Heurisztikus algoritmusok alkalmazása VLSI huzalozásra

– Tudományos diákköri dolgozat –

Hajnács Zoltán III. Inf.,
Kocsis Imre III. Inf.,
Ritter Ádám II. Inf.,
Szabó Péter IV. Inf.

Konzulensek:
Dr. Recski András, Számítástudományi és Információelméleti Tanszék,
Mann Zoltán, Irányítástechnika és Informatika Tanszék

Budapest, 2002. október

Tartalomjegyzék

1. Kivonat

A VLSI áramkörök tervezésének fontos része a részletes huzalozás: az egyes áramköri elemek közti összeköttetések minél kisebb helyigényű megvalósítása. Ez kulcsfontosságú a mai modern eszközök tervezésénél, ahol részben a méret függvénye a hőtermelés és az áramfogyasztás, a gyorsaság, valamint az anyagköltség. A feladat a gyakorlatilag érdekes esetekben szinte mindig NP-nehéz, ami a jelenlegi ismereteink alapján nagyon valószínűsíti, hogy nem található rá (elég) gyors algoritmus.

Természetesen vetődik fel a kérdés a huzalozási problémákkal kapcsolatban, hogy alkalmas módon megválasztott általános illetve problémaspecifikus heurisztikák használatával kapott megoldások mennyire esnek távol az elméleti optimumtól – a szükséges számítási idő drasztikus csökkenése mellett.

A dolgozatban a 2001. évi TDK konferencia *Huzalozó algoritmusok implementációja és vizsgálata* c. dolgozatában bemutatott szoftverrendszer felhasználásával az ezen kérdésekre adható válaszokkal foglalkoztunk. Ez a rendszer lehetővé teszi algoritmusok implementációját, empirikus vizsgálatát.

Több módszert fejlesztettünk ki, ezeket és egyéb ismert algoritmusokat implementáltunk és hasonlítottunk össze.

Kidolgoztunk egy általánosan alkalmazható módszert, amellyel egy VLSI problémát vissza tudunk vezetni egy kielégíthetőségi problémára. Ennek segítségével kétrétegű Manhattan-modellben megfogalmazott problémákat megoldó algoritmust fejlesztettünk ki. Szintén ebben a modellben implementáltuk a témakör egyik klasszikusnak mondható heurisztikáját, a Rivest és Fiduccia által javasolt csatornahuzalozási algoritmust, és ennek háromrétegű, illetve switchbox-huzalozási változatait, valamint a Gallai algoritmus háromrétegű Manhattan modellben csatornahuzalozásra alkalmazható változatát is. Végül egy általános célú heurisztikát, egy genetikus algoritmust implementáltunk, kétrétegű Manhattan modellben.

A vizsgált modelleket a gyakorlatban való elterjedtségük, valamint bonyolultságelméleti érdekességük miatt választottuk ki.

Az implementációk sajátosságaik miatt nem feltétlenül adnak mindig optimális eredményt, ezért ennek vizsgálata ugyanúgy tárgya a dolgozatnak, mint a már ismert hatékonyságú algoritmusokkal való összehasonlítás.

2. A fő eredmények áttekintése

Nagy bonyolultságú áramkörök egy fontos feladata a részletes VLSI huzalozás. Ehhez fejlesztettünk ki, és implementáltunk algoritmusokat. Az implementációt egy már meglévő rendszerbe [2] illesztettük be. A rendszer már tartalmazott néhány algoritmust:

- a Gallai nevéhez fűződő optimális egysoros algoritmust [3]
- a Recski–Strzyzewski féle csatorna-huzalozási algoritmust [12]
- egy Szeszlér D. által kifejlesztett switchbox algoritmust [24]

Imlementáltunk további algoritmusokat:

- a Gallai algoritmus csatorna-huzalozásra használt változatát [3]
- egy általunk kifejlesztett SAT solver alapú algoritmust
- két Prolog alapú algoritmust a SAT problémára való visszavezetést felhasználva
- egy általunk kifejlesztett genetikus huzalozó algoritmust
- A Rivest-Fiduccia switchbox és háromrétegű csatorna-huzalozó algoritmust

Sajnos a Rivest-Fiduccia és a genetikus algoritmusok implementációja további finomításra szorul, így ezen TDK keretében nem tudtuk őket tesztelni.

A dolgozat 3. fejezetében áttekintjük a részletes huzalozás fogalmát. A részletes huzalozást megnézzük általában, és látunk a többrétegű modell felhasználásával néhány valódi huzalozási problémát. Megvizsgáljuk a leggyakrabban alkalmazott technológiákat és geometriákat.

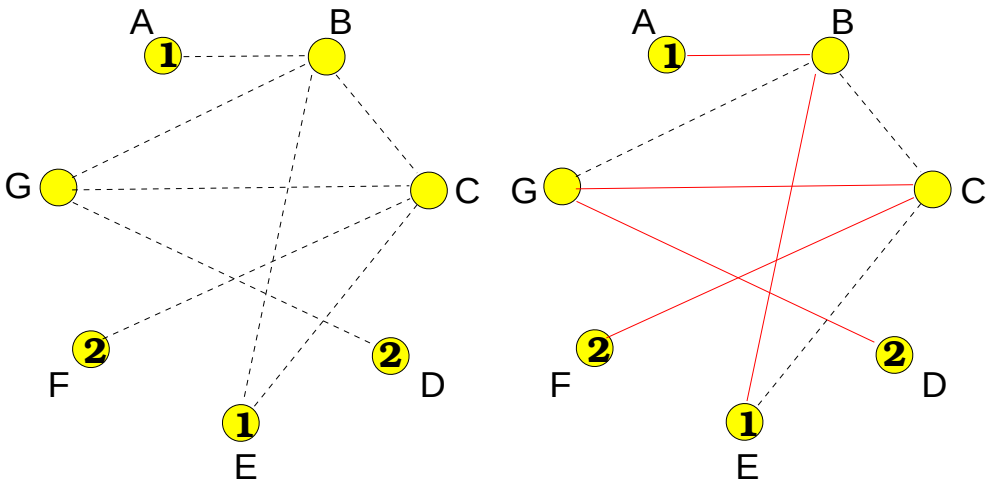
A 4. fejezetben adunk egy új algoritmust a huzalozási probléma kielégíthetőségi problémára való visszavezetésére. Az algoritmus segítségével meg tudjuk állapítani, hogy egy heurisztikus huzalozási algoritmus által generált megoldás milyen éles, mennyire áll közel az optimális megoldáshoz, és így használhatjuk egy algoritmus hatékonyságának a mérésére.

A 5. fejezetben bemutatjuk a Rivest-Fiduccia algoritmust, amely egy hatékony mohó algoritmus, ami használható switchbox- és három rétegű csatorna-huzalozásra is.

A 6. fejezetben adunk egy új genetikus algoritmust a csatorna-huzalozási probléma megoldására, és leírjuk az eddigi implementációs tapasztalatokat is. Bár a genetikus algoritmust nem konkrét cikk felhasználásával terveztük meg, mindenképpen segítségünkre volt az irodalomból nyert általános tapasztalat. Az algoritmus szoftverrendszere azonban teljes mértékig a szerzők saját munkája volt, semmiféle külső irodalmat, segítséget nem használtak fel hozzá.

A 7. fejezetben leírjuk az implementált algoritmusok eredményét.

