a,Var.lgruen);
        VirtKlemme  vk10   = new VirtKlemme(Var.f22a,Var.lgruen);

        IoKanal io1_1 = new IoKanal (  Var.d01  , io1,  1);
        IoKanal io1_2= new IoKanal (  Var.d02  , io1, 2);
        IoKanal io1_3 = new IoKanal (  Var.d03  , io1,  3);
        IoKanal io1_4= new IoKanal (  Var.d04  , io1, 4);

        IoKanal io1_9 = new IoKanal (  Var.t21  , io1,  9);
        IoKanal io1_10= new IoKanal (  Var.t22  , io1, 10);

        IoKanal io1_17= new IoKanal (  Var.sk21 , io1, 17);
        IoKanal io1_18= new IoKanal (  Var.sk22 , io1, 18);
        IoKanal io1_19= new IoKanal (  Var.out1 , io1, 19);
        IoKanal io1_20= new IoKanal (  Var.out2 , io1, 20);
        IoKanal io1_21= new IoKanal (  Var.out3 , io1, 21);


        //Anzeige des Initialisierungsendes
        System.out.println("Java-Initialisierung der Anlage abgeschlossen!");

        Var.anlage.startAnlage();
    }
    public String getVersion()
    {
       return "Budapest, Testanlage - Version 1.0, 21.06.2006";
    }

}
```

MinSearch.java

```
package budapest;

public class MinSearch {

    public static void getBejelentkezes()
    {

            Var.min=0;


            for (Var.szamlalomin=0; Var.szamlalomin<20; Var.szamlalomin++){
               Var.vizsgalt[Var.szamlalomin]=9999;
            }

           Var.szamlalomin=19;

              while ((Var.szamlalomin>=0) && (Var.bejelentkezesek[6][Var.szamlalomin]==0)){

               if (Var.bejelentkezesek[4][Var.szamlalomin]==0){
                       Var.vizsgalt[Var.szamlalomin]=9999;
                       }
               else {
                       Var.vizsgalt[Var.szamlalomin]=Var.bejelentkezesek[4][Var.szamlalomin];
               }
               Var.szamlalomin--;
            }


            for (Var.szamlalomin=0; Var.szamlalomin<20; Var.szamlalomin++){
               if (Var.vizsgalt[Var.szamlalomin]<=Var.vizsgalt[Var.min]){
                       Var.min=Var.szamlalomin;
               }
            }
            //ha a üres a vektor, akkor ne csináljon semmit

            if (Var.vizsgalt[Var.min]!=9999) {
            if (Var.bejelentkezesek[1][Var.min]==1){
               Var.k1ben=true;
               Var.k2ben=false;
               Var.k3ban=false;}
               else if (Var.bejelentkezesek[2][Var.min]==1){
                       Var.k2ben=true;
                       Var.k1ben=false;
                       Var.k3ban=false;}
               else if (Var.bejelentkezesek[3][Var.min]==1)
                       {Var.k3ban=true;
                       Var.k1ben=false;
                       Var.k2ben=false;}
            }

    System.out.println("MS lef; Ir: K1: "+Var.k1ben+"; K2: "+Var.k2ben+"; K3: "+Var.k3ban);
    }

}
```

MyAusProgramm.java

```
package budapest;

import sg.*;
import vt.*;


public class MyAusProgramm extends AusProg{

    public MyAusProgramm(TeilKnoten tk, String name, int nr, int tu) {
        super(tk, name, nr, tu);
    }

    public void programmFunktion(){
```

```
      Var.k1.setSg(Zustand.DUNKEL      ,  10);
      Var.k3.setSg(Zustand.DUNKEL      ,  10);
      Var.f22.setSg(Zustand.GESPERRT   ,  10);
    }
}
```

MyBlinkProgramm.java

```
package budapest;
import sg.*;
import vt.*;


public class MyBlinkProgramm extends BlinkProg{

    public MyBlinkProgramm(TeilKnoten tk, String name, int nr, int tu) {
        super(tk, name, nr, tu);
    }

    public void programmFunktion(){

      Var.k1.setSg(Zustand.DUNKEL      ,  0);
      Var.k2.setSg(Zustand.GELBBLINKEN ,  0);
      Var.k3.setSg(Zustand.DUNKEL      ,  0);
      Var.f21.setSg(Zustand.DUNKEL     ,  0);
      Var.f22.setSg(Zustand.DUNKEL     ,  0);
    }
}
```

MyEinProgramm.java

```
package budapest;

import sg.*;
import vt.*;

public class MyEinProgramm extends EinProg{

    public MyEinProgramm(TeilKnoten tk, String name, int nr, int tu) {
        super(tk, name, nr, tu);
    }

    public void programmFunktion(){

        Var.k1.setSg(Zustand.FREI     , 14);
        Var.k2.setSg(Zustand.GESPERRT ,  0);
        Var.k3.setSg(Zustand.GESPERRT  , 0);
        Var.f21.setSg(Zustand.GESPERRT,  0);
        Var.f22.setSg(Zustand.GESPERRT,  0);
        Var.f22.setSg(Zustand.GESPERRT , 0);
    }
}
```

MyMainFkt.java