Végül a 8. fejezetben összesítjük, hogy mire használhatja ezt a dolgozatot bárki, aki érdeklődik a heurisztikus VLSI huzalozó algoritmusok iránt, és milyen eredményeket értünk el.



(a) (b)

1. ábra. (a) egy huzalozási feladat (b) a huzalozási feladat megoldása

3. A részletes huzalozás fogalma, gyakori megszorítások

3.1. A részletes huzalozás

A dolgozat további részében már csak részletes huzalozással foglalkozunk. A huzalozási feladat általában egy téglalap kerületén lévő *terminálok* összekapcsolása úgy, hogy előre meg van határozva, hogy melyik *lábat* melyikkel kell összekapcsolni. Azon terminálok, melyeket össze kell kapcsolni, egy *netet* alkotnak. Egy netben legalább két terminál van, de állhat kettőnél többől is. Ha mégis minden net két terminálból áll, akkor a problémát *2-net huzalozási feladatnak* hívják. Az ilyen feladatok a gyakorlatban gyakran előkerülnek, ezért igen fontosak.

A terminálokon és a belőlük alkotott neteken kívül a részletes huzalozási problémához hozzátartozik egy gráf is, ahol futhatnak a vezetékek. A megoldás ennek a gráfnak egy részgráfja lesz. Két egy netbe tartozó terminált össze kell kötni, így a megoldásban lesz köztük út. Két különböző netbe tartozó terminált nem szabad összekötni, így a megoldásban köztük nem mehet út, tehát minden nethez a kapott gráf pontosan egy komponense fog tartozni: amelyik tartalmazza a net termináljait. Felesleges költség lenne, ha olyan komponens lenne a megodásban amelyik nem tartalmaz terminált, mivel ezeket elhagyhatnánk, így feltételezhetjük, hogy a kapott komponensek egy-egy netet képviselnek.

A huzalozási probléma és megoldásának formális definíciója:

3.1. definíció. *$P = \langle V, T, N, G\rangle$ egy huzalozási probléma, ahol V a csúcsok halmaza, $T \subset V$ a terminálok halmaza, $N \subset P(T)$, $n_1, n_2 \in N \Rightarrow |n_1| \geq 2 \wedge n_1 \cap n_2 = 0$ a páronként diszjunkt, legalább két elemű netek halmaza, és G a lehetséges élek gráfja, ahol $V_G = V$.*

3.2. definíció. *Legyen $P = \langle V, T, N, G\rangle$ egy huzalozási probléma, és $G' \subset G$. Ekkor G' megoldása P-nek, ha $t_1, t_2 \in T$ pontosan akkor vannak összekötve G'-ben, ha egy netben vannak, azaz $\exists n \in N : t_1 \in n \wedge t_2 \in n$, és G' minden komponense tartalmaz terminált. P megoldásainak halmazát $Sol(P)$-vel jelöljük.*

Az 1(a) ábrán látható egy huzalozási probléma. Ilyen a valóságban nincsen, de jó a huzalozási

feladatok absztrakt értelmezésének áttekintéséhez. 6 pontja van, és ebből 4 terminál (A, D, E, F). A terminálok két netben vannak: A és E van az elsőben, D és F a másodikban. Az 1(b) ábra mutat egy megoldást: az A és E terminál összekötéséhez elegendő a B pont használata, viszont a másik nethez már két pontra van szükség. A két út diszjunkt, így ez a huzalozás valóban megoldása a feladatnak.

3.2. A többrétegű modell

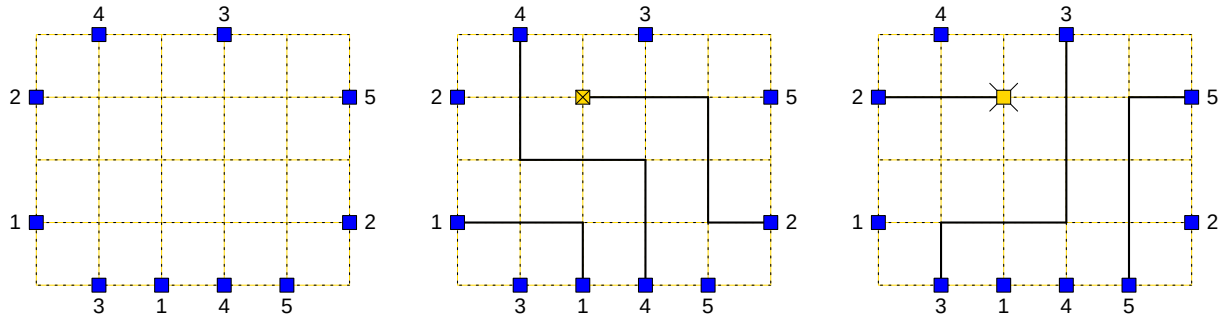
Habár a huzalozási probléma előbbi megfogalmazása tökéletesen megfelelne bármelyik valós probléma leírására, NP-nehéz, és túl általános ahhoz, hogy a huzalozáshoz szükséges leghatékonyabb heurisztikus algoritmusok leírására megfelelő legyen. A valódi feladatok ennél sokkal szűkebb problémakörből kerülnek elő, így érdemes a fent adott huzalozási problémák halmazát szűkíteni.

A leggyakrabban a részletes huzalozásra a négyzetrácsot használják: egy téglalap oldalain vannak a terminálok a rácspontokon, és a téglalapon belül futhatnak az vezetékek rácséleken keresztül. A téglalap négy oldalát északi, keleti, déli és nyugati oldalaknak nevezik. Általában nem elég egy réteg, így több réteget egymás fölé raknak, és ezek közé is összeköttetéseket hoznak létre, és így egy kockarácsot kapnak, leszámítva, hogy az alap téglalapon lévő egyes terminálok minden rétegen megjelennek, azaz a huzalozásban résztvevő téglatest oldalain bizonyos pontok topológiailag ekvivalensek.

3.3. A Manhattan modell

A többrétegű modell-beli feladatok egy részénél kikötik, hogy egy rétegen csak függőleges (észak-dél irányú), vagy vízszintes (nyugat-kelet irányú) erek mehetnek. Az előbbit függőleges (vagy V típusú), az utóbbit vízszintes (vagy H típusú) rétegnek nevezik. Ha minden réteg V vagy H típusú, és két réteg típusa különböző, akkor a feladat Manhattan modell-beli. A Manhattan modell gyartástechnikai szempontból előnyös, mert könnyebb olyan rétegeket gyártani, melyeken csak egy irányban futnak vezetékek. További előnye a Manhattan modellnek, hogy a kisebb problématér miatt a huzalozó algoritmusok gyorsabban futnak. Ugyanakkor a megoldás szélessége várhatóan nagyobb lesz, mint korlátozás nélküli modell esetén.

Vizsgáljuk meg a Manhattan modellbeli problémákat a szintek száma szerint. Egy rétegen nincs értelme egy feladatot Manhattan modellben vizsgálni, mivel ekkor csak szemben levő terminálokat tudunk összekötni. Két rétegen könnyen látható, hogy mindegy, hogy egy H típusú rétegre rakunk V típusút, vagy fordítva. Viszont három rétegen a feladat lehet VHV vagy HVH modellbeli (ilyenkor ha csak a Manhattan modell van kikötve, akkor megnézzük, hogy melyik a kedvezőbb nekünk a feladat szempontjából). Több rétegre könnyen látható, hogy ha a rétegek száma páros, akkor egyértelmű a feladat, ha páratlan, akkor kétféle Manhattan modell-beli feladattípus van (VHVH...V és HVHV...H).



2. ábra. (a) egy switchbox feladat (b) a feladat megoldása

3.4. A switchbox huzalozási probléma

Az eddigiekben nem tettünk megszorítást arra, hogy a téglalap mely oldalain lehetnek terminálok. Ha legalább három oldalán van, akkor a feladatot switchbox huzalozási problémának nevezzük. Ha pontosan három oldalon vannak terminálok, akkor nyitott switchbox huzalozási problémának. Ez egy jóval nehezebb feladat, mintha csak a téglalap egy vagy két oldalán lenne terminál elhelyezve.

A problémakör annyira nehezen megoldható, hogy nem is létezik olyan l korlát, amire legfeljebb l szintre lenne szükség minden switchbox huzalozási probléma megoldásához. Szerencsére a szükséges rétegek száma csak a négyzettől nagyon távol eső téglalapokra nagy, ha $m = max(a/b, b/a)$, egy $a \times b$ méretű probléma megoldásához elég k réteg, ahol k csak m-től függ ([24]).

Egy switchbox feladat látható megoldással a 2. ábrán. A feladatban szükség van ún. *via-ra*, rétegek közötti vezetékre (ez az ábrán a kettes netnél valósul meg).

3.5. A csatorna huzalozási probléma

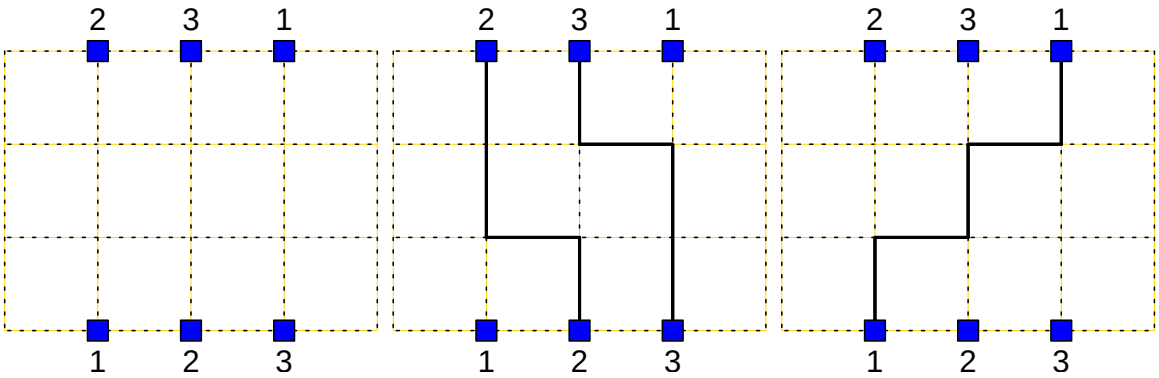
Az előző problémáknál sokkal könnyebben kezelhetők azok, amelyeknél csak két szemben lévő oldalon fordulhatnak elő terminálok. Feltételezhetjük, hogy ez a két oldal az északi és a déli, hiszen egyébként elforgathatjuk a téglalapot 90 fokkal, és egy ilyet kapunk.

Csatorna huzalozásnál valódi különbség van a VHV és a HVH típusú Manhattan modellbeli problémák között. Ez a különbség olyan nagy, hogy teljesen különböző megoldó algoritmusok léteznek a két feladattípusra. A 3. ábra egy csatorna-huzalozási problémát ábrázol.

3.6. Az egysoros huzalozási probléma

Lehetséges, hogy csak egy oldalon vannak terminálok. Ezt egysoros huzalozási problémának hívjuk.

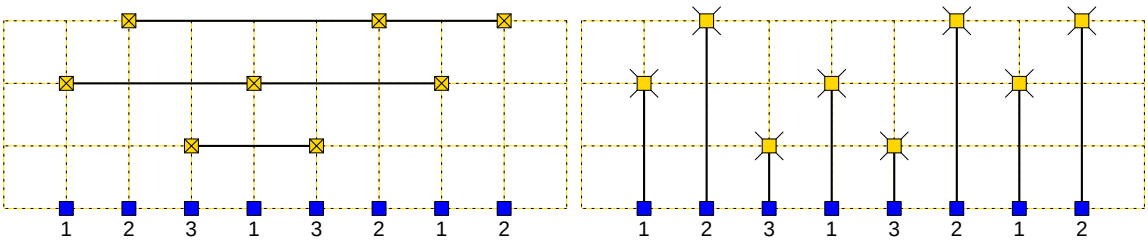
Ebben az esetben a Gallai algoritmus ([3]) minden szempontból gyors, optimális megoldást ad, így ezekkel a feladatokkal nem is kell foglalkozni. A Gallai algoritmust viszont érdemes megnézni, mivel kisebb változtatással alkalmazható a csatorna huzalozási probléma megoldására is. A 4. ábra egy egysoros huzalozási probléma megoldása a Gallai-féle (mohó) algoritmussal.



a) b)

3. ábra.

a) egy csatorna-huzalozási feladat b) megoldása kétrétegű, korlátozás nélküli modellben



4. ábra. Egysoros huzalozási probléma megoldása a Gallai-féle algoritmussal

3.7. A gammahuzalozás

Megemlítjük, hogy kimaradt egy eset, amikor két szomszédos oldalon foglalnak el helyet a terminálok, viszont ezekkel ebben a dolgozatban nem foglalkozunk, mert ritkán van rá szükség, és így mérnöki szempontból kisebb jelentőségű.

4. A huzalozási probléma visszavezetése kielégíthetőségi problémára

A VLSI huzalozás problémájára legtöbbször gyors, közvetlen, vagy gráfelméleten alapuló heurisztikákat használnak. Mi most egy más fajta algoritmust készítettünk a huzalozási probléma megoldására, amely nagy hálózatok esetén nem a leghatásosabb, viszont kisebb tesztesetekre pontosabban megkapjuk, hogy milyen problémák oldhatók meg, és ezzel a gyors heurisztikák tökéletes megoldáshoz való közelségét is mérni tudjuk. Például a most általunk adott algoritmussal megállapíthatjuk, hogy mekkora csatornára van szükség egy csatorna huzalozási probléma megoldásához, és így megtudhatjuk, hogy például a Recski-Strzyzewski féle csatorna huzalozási algoritmus mennyire éles, milyen közel van az elméleti határhoz, és azzal, hogy megkeressük, hogy milyen kis esetekre nem ad optimális megoldást, lehetőséget adunk a heurisztika legnagyobb hibáinak a kijavítására.

4.1. Szükséges definíciók

A most adott algoritmusnak az alapja a huzalozási probléma visszavezetése a *kielégíthetőségi problémára*. Ezt azért tehetjük meg, mert napjainkban nagyon jó megoldók léteznek az említett problémakörre. A visszavezetéshez nem a gyakran használt többrétegű modellt használjuk, hanem az általánosabb huzalozási problémát. Mivel itt az absztrakt feladatról adunk egy visszavezetést a *SAT problémára*, érdemes áttekinteni a huzalozási és a SAT probléma definícióját.