```
package budapest;

import vt.*;
import sg.*;
import vtvar.*;

public class MyMainFkt extends MainFkt{

    private ProgBase prwunsch;
    public MyMainFkt(TeilKnoten akttk)
    {
        super(akttk);
    }

    public void inJedemZyklus(){
```

```
        /* Zentralensteuerung */
        // keine

        /* Anforderungen ruecksetzten */
        if (Var.f21.getZustand() == 4) {
           Var.t21.resetAnforderung();
        }
        if (Var.f22.getZustand() == 4) {
           Var.t22.resetAnforderung();
        }

        /* Zeitluecken        */

        /* Anforderungsbedingungen */
        Var.an_k2   = (Var.k2.getZustand() == Zustand.ROT);
        Var.an_f21  = Var.t21.getAnforderung();
        Var.an_f22  = Var.t22.getAnforderung();
        Var.an_nr   = Var.an_k2 || Var.an_f21;

        /* Wartezeitzaehler */

        /* Abbruchbedingungen */
        if (Var.tk1.getAktProgId() == Var.BusProg1.getId()) {
           Var.ab_k1 = (Var.k1.getZustand() == Zustand.GRUEN) && (Var.k1.getZustSek() >=
ParSets.mingruen.k1.get());
           Var.ab_k2 = (Var.k2.getZustand() == Zustand.GRUEN) && (Var.k2.getZustSek() >=
ParSets.mingruen.k2.get());
           Var.ab_k3 = (Var.k3.getZustand() == Zustand.GRUEN) && (Var.k3.getZustSek() >=
ParSets.mingruen.k3.get());
           Var.ab_f21= (Var.f21.getZustand() == Zustand.GRUEN) && (Var.f21.getZustSek() >=
ParSets.mingruen.f21.get());
           Var.ab_f22= (Var.f22.getZustand() == Zustand.GRUEN) && (Var.f22.getZustSek() >=
ParSets.mingruen.f22.get());
        }
    }
}
```

MyPostMainFkt.java

```
package budapest;

import sg.*;
import vt.*;


public class MyPostMainFkt extends PostMainFkt{

    public MyPostMainFkt (TeilKnoten akttk){
        super(akttk);
    }

    public void inJedemZyklus(){

        //Zentralerueckmeldung

        //Blinkersteuerung

        //Signal kommt
       //gyalogosjelzők
        if  (Var.tk1.getAktProg().getNummer() < 28) {
            if  (Var.an_f21 && !Var.sk21.getAusgang()) {
                Var.sk21.setAusgang();
            }
            else {
                 if  (!Var.an_f21 && Var.sk21.getAusgang()) {
                     Var.sk21.resetAusgang();
                 }
            }
            if  (Var.an_f22 && !Var.sk22.getAusgang()) {
                Var.sk22.setAusgang();
            }
            else {
                 if  (!Var.an_f22 && Var.sk22.getAusgang()) {
```

```
                    Var.sk22.resetAusgang();
                }
            }

            //OCIT-detektorok
            if  (Var.d04.eingangGesetzt()==true){
               Var.out1.setAusgang();
            }
            else Var.out1.resetAusgang();

            if  (Var.d02.eingangGesetzt()==true){
               Var.out2.setAusgang();
            }
            else Var.out2.resetAusgang();

            if  (Var.d03.eingangGesetzt()==true){
               Var.out3.setAusgang();
            }
            else Var.out2.resetAusgang();
        }

        else {
           Var.sk21.resetAusgang();
           Var.sk22.resetAusgang();
           Var.out1.resetAusgang();
           Var.out2.resetAusgang();
           Var.out3.resetAusgang();
        }
   }

}
```

MyUmschaltFkt.java

```
package budapest;
import vt.*;
import sg.*;

public class MyUmschaltFkt extends UmschaltFkt {

    public MyUmschaltFkt(TeilKnoten akttk) {
        super(akttk.getId());
    }

    private boolean Default() {
    int t41 = Var.tk1.getProgZeit();
    int gwpa= Var.tk1.getAktProg().getGwpA()*1000; //Korektur, da Progammzeit in ms
    int gwpb= Var.tk1.getAktProg().getGwpB()*1000; //Korektur, da Progammzeit in ms

       if (gwpb > gwpa) {
          if ((t41 >= gwpa) && (t41 <= gwpb)) { return true; }
          else { return false; }
       }
       else if (gwpa > gwpb) {
          if ((t41 >= gwpa) || (t41 <= gwpb)) { return true; }
          else { return false; }
       }
       else {
          if (t41 == gwpa) { return true; }
          else { return false; }
       }
    }

    public boolean umschFkt() {
        Var.umschalten = true;
        if  (Default() == true) {
            if  ((Var.k1.getZustand() == Zustand.GRUEN) && (Var.k3.getZustand() ==
Zustand.GRUEN) &&
                (Var.f22.getZustand() == Zustand.GRUEN) &&
                (Var.k1.getAbzulMindSek() == 0) && (Var.k3.getAbzulMindSek() == 0) &&
                (Var.f22.getAbzulMindSek() == 0) ) {
                System.out.println("Variablen uebernommen: " + vtvar.VtVar.update(Var.tk1));
                Var.umschalten = false;
                return true;
            }
```