4.1. definíció. *$P = \langle V, T, N, G\rangle$ egy huzalozási probléma, ahol V a csúcsok halmaza, $T \subset V$ a terminálok halmaza, $N \subset P(T)$, $n_1, n_2 \in N \Rightarrow n_1 \cap n_2 = 0$ a páronként diszjunkt netek halmaza, és G a lehetséges élek gráfja, ahol $V_G = V$.*

4.2. definíció. *Legyen $P = \langle V, T, N, G\rangle$ egy huzalozási probléma, $G' \subset G$, és G' minden komponense tartalmaz terminált. Ekkor G' bővített megoldása P-nek, ha $t_1, t_2 \in T$ pontosan akkor vannak összekötve G'-ben, ha egy netben vannak, azaz $\exists n \in N : t_1 \in n \wedge t_2 \in n$. P bővített megoldásainak halmazát $Solx(P)$-vel jelöljük.*

A definíció nagyon hasonlít a huzalozási probléma megoldására. Mindössze annyiban különbözik a *bővített megoldás* a megoldástól, hogy megengedjük a G' részgráfoknak olyan komponenseit is, amik nem tartoznak egyik nethez sem. Bár minket csak a valódi megoldások érdekelnek, a bővített megoldások megengedése nem okoz valódi problémát, hiszen ha van egy bővített megoldásunk, könnyen készíthetünk belőle eredeti értelemben vett megoldást: csak el kell hagyni a kapott részgráf azon komponenseit, amelyek nem tartalmaznak terminált. Később látni fogjuk, hogy a bővített megoldások megengedése egyszerűbben visszavezethető feladatokat fog eredményezni.

Kimondhatjuk az alábbi lemmát:

4.3. lemma. *Legyen $P = \langle V, T, N, G\rangle$ egy huzalozási probléma. Ekkor P-nek akkor és csak akkor van megoldása, ha van bővített megoldása*

A lemma következik az 4.2. definícióhoz tett megjegyzésből.

4.4. definíció. *Logikai (igaz vagy hamis értékű) változók és negáltjaik* vagy *(∨) kapcsolatát klóznak nevezzük.*

4.5. definíció. *Klózok* és *(∧) kapcsolatát konjunktív normálformának (CNF) nevezzük.*

4.6. definíció (SAT). *Legyen $S = \langle X, C \rangle$ egy konjunktív normál alakban megadott kielégíthetőségi probléma (SAT[1]) ahol X a változók halmaza, és C a klózok halmaza, ahol minden $c \in C$ klóz valahány literál halmaza, ahol egy literál egy változó ponált vagy negált alakja.*

Itt feltételeztük, hogy a SAT probléma konjunktív normál alakban van megadva. Ilyen alakra akarjuk visszavezetni a huzalozási problémát, mivel a SAT megoldók konjunktív normál alakkal tudnak hatékonyan dolgozni. Természetesen lehetne egy tetszőleges logikai függvény megoldására is visszavezetni a huzalozási problémát, de ekkor a konjunktív normál formára való visszavezetésnél drasztikusan nőne a változók és a literálok száma, ami rontaná a megoldás hatékonyságát.

A visszavezetést először csak a problémák egy részére fogjuk végrehajtani. A következő definíció adja meg a vizsgált problémakört:

4.7. definíció. *Legyen $P = \langle V, T, N, G \rangle$ olyan huzalozási probléma, ahol a netek elemszáma 2 ($\forall n \in N\, |n| = 2$). Ekkor P 2-net probléma.*

Ezt a problémakört azért érdemes vizsgálni, mert a megoldások könnyen kezelhetők, és a gyakorlatban is sokszor előfordul. Az itt elmondottak általánosíthatók nagyobb elemszámú netekre is, de a visszavezetés lényeges vonásai már 2-net esetén is megmutatkoznak. Látni fogjuk, hogy ha van megoldása egy 2-net problémának, akkor erős megkötéseket tehetünk a megoldásra, és így is lesz megoldása.

4.8. definíció. *A P probléma egy bővített megoldása* szép, *ha G'-ben minden $v \in V$ csúcsnak legfeljebb 2 a foka, és v foka pontosan akkor 1, ha $v \in T$.*

Nézzük, hogy így miért igaz, hogy valóban nem szűkítjük le túlságosan P megoldásainak a halmazát.

4.9. lemma. *Legyen P 2-net huzalozási probléma. P-nek akkor és csak akkor van megoldása, ha van szép megoldása is.*

Bizonyítás. Legyen G' P egy megoldása. G' minden g komponensében pontosan 2 netbeli csúcs van (a, b), és ezek egy netben vannak (mivel ez megoldás), azaz a és b egy N-beli netet alkotnak ($\{a, b\} \in N$). Ekkor a G' gráfban nem rontjuk el az a és b pontok összeköttetését, ha g-t kicseréljük az a és b között futó g-ben lévő legrövidebb útra. Hajtsuk végre ezt a cserét G' összes komponensére. Legyen az így kapott gráf G''. Ekkor $|N|$ diszjunkt utat kapunk, ahol G''-ben a terminálok foka pontosan 1, és minden más csúcs foka 0 vagy 2, tehát P-nek van szép

[1] SATisfiability problem

megoldása. Visszafelé azért igaz az állítás, mert a szép megoldások halmaza része a bővített megoldások halmazának, így ha van P-nek szép megoldása, akkor van bővített megoldása is, de ekkor a 4.3. lemma miatt van megoldása is.

4.2. 2-net huzalozási feladat visszavezetése a SAT problémára

4.10. tétel. *Legyen $P = \langle V, T, N, G\rangle$ 2-net huzalozási probléma. Ekkor van olyan ekvivalens kielégíthetőségi probléma, ami kétrétegű Manhattan modellben nem több, mint $(\frac{3}{2} + \log h)wh$ változót és $(18 + 3\log h)wh$ literált használ.*

Bizonyítás. Az 4.9. lemmával a megoldás keresésének feladatát visszavezettük a P problémára szép megoldás keresésének feladatára. Ezek után a P-hez szép megoldás keresését kell visszavezetni egy $S = \langle X, C\rangle$ kielégíthetőségi problémára. Ehhez egy P 2-net huzalozási problémához úgy rendelünk hozzá egy S kielégíthetőségi problémát, hogy P szép megoldásai és S megoldásai között legyen kölcsönösen egyértelmű megfeleltetés.

A SAT problémára visszavezetéshez fel kell vennünk változókat és klózokat. Először nézzük meg, hogy milyen változókra lesz szükségünk.

Használt változók. Vegyünk fel minden $e \in E_G$ élre egy x_e élindikátor változót ($|E_G|$ változó). Az x_e akkor legyen igaz, ha az e él benne van P szép megoldásában. Az x_e változók megadása elég a G' megoldás leírásához, viszont a probléma megoldásainak leírásához további változókra vannak szükségeink.

Vegyünk fel minden $v \in V\backslash T$ nem terminálbeli csúcsra egy x_v csúcsindikátor változót ($|V\backslash T|$ változó). Egy x_v változó akkor legyen igaz, ha v a G' megoldásban egy netben lévő két terminál között lévő úton lévő egyik csúcs, vagy egy G'-ben lévő körön lévő csúcs.