```
        }
        return false;
    }
}
```

ParSets.java

```
package budapest;
import vtvar.*;

public class ParSets {

  //globale Parameter:

  public static ParZeit mingruen;
}
```

ParZeit.java

```
package budapest;
import vtvar.*;

public class ParZeit extends ParameterSatz
{
    // satzgebundene Parameter:
    public ParaInt k1, k2, k3;
    public ParaInt f21, f22;

    public ParZeit(String name) {
        super(name, true, true, true);
    }

}
```

Phase1.java

```
package budapest;

import sg.*;
import vt.*;

public class Phase1 extends Phase {

    public Phase1(String name, int nr) {
        super(name, nr);
    }


    public Phase phasenFunktion() {
        //Hier wird die Phasenlogik implementiert
        if (Var.tk1.getAktProgId() == Var.BusProg1.getId()) {

           /*if ((Var.tk1.getProgSek() > Var.phase1.getPhasenSek()) || !Var.an_nr) {
              Var.tk1.setProgSek(0);
              return KEINE_UMSCHALTUNG;*/
           }

        MinSearch.getBejelentkezes();

           //fázis 1 nyújtása

        if (Var.k1ben && Var.phase1.getPhasenSek()>Var.zoldk1k3-10)
             {Var.ido1=10-(25-Var.phase1.getPhasenSek());
              Var.ido2=Var.phase1.getPhasenSek()-Var.ido1;
                 Var.phase1.setPhasenSek(Var.ido2);
             Var.bejelentkezesek[6][Var.min]=1;
             Var.k1ben=false;
             System.out.println("Fazis 1 nyujtas");}


           // fázis 3 beillesztése
            if ((Var.k3ban)&&(Var.ab_k1))
```

```
            {Var.beillesztes=true;
            Var.bejelentkezesek[6][Var.min]=1;
            Var.elozofazis=1;
            Var.k3ban=false;
            System.out.println("Fazis 3 beillesztese");
            return Var.phUeb13;}

          //fázis 4 beillesztése
          if ((Var.k2ben)&&(Var.ab_k1))
           {Var.beillesztes=true;
            Var.bejelentkezesek[6][Var.min]=1;
            Var.elozofazis=1;
            Var.k2ben=false;
            System.out.println("Fazis 4 beillesztes");
            return Var.phUeb14;}

          if ((Var.beillesztes==true) && (Var.phase1.getPhasenSek()>10)){
                Var.beillesztes=false;
                if (Var.elozofazis==2){
                      Var.elozofazis=0;
                      return Var.phUeb13;
                }
                if (Var.elozofazis==3){
                      Var.elozofazis=0;
                      return Var.phUeb14;
                }

          }

      if (Var.k1ben && Var.phase1.getPhasenSek()<=Var.zoldk1k3-10){
            Var.k1ben=false;
            System.out.println("Nem kell nyujtani");
      }

          System.out.println("1.fazis fut; fazis ideje: " +Var.phase1.getPhasenSek());
          if (Var.phase1.getPhasenSek()>25) {
                if (Var.an_f21||Var.an_f22){
                  return Var.phUeb12;}
             return Var.phUeb13;
          }

       return KEINE_UMSCHALTUNG;
    }
}
```

Phase2.java

```
package Budapest;

import sg.*;
import vt.*;

public class Phase2 extends Phase {

    public Phase2(String name, int nr) {
        super(name, nr);
    }

    public Phase phasenFunktion() {

        MinSearch.getBejelentkezes();

        //fázis1 beillesztése
        if (Var.k1ben && Var.ab_f21 && Var.ab_f22)
               {Var.beillesztes=true;
               Var.bejelentkezesek[6][Var.min]=1;
               Var.elozofazis=2;
               Var.k1ben=false;
               System.out.println("Fazis 1 beillesztes");
               return Var.phUeb21;}

        //fázis3 előrehozása
        if (Var.k3ban && Var.ab_f21 && Var.ab_f22){
         Var.bejelentkezesek[6][Var.min]=1;
```

```
          Var.k3ban=false;
          System.out.println("Fazis 3 elorehozasa");
          return Var.phUeb23;}

          //fázis4 beillesztése
                 if (Var.k2ben && Var.ab_f21 && Var.ab_f22)
                {Var.beillesztes=true;
                Var.bejelentkezesek[6][Var.min]=1;
                Var.elozofazis=2;
                Var.k2ben=false;
                System.out.println("Fazis 4 beillesztese");
                return Var.phUeb24;}