Minden $v \in V$ csúcsra vegyünk föl k változót $(x_{v1}, x_{v2}, \ldots, x_{vk})$, ahol $k = \lceil \log |N| \rceil$ a netek számának a logaritmusa ($k|V|$ változó). Rendeljük az $|N|$ darab nethez a $0, 1, \ldots, N-1$ számokat. Ha v egy net két terminálja között lévő úton van, akkor jelölje $net(v)$ a megfelelő net sorszámát, és legyen $x_{v1}, x_{v2}, \ldots, x_{vk}$ a $net(v)$ szám binárisan kódolva. Ez lehetséges, hiszen a k számjeggyel legnagyobb kódolható szám $2^k - 1$, és mivel $k = \lceil \log |N| \rceil$, ezért a legnagyobb kódolható szám $2^k - 1 = 2^{\lceil \log |N| \rceil} - 1 \geq 2^{\log |N|} - 1 = |N| - 1$, azaz |N|-1 még kódolható, tehát az összes net sorszám kódolható. Ezen új változókkal le tudjuk tárolni, hogy az egyes csúcsok a gráf melyik komponensében vannak, és így biztosítani tudjuk majd, hogy ne legyen összekötve két nem egy netben lévő csúcs. Ehhez arra lesz szükség, hogy bármely két szomszédos csúcs egy netben legyen.

Az eddigi változókból úgy tűnhet, hogy nem használjuk ki sehol sem, hogy a huzalozásban a netek mérete csak 2 lehet. A klózokat megadhatjuk, és végül kapunk egy megoldást a huzalozási problémára. Viszont azt is biztosítani kell, hogy két ugyanazon netbeli terminál mindenképpen össze legyen kötve. És ezt csak két terminál esetén lehet ezekkel a változókkal megoldani, azt kihasználva, hogy szép megoldáshoz viszonylag könnyű fölírni a klózokat (lásd alább).

Klózok. Az a következő feladatunk, hogy megfelelő klózokkal biztosítsuk, hogy csak szép megoldásnak megfelelő megoldás álljon elő, de meg kell engednünk minden szép megoldást, te-

hát nem szabad szigorúbbaknak lennünk a kelleténél. Egy szép megoldáshoz az alábbi feltételek kellenek:

– egy csúcsnak, ami nem terminál, kettő vagy nulla legyen a fokszáma a megoldásban
– egy terminálnak épp egy legyen a foka a megoldásban
– ha egy él benne van a megoldásban (x_e igaz), akkor az él két csúcsának is igaznak kell lenni.
– egy él két csúcsához ugyanazt a netszámot kell rendelni.
– egy terminál netszámának a terminál sorszámával meg kell egyezni

Az első két feltétel a szép megoldás tulajdonsága, a többi pedig a változók jelentéséből következik. Ha ezek a feltételek teljesülnek, egy olyan gráfot kapunk, amelyben minden pont foka legfeljebb 2, így csak utakból és körökből állnak. A körök nem mehetnek át terminálon, hiszen minden terminálnak 1 a foka, tehát a körök nem befolyásolják a megoldás helyességét. Az utak pedig terminálból terminálba mennek nem terminál pontokon keresztül (a fokszámokból következik), és az út minden csúcsához ugyanaz a netszám van rendelve, így a két terminálhoz is ugyanaz a fokszám volt rendelve eredetileg, tehát ha két terminál össze van kötve, akkor azok egy netben vannak. Viszont minden terminál össze van kötve egy másikkal, hiszen minden terminálból indul ki út. Emiatt egy szép megoldását kapjuk a P problémának, ha teljesítjük a feltételeket.

Ezek után már csak fel kell írnunk a feltételeknek megfelelő klózokat:

– egy csúcsnak, ami nem terminál, 0 vagy 2 legyen a fokszáma a G' gráfban. ($|N\backslash T|(3\binom{\Delta_{nt}}{2} + 4\binom{\Delta_{nt}}{3})$ literál)
Legyenek a v csúcsból kiinduló élek $e_1, e_2, \ldots, e_l$.

$$\forall\, 1 \leq i_1 < i_2 \ldots < i_{l-1} \leq l \;\; (\overline{x}_v \vee x_{e_{i_1}} \vee x_{e_{i_2}} \vee \ldots \vee x_{e_{i_{l-1}}}) \wedge$$

$$\wedge\, \forall\, 1 \leq i < j < k \leq l \;\; (\overline{x}_v \vee \overline{x}_{e_i} \vee \overline{x}_{e_j} \vee \overline{x}_{e_k})$$

Megjegyzés: ezt nagy l-ekre ($l > 4$) érdemesebb úgy csinálni, hogy tároljuk binárisan a két értéket, és azokkal hasonlítjuk össze a számot. Ha pl. v fokszáma 3, az alábbi klózokat kapjuk: Legyenek e_1, e_2, e_3 a v csúcsból kiinduló élek.

$$(\overline{x}_v \vee x_{e_1} \vee x_{e_2}) \wedge (\overline{x}_v \vee x_{e_2} \vee x_{e_3}) \wedge (\overline{x}_v \vee x_{e_3} \vee x_{e_1}) \wedge (\overline{x}_v \vee \overline{x}_{e_1} \vee \overline{x}_{e_2} \vee \overline{x}_{e_3})$$

– egy terminálnak egy legyen a foka a megoldásban (legyenek az élei $e_1, e_2, \ldots, e_l$) ($|T| * (\Delta_t + 2 * \binom{\Delta_t}{2})$ literál)

$$(x_{e_1} \vee x_{e_2} \vee \ldots \vee x_{e_l}) \wedge (\overline{x}_{e_1} \vee \overline{x}_{e_2}) \wedge (\overline{x}_{e_1} \vee \overline{x}_{e_3}) \wedge \ldots \wedge (\overline{x}_{e_1} \vee \overline{x}_{e_l}) \wedge (\overline{x}_{e_2} \vee \overline{x}_{e_3}) \wedge \ldots \wedge (\overline{x}_{e_{l-1}} \vee \overline{x}_{e_l})$$

Megjegyzés: itt is érdemesebb lenne nagy fokszám esetén bináris tároláshoz folyamodni.

– ha egy él benne van a megoldásban (x_e igaz), akkor az él két csúcsának is igaznak kell lenni. ($4|E_G|$ literál)
Minden $e = \{v_1, v_2\}$ élre

$$(\overline{x}_e \vee x_{v_1}) \wedge (\overline{x}_e \vee x_{v_2})$$

– egy él két csúcsához ugyanazt a netszámot kell rendelni. ($3k|E_G|$ literál)

$$(\overline{x}_e \vee x_{v_1 1} \vee \overline{x_{v_2 1}}) \wedge (\overline{x}_e \vee x_{v_2 1} \vee \overline{x_{v_1 1}}) \wedge (\overline{x}_e \vee x_{v_1 2} \vee \overline{x_{v_2 2}}) \wedge (\overline{x}_e \vee x_{v_2 2} \vee \overline{x_{v_1 2}}) \wedge \ldots \wedge$$

$$\wedge (\overline{x}_e \vee x_{v_1 k} \vee \overline{x_{v_2 k}}) \wedge (\overline{x}_e \vee x_{v_2 k} \vee \overline{x_{v_1 k}})$$

– egy terminál netszámának a terminál sorszámával meg kell egyezni
Ehhez csak be kell állítani $\forall x \in T$-re az x_{vi} változókat a v csúcshoz tartozó $net(v)$ netszámra. Ez nem jelent új feltételeket: elég konstans egyesekkel és nullákkal helyettesíteni a megfelelő literálokat az előzőleg leírt klózokban.