                 System.out.println("2.fazis fut; fazis ideje: " +Var.phase2.getPhasenSek());
          if  (Var.phase2.getPhasenSek()>10) {
              return Var.phUeb23;
          }
          return KEINE_UMSCHALTUNG;
    }
}
```

Phase3.java

```
package budapest;

import sg.*;
import vt.*;

public class Phase3 extends Phase {

    public Phase3(String name, int nr) {
        super(name, nr);
    }

    public Phase phasenFunktion() {

        MinSearch.getBejelentkezes();

   //fázis1 beillesztése
        if (Var.k1ben && Var.ab_k3)
          {Var.beillesztes=true;
          Var.bejelentkezesek[6][Var.min]=1;
          Var.elozofazis=3;
          Var.k1ben=false;
          System.out.println("Fazis 1 beillesztes");
          return Var.phUeb31;}

        //zöld nyújtás
        if(Var.k3ban && Var.phase3.getPhasenSek()>Var.zoldk1k3-10)
         {  Var.ido1=10-(25-Var.phase3.getPhasenSek());
                Var.ido2=Var.phase3.getPhasenSek()-Var.ido1;
                Var.phase3.setPhasenSek(Var.ido2);
                        Var.bejelentkezesek[6][Var.min]=1;
                        System.out.println("Fazis 3 nyujtas");
                Var.k3ban=false;};


    //fázis4 előrehozása
        if (Var.k2ben && Var.ab_k3){
          Var.bejelentkezesek[6][Var.min]=1;
          Var.k2ben=false;
          System.out.println("Fazis 4 elorehozas");
          return Var.phUeb34;}

        if ((Var.beillesztes==true) && (Var.phase3.getPhasenSek()>10)){
           Var.beillesztes=false;
           if (Var.elozofazis==1){
                  Var.elozofazis=0;
                  if (Var.an_f21||Var.an_f22){
                         return Var.phUeb32;
                  }
                   }
           if (Var.elozofazis==4){
                 Var.elozofazis=0;
```

```
                return Var.phUeb31;
            }

         }

        if (Var.k3ban && Var.phase3.getPhasenSek()<=Var.zoldk1k3-10){
                Var.k3ban=false;
                System.out.println("Nem kell nyujtani");
        }


        System.out.println("3.fazis fut; fazis ideje: " +Var.phase3.getPhasenSek());
        if  (Var.phase3.getPhasenSek()>25) {
           return Var.phUeb34;
       }

       return KEINE_UMSCHALTUNG;
  }
}
```

Phase4.java

```
package budapest;

import sg.*;
import vt.*;

public class Phase4 extends Phase {

    public Phase4(String name, int nr) {
        super(name, nr);
    }

    public Phase phasenFunktion() {

        MinSearch.getBejelentkezes();

        // fázis1 előrehozása
        if (Var.k1ben && Var.ab_k2)
          {Var.bejelentkezesek[6][Var.min]=1;
          Var.k1ben=false;
          System.out.println("Fazis 1 elorehozasa");
          return Var.phUeb41;}

        //zöld nyújtás
        if (Var.k2ben && Var.phase4.getPhasenSek()>Var.zoldk2-8)
          {Var.ido1=8-(15-Var.phase4.getPhasenSek());
                Var.ido2=Var.phase4.getPhasenSek()-Var.ido1;
                Var.phase4.setPhasenSek(Var.ido2);
          Var.bejelentkezesek[6][Var.min]=1;
          System.out.println("Fazis 4 nyujtas");
          Var.k2ben=false;}

        // fázis3 beillesztése
        if (Var.k3ban && Var.ab_k2)
          {Var.beillesztes=true;
          Var.bejelentkezesek[6][Var.min]=1;
          Var.elozofazis=4;
          Var.k3ban=false;
          System.out.println("Fazis 3 beillesztes");
          return Var.phUeb43;}


        if ((Var.beillesztes==true) && (Var.phase4.getPhasenSek()>8)){
           Var.beillesztes=false;
           if (Var.elozofazis==1){
                  Var.elozofazis=0;
                if (Var.an_f21||Var.an_f22){
                          return Var.phUeb42;
                  }
                  return Var.phUeb43;
           }

           if (Var.elozofazis==2){
                  Var.elozofazis=0;
                  return Var.phUeb43;
```

```
          }

        }


        if (Var.k2ben && Var.phase4.getPhasenSek()<=Var.zoldk2-8){
               Var.k2ben=false;
               System.out.println("Nem kell nyujtani");
        }

        System.out.println("4.fazis fut; fazis ideje: " +Var.phase4.getPhasenSek());
         if  (Var.phase4.getPhasenSek()>15) {
             return Var.phUeb41;
         }
         return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb12.java

```
package budapest;

import vt.*;
import sg.*;

public class PhUeb12 extends Phase {

  public PhUeb12(String name, int nr) {
      super(name, nr);
  }

  public Phase phasenFunktion() {

         Var.k1.setSg( Zustand.GESPERRT ,  0 );
         Var.f21.setSg( Zustand.FREI, 6);
         Var.f22.setSg( Zustand.FREI, 6);

         if ((Var.f21.getZustand() == Zustand.GRUEN) && (Var.f22.getZustand() ==
Zustand.GRUEN) && (Var.phUeb12.getPhasenSek() >= 7) ) {
            return Var.phase2;
         }
         return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb13.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb13 extends Phase{

         public PhUeb13(String name, int nr) {
              super(name, nr);
          }
         public Phase phasenFunktion() {

         Var.k1.setSg( Zustand.GESPERRT ,  0 );
         Var.k3.setSg( Zustand.FREI, 5);

         if ((Var.k3.getZustand() == Zustand.GRUEN) && (Var.phUeb13.getPhasenSek() >= 6) ) {
            return Var.phase3;
         }
         return KEINE_UMSCHALTUNG;
  }
}
```

phUeb14.java

```
package budapest;
import sg.Zustand;
import vt.*;
```

```
public class PhUeb14 extends Phase{


        public PhUeb14(String name, int nr) {
              super(name, nr);
}
        public Phase phasenFunktion() {

         Var.k1.setSg( Zustand.GESPERRT ,  0 );
         Var.k2.setSg( Zustand.FREI     ,  6 );

         if ((Var.k1.getZustand() == Zustand.ROT) && (Var.phUeb14.getPhasenSek() >= 7) ) {
            return Var.phase4;
         }
         return KEINE_UMSCHALTUNG;
        }
}
```

PhUeb21.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb21 extends Phase{

         public PhUeb21(String name, int nr) {
              super(name, nr);
          }

         public Phase phasenFunktion() {


          Var.k1.setSg( Zustand.FREI ,  10 );
          Var.f21.setSg( Zustand.GESPERRT    ,  0 );
          Var.f22.setSg( Zustand.GESPERRT    ,  0 );

          if ((Var.k1.getZustand() == Zustand.GRUEN)&& (Var.phUeb21.getPhasenSek() >= 11) ) {
             return Var.phase1;
          }
          return KEINE_UMSCHALTUNG;
  }

}
```

PhUeb23.java

```
package budapest;
import vt.*;
import sg.*;

public class PhUeb23 extends Phase {

  public PhUeb23(String name, int nr) {
      super(name, nr);
  }

  public Phase phasenFunktion() {


          Var.k3.setSg( Zustand.FREI ,  7 );
          Var.f21.setSg( Zustand.GESPERRT    ,  0 );
          Var.f22.setSg( Zustand.GESPERRT    ,  0 );

          if ((Var.k3.getZustand() == Zustand.GRUEN)&& (Var.phUeb23.getPhasenSek() >= 8) ) {
             return Var.phase3;
          }
          return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb24.java

```
package budapest;
import vt.*;
import sg.*;

public class PhUeb24 extends Phase {

  public PhUeb24(String name, int nr) {
      super(name, nr);
  }


  public Phase phasenFunktion() {

         Var.f21.setSg( Zustand.GESPERRT ,  0 );
         Var.k2.setSg( Zustand.FREI     ,  14 );
         Var.f22.setSg( Zustand.GESPERRT ,  0 );

         if ((Var.k2.getZustand() == Zustand.GRUEN)&&(Var.phUeb24.getPhasenSek() >= 15) ) {
            return Var.phase4;
         }
         return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb31.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb31 extends Phase{

        public PhUeb31(String name, int nr) {
              super(name, nr);
          }


         public Phase phasenFunktion() {

         Var.k3.setSg( Zustand.GESPERRT ,  0 );
         Var.k1.setSg( Zustand.FREI, 9);

         if ((Var.k1.getZustand() == Zustand.GRUEN) && (Var.phUeb31.getPhasenSek() >= 10) ) {
            return Var.phase1;
         }
         return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb32.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb32 extends Phase{

        public PhUeb32(String name, int nr) {
              super(name, nr);
          }

        public Phase phasenFunktion() {

         Var.k3.setSg( Zustand.GESPERRT ,  0 );
         Var.f21.setSg( Zustand.FREI, 10);
         Var.f22.setSg( Zustand.FREI, 10);

        if ((Var.f21.getZustand() == Zustand.GRUEN) && (Var.f22.getZustand() == Zustand.GRUEN)
&& (Var.phUeb32.getPhasenSek() >= 11) ) {
            return Var.phase2;
         }
```

```
        return KEINE_UMSCHALTUNG;
 }

}
```

PhUeb34.java

```
package budapest;

import vt.*;
import sg.*;

public class PhUeb34 extends Phase {

  public PhUeb34(String name, int nr) {
      super(name, nr);
  }


  public Phase phasenFunktion() {

        Var.k3.setSg( Zustand.GESPERRT ,  0 );
        Var.k2.setSg( Zustand.FREI, 6);

        if ((Var.k2.getZustand() == Zustand.GRUEN) && (Var.phUeb34.getPhasenSek() >= 7) ) {
           return Var.phase4;
        }
        return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb41.java