Ezekkel a klózokkal tehát biztosítottuk azt, hogy szép megoldást kapjunk. A felhasznált literálok számának felső becslése látható minden klózcsoport elején. Δ_t a legnagyobb fokú terminál fokát, és Δ_{nt} a legnagyobb fokú nem terminál csúcs fokát jelöli. Az összesen felhasznált literálok száma $|N\backslash T|(3\binom{\Delta_{nt}}{2} + 4\binom{\Delta_{nt}}{3}) + |T| * (\Delta_t + 2 * \binom{\Delta_t}{2}) + 4|E_G| + 3k|E_G|$, és az összesen felhasznált változók száma $|E_G| + |V\backslash T| + \lceil logN \rceil |V|$. Ez egy kétrétegű Manhattan modellbeli feladatnál nem több, mint $(\frac{3}{2} + \log h)wh$ változó és $(18 + 3\log h)wh$ literál.

4.3. Az általános huzalozási feladat visszavezetése a SAT problémára

Az előző tétel segítségével visszavezettük a 2-net huzalozási feladatokat a SAT problémára. Bár a gyakorlatilag fontos eseteket tárgyaltuk, megnézzük, hogy hogy lehet egy olyan huzalozást visszavezetni a SAT problémára, amelynél nincsen a terminálok száma egy netben korlátozva. Ezek azért nem lesznek olyan fontosak a gyakorlatban, mint a 2-net huzalozási problémák, mert többször annyi változót használnak, mint azok.

Az ötlet a következő: bontsunk fel egy netet több pontpárra. Tekintsük az összes pontpárt, ahol az egyik pont egy előre kiválasztott net-beli pont, a másik egy ettől eltérő, ugyanabba a netbe tartozó pont. Így $|n| - 1$ pontpárt kaptunk, ahol n az aktuális net. Ha biztosítjuk minden pontpárra, hogy a két pontja össze legyen kötve, de továbbra se legyenek különböző netbe tartozó pontok összekötve, akkor készen vagyunk, visszavezettük a feladatot a 2-net huzalozási feladatra.

Ezt viszont megtehetjük az alábbi módon: készítsünk élváltozókat és a pontváltozókat minden pontpárra, viszont a netváltozókat nem többszörözzük. Ha ezek után a megfelelő klózokat fölírjuk (amelyek ugyanazok, mint amik eddig voltak, csak több van belőlük, mivel több az él- és pontváltozó), olyan megoldásokat kapunk, ahol bármely n netbeli terminál össze van kötve az n net-beli reprezentatív ponttal, s így bármely két n-beli terminál össze van kötve. Viszont mivel egy él csak úgy lehet behúzva, ha az él két végpontja ugyanabba a netbe tartozik (ugyanaz a netszáma), ezért nem lehet, hogy össze van kötve két különböző netbeli pont. Tehát a végleges kimenetben akkor lesz két pont között él, ha van olyan élváltozó, ami ahhoz az élhez tartozik (most már több élváltozó is lehet egy élhez). Viszont nem érdemes az összes net összes pontpárjának külön változókat rendelni: vegyük először az összes netből az első pontpárt. Ezek egy 2-netet alkotnak, úgyhogy ezekre felírhatjuk a klózokat. Aztán megint kiveszünk egy pontpárt minden netből, és egy új 2-netet kapunk. Ezt addig csináljuk, amíg el nem fogynak a netek. Ekkor egyesítjük a netváltozókat (vagy a generálás során figyelünk, hogy

mindig ugyanazt a pontváltozót használjuk), megoldjuk a kapott SAT problémát, és miután összevagyoljuk a megfelelő élhez tartozó élváltozókat, megkapjuk a bővített megoldást (amiből persze el tudjuk hagyni a nem szükséges komponenseket).

A probléma ezzel a megoldással csak az, hogy a sok élváltozótöbbszörözés majdnem $q-1$-szeresére növelheti a változók/literálok számát, ahol q a legtöbb terminálból álló net elemszáma. Ha csak legfeljebb 3 elemű netek vannak, akkor ez csak méret kétszerezést jelent, de $q > 5$ esetén már nagyon lelassíthatja a SAT megoldó működését.

5. A Rivest-Fiduccia algoritmus

5.1. A Rivest–Fiduccia heurisztika

Célunk egy kétrétegű csatornahuzalozási feladat megoldása. Általános esetben (*unconstrained modell*) ez hatékony módon lehetséges:

5.1. tétel (Recski–Strzyzewski). *Minden csatornahuzalozási feladat lineáris időben megoldható a kétrétegű korlátozás nélkűli modellben.*

Az ipari alkalmazások szempontjából fontos Manhattan modellben azonban a feladat igen nehéz:

5.2. tétel. *NP-teljes annak eldöntése, hogy egy csatornahuzalozási feladat adott szélességben megoldható-e két rétegen a Manhattan-modellben.*

Mivel hatékony algoritmus készítésére nemigen van esélyünk, kézenfekvőnek látszik különböző heurisztikákkal próbálkoznunk. A következőkben egy mohó csatornahuzalozási módszert mutatunk be, a Rivest-Fiduccia heurisztikát[30,]. Az algoritmus inputként kapja a huzalozandó csatorna netlistáját és annak kezdeti szélességét – a csatorna szélességével kapcsolatban ugyanis akár a minimális csatornaszélességet, akár a huzalozás adott szélesség mellett való megvalósíthatóságát kérdezzük, azt előre kiszámolni nem áll módunkban. Alsó korlát persze mindenképpen adható:

5.3. tétel. *A csatorna terhelése* (d)*[2] alsó korlátja a szükséges csatornaszélességnek.*

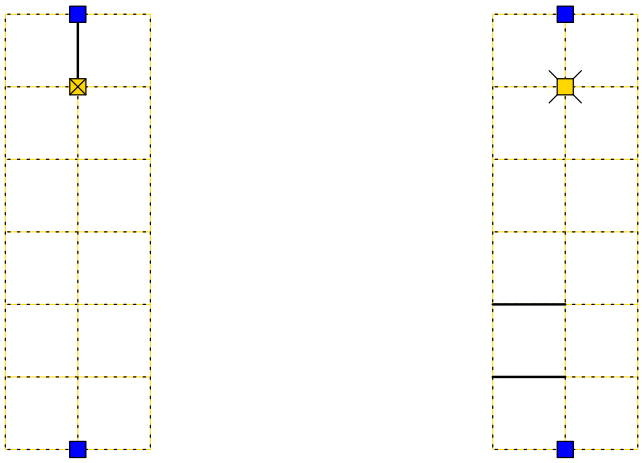
Az eljárás átlagos esetben elég gyors (a problémák nehézségéhez képest) és igen robosztus; oszlopról oszlopra haladva huzalozza be determinisztikusan a csatornát (*rule based column sweep approach*), mindig talál egy, az összekötöttségi igényeket kielégítő huzalozást, ami azonban lehet, hogy a peremfeltételeknek nem felel meg: vagy túl széles, vagy pedig „kilóg" a csatorna keleti oldalán (ha a nyugati oldaltól indítotttuk az eljárást). A másik nagy, bemutatott heurisztika-családtól, a genetikus algoritmusoktól az oszlopról oszlopra pásztázó huzalozóalgoritmusok feladatspecifikus es determinisztikus voltukban különböznek – éppen ezért, valamint elterjedtségük miatt esett rájuk a választásunk (ami azt illeti, a genetikus algoritmust, mint általános optimalizációs eljárást szokás mérnöki panaceaként is aposztrofálni).