```
package budapest;

import vt.*;
import sg.*;

public class PhUeb41 extends Phase {

  public PhUeb41(String name, int nr) {
      super(name, nr);
  }

  public Phase phasenFunktion() {

        Var.k1.setSg( Zustand.FREI     ,  6 );
        Var.k2.setSg( Zustand.GESPERRT ,  0 );

        if ((Var.k1.getZustand() == Zustand.GRUEN) && Var.phUeb41.getPhasenSek() >= 7)  {
           return Var.phase1;
        }
        return KEINE_UMSCHALTUNG;
  }
}
```

PhUeb42.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb42 extends Phase{

       public PhUeb42(String name, int nr) {
             super(name, nr);
         }

       public Phase phasenFunktion() {

        Var.k2.setSg( Zustand.GESPERRT ,  0 );
        Var.f21.setSg( Zustand.FREI, 8);
```

```
        Var.f22.setSg( Zustand.FREI, 8);

        if ((Var.f21.getZustand() == Zustand.GRUEN) && (Var.f22.getZustand() == Zustand.GRUEN)
&& (Var.phUeb42.getPhasenSek() >= 9) ) {
            return Var.phase2;
        }
        return KEINE_UMSCHALTUNG;
 }

}
```

PhUeb43.java

```
package budapest;
import sg.Zustand;
import vt.*;

public class PhUeb43 extends Phase{

        public PhUeb43(String name, int nr) {
              super(name, nr);
          }

        public Phase phasenFunktion() {

         Var.k3.setSg( Zustand.FREI     ,  6 );
         Var.k2.setSg( Zustand.GESPERRT ,  0 );

         if ((Var.k3.getZustand() == Zustand.GRUEN) && (Var.phUeb43.getPhasenSek() >= 7))  {
            return Var.phase3;
         }
         return KEINE_UMSCHALTUNG;
 }


}
```