Az algoritmus lépései

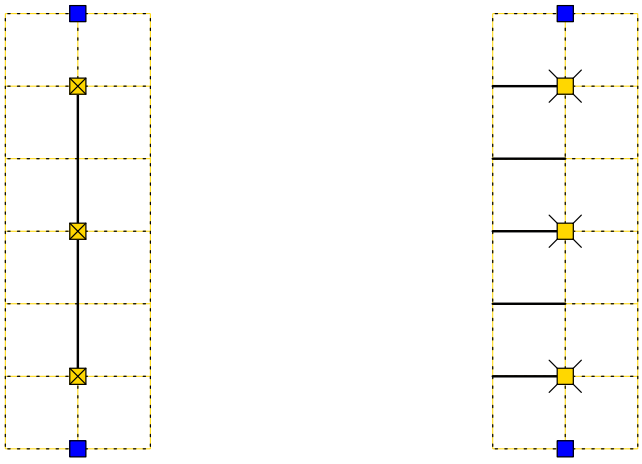
5.4. definíció. *a továbbiakban a csatorna* trackjeinek *hívjuk a csatornahuzalozási feladat négyszögrácsának egy, a csatorna partjaival párhuzamos sávjait.*

Az algoritmus a következő lépéseket hajtja végre oszlopról oszlopra haladva:

1. Az első lépésben az oszlop északi és déli partján jelenlévő terminálokat próbáljuk meg behuzalozni (a heurisztika koncepciójának megfelelően ezt az oszlop elhagyása után már nem tehetnénk meg), egyszerűen egy a legközelebbi szabad, vagy már a terminál netjéhez rendelt



5. ábra. A horizontális vezetékek netkiosztása alulról felfelé: 1, 2



6. ábra. (1,2,1,2,1)

trackhez való vertikális kötéssel. Amennyiben ilyen track nincsen, a terminál huzalozását az ötödik lépésben végezzük el. (5. ábra)

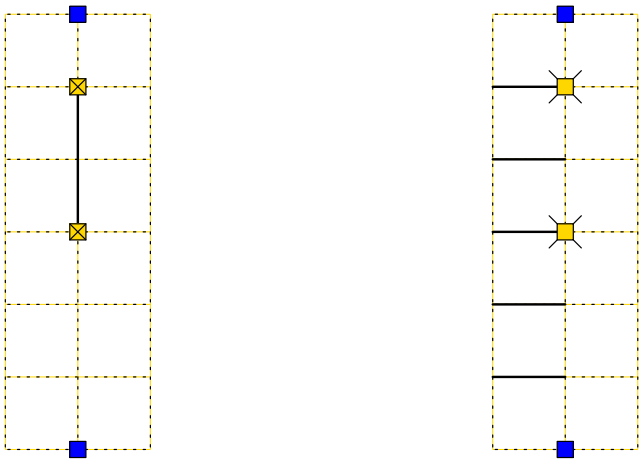
2. Az algoritmus működése során (az első lépésnek köszönhetően) létrejöhetnek olyan netek, melyek több tracken is helyet foglalnak. Megfelelő vertikális huzalozással az ilyen *osztott netek*[2] száma majdnem minden esetben csökkenthető. Egy-egy ilyen vertikális huzalozással persze blokkolhatjuk más, szintén osztott netek egyszerűsítését az adott oszlopban. Az osztott netek kiválasztásának sorrendje: ha még lehetséges a további egyszerűsítés, akkor mindig a legtöbb tracket felszabadító osztott netek közül a legnagyobb szélességgel rendelkezőt választjuk[3] – a további oszlopok számara így felszabadulnak trackek; az osztott net trackjei közül azt visszük tovább a hatodik lépésben, amelyik a negyedik lépés eljárása szerint a legkedvezőbb. (6. ábra)

3. Ebben a lépésben a nagyon „széles” osztott, de a blokkolások miatt tovább nem egyszerűsíthető neteket probáljuk meg összehúzni, mégpedig úgy, hogy a net vezetékeit minél közelebb visszük a net aktuális trackjeinek centrumához. (7. ábra)

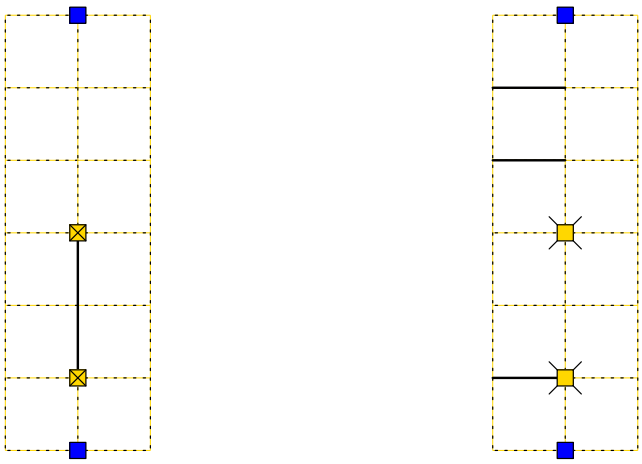
4. Az egyszerű neteket közelebb visszük a következő terminájukhoz – az emelkedő (a net követ-

[2] az angol nyelvű szakirodalomban: *split net*

[3] ennek – jobb híján – empirikus okai vannak: a megfigyelések alapján minél szélesebb egy egyszerűsítendő net, annál többet blokkol a többi egyszerűsíthető net közül, de ezt ellensúlyozhatja sok, távoli track felszabadítása.



7. ábra. (1,2,1,3,1)



8. ábra. (1,2,3)

kező terminálja az északi oldalon helyezkedik el) neteket felfelé, az eső (a következő terminál a déli oldalon) neteket pedig lefelé mozgatjuk. (8. ábra)

5. Vertikális huzalozással az oszlop tovább nem egyszerűsíthető, azaz tovább is léphetünk, amennyiben az első lépésben sikerült a terminál(oka)t a huzalozásba behoznunk. Ha nem ez a helyzet, akkor a problémás oldalon – oldalakon egy trackkel kibővítjük a csatornát, ezzel lehetővé téve az eddig kimaradt terminálok megnyugtató kezelését (a csatorna kibővítése persze visszahat a többi oszlopra is, ám ennek feloldása már triviális).
6. Minden osztott netet valamint azon nem osztottakat, melyeknek huzalozása még nem teljes (van még be nem kötött terminálju k) kiterjesztünk a következő oszlopra.

Költség. az algoritmus lépésszáma átlagos esetben lineárisan arányos az oszlopok számával.

A Rivest-Fiduccia heurisztika, mint azt fentebb bemutattuk, az adott csatorna huzalozását kétrétegű Manhattan-modellben hajtja végre – vagy legalábbis próbálja meg végrehajtani. A következö algoritmus a Rivest-Fiduccia heurisztika – mely a tapasztalat szerint robosztus a huzalozási modellt érintő változtatásokra nézve – kiterjesztése három rétegre.

5.2. A mohó háromrétegű csatornahuzalozó algoritmus

A félvezetőtechnológia fejlődésével a kezdeti két jelvezető réteghez képest lehetővé vált több réteg használata is – felvetve a kérdést, hogy az „egyszerű kétrétegű algoritmusok kis átalakítással megfelelnek-e három-négy réteg esetén is. Az itt tárgyalt, átalakított Rivest-Fiduccia heurisztika két, egymás fölé helyezett horizontális huzalozási réteg meglétét tételezi fel az eredeti egy helyett (HVH modell). A harmadik réteg horizontális irányú használatának előnye a vertikálissal szemben, hogy ezzel csökken a csatorna szélességére vonatkozó alsó korlát ($\lceil \frac{d}{2} \rceil$ lesz).