Pruefung.java

```
package budapest;

import sg.*;
import vt.*;
import hw.*;
import java.util.Vector;

public class Pruefung {

  //Membervariablen

  private boolean[] Laufkennung;
  private static Sg[] gruppen;
  private AllesRot allesRot;
  private EinProg einProg;
  private BlinkProg blinkProg;
  private AusProg ausProg;
  private LaufProg laufProg;
  private Phase phase1;
  private Phase phase2;
  public static TeilKnoten pruefKnoten;

  SgProg[] progs = new SgProg[Var.MAXSG];
  public static Anlage anlage ;
  private int alterIndex;

  public Pruefung() {
  }
  public static void main(String[] args) {
    Pruefung Pruefung1 = new Pruefung();
    Init Initialisierung1 = new Init();

    enp.Actros.registerInitialisierung(Initialisierung1);
    Initialisierung1.initialisiereSg();
    Initialisierung1.initialisiereZwz();
```

```
    Initialisierung1.initialisiereDet();
    Initialisierung1.initialisiereZentrale();

//                   ||       Aktuellen Teilknoten
//                   ||
//                  \  /      eintragen!!!
//                   \/
    pruefKnoten = Var.tk1.getTeilKnoten(1);//Hier muss der aktuelle Teilknoten eingetragen
werden
    Pruefung1.initialisiereProgs();
    Pruefung1.initialisierePhasen();
    Initialisierung1.initialisiereHW();
    gruppen = Sg.getSgArray();
    }

    //innere Klasse Sg Schaltprogramm
    public class SgProg extends LogikProg {
      private int prognr;

      public SgProg(TeilKnoten kn, String name, int num, int umlZeit,
                  int gwpa, int gwpb, int warteZeit, int versatzZeit) {
        super(kn, name, num, umlZeit, gwpa, gwpb, warteZeit, versatzZeit);
      }

      public void programmFunktion (){

        try{

        }
        catch(Exception e){
          System.out.println("Exception aufgetreten in SgProg");
          e.printStackTrace();
        }
      }//Ende programmFunktion
    }//Ende class SgProg

    //Phase1
    public class Phase1 extends Phase {

      public Phase1(String name, int nr) {
        super(name, nr);
     }
     private boolean wechsel;

      public Phase phasenFunktion() {

          wechsel = schalteAllesRot();
          if (wechsel) {
            return phase2;
          }
        return KEINE_UMSCHALTUNG;
      }
    }//Ende Phase1

    //Phase2
    public class Phase2 extends Phase {

      public Phase2(String name, int nr) {
        super(name, nr);
     }
       private boolean wechsel;

      public Phase phasenFunktion() {

          int sgindex = ((pruefKnoten.getAktProg().getNummer())-1);
          if (sgindex != alterIndex) { System.out.println("Einschalten: Gruppe"+(sgindex+1));
alterIndex = sgindex; }
          if  (gruppen[sgindex].getAbzulMindSek()<=0) {
              if  (gruppen[sgindex].getAbzulZwzSek() == 0) {
                  gruppen[sgindex].setSg(Zustand.FREI, Sg.SCHALTE_SOFORT);
              }
          }
        return KEINE_UMSCHALTUNG;
      }
    }//Ende Phase1
```

```
//innere Klasse LaufProg
//wird erst fertig implementiert, wenn optimiertes Schalten möglich ist
private class LaufProg extends FestProg{
  public LaufProg(TeilKnoten tk, String name, int nr,
                  int tu, int gwpa, int gwpb, int tw, int tv) {
    super(tk, name, nr, tu, gwpa, gwpb, tw, tv);
  }

  public void programmFunktion()  {

  }
  private void AlleEin(){
    int i = 0;
    while (Laufkennung [i]) {
      i++;
      if (i == Var.MAXSG) {
        //alle Gruppen waren einmal frei
        while (i > 0){
          i--;
          Laufkennung[i] = false;
        }
        //Grundstellung
        allesRot.programmFunktion();
      }
    }
   }
   //2" nach Freigabe der vorherigen Gruppe naechste freischalten


}// Ende class LaufProg


//innere Klasse Alles Rot
private class AllesRot extends FestProg{
  public AllesRot(TeilKnoten tk, String name, int nr,
           int tu, int gwpa, int gwpb, int tw, int tv) {
    super(tk, name, nr, tu, gwpa, gwpb, tw, tv);
  }
  public void programmFunktion () {
    try{
      schalteAllesRot();
    }
    catch(Exception e){
         e.printStackTrace();
    }
  }
}//Ende class Alles Rot


//innere Klasse EinProgramm
private class EinProgramm extends EinProg {

  public EinProgramm(TeilKnoten knoten, String name,int nr, int tu) {
    super(knoten, name, nr, tu);
  }

  public void programmFunktion (){

    schalteAllesRot();

  }

}//Ende EinProgramm

//innere Klasse BlinkProgramm
private class BlinkProgramm extends BlinkProg{

  public BlinkProgramm(TeilKnoten tk,String name, int nr, int tu) {
    super(tk, name,nr, tu);
  }

  public void programmFunktion (){

  }

}//Ende BlinkProgramm
```

```
    //innere Klasse AusProgramm
    private class AusProgramm extends AusProg{

      public AusProgramm(TeilKnoten tk, String name, int nr, int tu) {
        super(tk, name, nr, tu);
      }

      public void programmFunktion (){

        allesRot.programmFunktion();
      }
    }//Ende AusProgramm


    //Methode initialisiereProgs, hier werden die Programme initialisiert
    private void initialisiereProgs (){
      int i;

                    //Knoten    | Prg.   |Prg |Umlauf|GWPA|GWPB|Koord| W
                       Name     | Name   | Nr.|Zeit  |    |    | Zeit| Vsatz
     for (i=0; i<Var.MAXSG; i++){
progs[i]=new SgProg(pruefKnoten,"Prog"+i,i+1 ,  1   ,   0,   1,    0, 0);}
allesRot=new AllesRot(pruefKnoten,"AllesRot"n,i+1, 1  , 0,   1,    0, 0);
laufProg= ew LaufProg(pruefKnoten,"LaufProg", i+2 ,1 ,  0,   1 ,   0, 0);
einProg=new EinProgramm(pruefKnoten,"EinProg",   i+3,   1);
blinkProg=new BlinkProgramm(pruefKnoten,"BlinkProg",i+4 ,1);
ausProg=new AusProgramm(pruefKnoten,"AusProg"  , i+5,   1);

      Vt.endInit();
    }//Ende intialisiereProgs

    //Mthode initialisierePhasen, um die benötigten Phasen zu initialisieren
    private void initialisierePhasen (){
      //Phasen
      this.phase1 = new Pruefung.Phase1("Phase1", 1);
      this.phase2 = new Pruefung.Phase2("Phase2", 2);

    }//Ende initialisiere Phasen


    //Methode um alle Gruppen auf "gesperrt" zu setzen. Liefert solange "false" solange nicht
alle Gruppen gesperrt sind
    private boolean schalteAllesRot (){
      int i = 0;
      boolean alleRot = true;
      try {
        for (i=0; i<Var.MAXSG; i++){
          if  (gruppen[i].getZustand()<10) {                    //alle Werte <10 sind
Freizustaende
              if  (gruppen[i].getAbzulMindSek() <= 0) {
                  gruppen[i].setSg(Zustand.GESPERRT, Sg.SCHALTE_SOFORT);
                  alleRot = false;
              }
              alleRot = false;
          }
          else if  (gruppen[i].getZustand()>19)  {              //alle Werte >19 sind
Uebergangszustaende
                   alleRot = false;
          }
        }
        return alleRot;
      }
      catch(Exception e){
        System.out.println("Exception aufgetreten in AllesRot!");
        e.printStackTrace();

      }
      return false;
    }
    public String getVersion (){
        return "Budapest - Version 1.0, 21.06.2006";
    }

}
```

Var.java

```
package budapest;

//Import der Packages deren Methoden benoetigt werden
    import sg.*;
    import lmp.*;
    import hw.*;
    import vt.*;
    import det.*;
    import sg.typen.*;

    import java.util.Vector;

public class Var
{
    //Signalgruppen
    public static Sg k1,k2,k3,f21,f22;

    public static final int MAXSG = 5;

    //Wiederholer
    public static Wiederholer k1a,k2a,k3a,k3b;
    public static Wiederholer f21a,f22a;

    //Detektoren
    public static Detektor t21,t22;

  //Detektor kimenetek az OCIT-hoz
    public static Detektor d01;
    public static Detektor d02;
    public static Detektor d03;
    public static Detektor d04;

    //Ausgaenge
    public static Ausgang sk21, sk22, out1, out2, out3;

    //Teilknoten
    public static TeilKnoten tk1;

    //Zentrale

    //Debugmodul

    //Übergeordnete Methoden
    public static MyMainFkt main;
    public static MyPostMainFkt postmain;
    public static MyUmschaltFkt umschalt;

    //Systemprogramme: Ein-, Aus-, Blinkprogramm
    public static MyEinProgramm einprog;
    public static MyAusProgramm ausprog;
    public static MyBlinkProgramm blinkprog;

    //weitere Programme (Festzeit- und Logikprogramme)
    public static BusProg   BusProg1;


    //Phasen und Phasenuebergaenge
    public static Phase phase1;     // Phase1
    public static Phase phase2;     // Phase2
    public static Phase phase3;     // Phase3
    public static Phase phase4;     // Phase4

    public static Phase phUeb12;   // Uebergang 1->2
    public static Phase phUeb23;   // Uebergang 2->3
    public static Phase phUeb24;   // Uebergang 2->4
    public static Phase phUeb34;   // Uebergang 3->4
    public static Phase phUeb41;   // Uebergang 4->1

    public static Phase phUeb13;
    public static Phase phUeb14;
    public static Phase phUeb21;
    public static Phase phUeb31;
    public static Phase phUeb32;
    public static Phase phUeb42;
```

```
public static Phase phUeb43;


//Lampentypen
public static LmpTyp lrot   = new LmpEinfachUga();
public static LmpTyp lgelb  = new LmpBlinkUge();
public static LmpTyp lgruen = new LmpEinfachUge();

//SgTypen
public static SgTyp kfz   = RoGeGn.init(lrot, lgelb, lgruen, 1, 3, 5);
public static SgTyp fg    = RoGn.init(lrot, lgruen,false);

//weitere Variablen wie Merker, etc.
//sonstige Globale Variablen
public static boolean an_k2, an_f21, an_f22, an_nr;
public static boolean ab_k1, ab_k2, ab_k3, ab_f21, ab_f22;
public static boolean umschalten;

//Buszos program új változói
  public static boolean k1ben, k2ben, k3ban, ocit1, ocit2, ocit3, beillesztes, v;
  public static float utasszam, keses, hhk, lgy, pontszam;
  public static float bejelentkezesek[][]=new float [7][20];
  public static int szamlalo, szamlalo2, szamlalomin, min, elozofazis, x, ido1, ido2;
  public static int zoldk1k3=25;
  public static int zoldk2=15;
  public static float vizsgalt[]=new float[20];


public static Anlage anlage = null;
```

Referenciák

[1] Luspay Tamás - Tettamanti Tamás - Dr. Varga István: Közúti közlekedési automatika; p151
http://www.kka.bme.hu/images/stories/targyak/kozutaut1/kzuti_automatika_ver2.pdf

[2] Angus P. Davol: Modeling of traffic signal control and transit signal priority strategies in a microscopic simulation laboratory; p118

http://web.mit.edu/its/papers/TRAFF.PDF

[3] Polgár János: A közforgalmú közlekedés előnybiztosításának vizsgálata a társadalmi, klimatikus költségek tükrében (TDK dolgozat 2009, Konzulens Dr. Török Ádám tud. s.m.) p37

[4] Signalbau Huber GmbH: Steuergerätfamilie Actros: Systembeschreibung; p53

[5] Tettamanti Tamás: Actros VTC 3000 (Tanszéki segédlet); p11
http://www.kka.bme.hu/images/stories/targyak/kozutir2/actros_vtc_3000.pdf

[6] Uwe Wilbrand: Aufbau einer VT in Java; p69

[7] Tettamanti Tamás – Luspay Tamás – Varga István: Forgalomirányító rendszerek zárt hurkú szimulációja; Innováció és fenntartható felszíni közlekedés konferencia 2008, előadás; p4
http://kitt.bmf.hu/mmaws/2008/eloadasok/1-szekcio/tettamanti-luspay-varga.pdf