Elöljáróban egy terminológiai konvenció:

5.5. definíció. *Egy tracket hozzáférhetőnek nevezünk valamely oszlopban, ha abban az oszlopban az adott track két horizontális rétege közül pontosan egy foglalt.*

Háromrétegű huzalozás megvalósítására [29] a következő változtatásokat javasolják a Rivest-Fiduccia algoritmuson:

Az eredeti algoritmuson eszközölt változtatások

1.' Amennyiben az eredeti eljárás első lépése nem talál üres, vagy már az adott nethez rendelt trackkel rendelkező oszlopot, akkor a legközelebbi elérhető trackhez köti a terminált.

3.' Az osztott és az eredeti algoritmus alapján nem egyszerűsíthető neteknél a netek egyszerűsítésénél megengedett az elérhető oszlopok használata.

4.' Az egyszerű netek következő terminál felé mozgatásánál az elérhető és üres trackeket egyenrangúnak kell tekinteni.

Költség. Az algoritmus futási ideje átlagos esetben egyenesen arányos az oszlopok számával (hasonlóan az eredeti heurisztikához)[4]

5.3. A „mohó" switchboxhuzalozás

A Rivest-Fiduccia heurisztika átalakítható kétrétegű Manhattan-modellbeli switchboxhuzalozási problémák megoldására is. Az új heurisztika megtartja az eredeti legfőbb erősségét – és robosztusságának okozóját –, az oszlopról oszlopra való haladást (a már említett *column sweep approach* jelleget), ám azt két fázisban alkalmazza: először balról jobbra, majd jobbról balra (később kitűnik, hogy a kezdőoldal megválasztása tulajdonképpen teljesen véletlenszerű – ami egyébként várható is). Érdekesség, hogy az itt leírt algoritmus ipari alkalmazásként is „bizonyított", mint a General Electric Company Two-Pi huzalozórendszerének magja[31].

[4]Ez a tulajdonság az oszlopvezérelt gondolkodásmód megtartásából következik. Az egész algoritmuscsalád feladatmegoldó erejét innen meríti: a módszer az oszlopok huzalozása közötti topológiai függőségeket lokálisan oldja fel – igaz, felesleges kitérők és furatok *(via holes)* felhasználása árán. Gyorsasága, egyszerű implementálhatósága es átalakíthatósága azonban egyértelműen mellette szólnak.

Az algoritmus működési elve

Az alapgondolat: egy kétrétegű, Manhattan-modellbeli switchbox huzalozási feladat természetes módon felfogható csatornahuzalozási problémaként is, ahol a két, önkényesen csatornapartnak kikiáltott oldalon értelmezett csatornahuzalozás megoldása mellett még arra is ügyelni kell, hogy a megmaradó két oldalon található terminálok is megfelelően csatlakozzanak a hálózatba. Nyitott kérdés, hogy nem lehet-e „ügyesen" megválasztani a csatorna partjait; gyakorlati jelentősége a kérdésfeltevésnek azonban nincs: a „komoly", azaz ipari huzalozási problémáknál – melyeknél az óriási terminálszám mellett nem szabad elfelejteni, hogy a teljes rendszerben jól elkülönülnek az összetartozó elemi feladatokat végző, összekötendő elemi egységek – a gyakorlatban az „oszd meg és uralkodj" elvet követve felbontják a problémát téglalap alakú huzalozási területekre, majd az egyes netekre megengedett – elvárt, a téglalapok határán definiált kvázi-terminálok (lényegében a határvonalak és a netek metszéspontjai) rögzítése után az egyes területeket egy megfelelően megválasztott algoritmussal behuzalozzák. Az általunk vizsgált heurisztika a tapasztalat szerint a switchbox-feladat terminálkiosztásán túl más igényeket (vezetékek összhossza, dogleg-mentes huzalozás, etc.) nem támasztó részterületek huzalozására alkalmas leginkább. Mindezen megfontolások után a

kétszeres ("oda-vissza") pásztázás megvalósítása:

- Az első, balról jobbra történő próbálkozásnál az első oszlop nem "üres", hanem már tartalmazza a nyugati oldal termináljainak bekötésére szolgáló vezetékeket. Ezen újonnan bevezetett inicializálólépés után mindaddig alkalmazzuk a Rivest-Fiduccia heurisztikát, amíg minden net pontosan egy horizontális vezetékkel képviselteti magát az oszlopban[5], vagy pedig ez a feltétel ugyan nem teljesül, ám elértünk a keleti oldalhoz. Az első esetben az algoritmust folytatjuk a jobbról balra való huzalozással, a másodikban a heurisztikus probálkozás sikertelen volt.
- A jobbról balra való column sweep közben két célunk van: egyrészt itt is el kell tüntetnünk az osztott neteket, másrészt pedig az egyszerű neteket az első nekirugaszkodásban megállapított céltrackekre kell hoznunk. A probálkozás sikeres, ha legkésőbb az első lépés befejezőoszlopát elérve minden net egyszerű és minden net a céltrackjén található. Amennyiben ez a feltetel már korábban teljesül (és minden terminált behuzaloztunk), a feladat megoldása a részleges huzalozásból triviálisan megkapható.

A céltrackek kiosztása

A tapasztalat azt mutatja, hogy az egyes netekhez tartozó céltrackekre érdemes már a első pásztázás előtt javaslatot tenni; a kondenzációs fázisban az algoritmus megprobálja az egyszerűsített netet minél közelebb vinni ehhez a trackhez. A trackek meghatározása eléggé kézenfekvő – azt választjuk célként, ami előreláthatóan a legkönnyebben elérhető lesz a második fázisban, azaz

1. ha a netnek mind a keleti, mind a nyugati oldalon vannak termináljai, akkor a net céltrackje a net keleti oldalon található termináljainak centruma, különben

[5] az angol nyelvű szakirodalomban *condensed net*; az így elért trackekre a továbbiakban az adott net céltrackjeként hivatkozunk

2. ha a netnek az északi és a déli oldalon is vannak termináljai, akkor a céltrack a középső track (páros számú tracknél nyilván a trackszám felső vagy alsó egészrésze), különben
3. ha a netnek van az északi vagy a déli oldalon trackje, akkor a legfelső/legalső, vagy az arra következő track,
4. különben a céltrack nem definiált, hiszen a fogalom bevezetése ebben az esetben felesleges.

Megjegyezzük még, hogy hasonlóan az eredeti Rivest-Fiduccia heurisztikához, az algoritmus jó hatékonysággal kiterjeszthető kettőnél több rétegű esetekre.

A céltrackek koncepciója miatt az eredeti algoritmus lépéseit némiképp módosítanunk kell:

2'. Az osztott netek egyszerűsítésénél a net céltrackjéhez legközelebb eső tracket tarjuk meg, a többit felszabadítjuk

4.' minden egyszerű net esetén ahol lehetséges, a netet egy vertikális irányú huzalozással a lehető legközelebb „visszük" a céltrackjéhez.

6. A huzalozási probléma megközelítése genetikus algoritmussal

6.1. A genetikus algoritmus

Biológiai háttér. A természetes kiválasztódás, az evolúció máig legfőbb hajtóerejének tartott jelenség felismerése Charles Darwin (valamint Alfred Russel Wallace, aki Darwin javára visszahívta egyébként valamivel korábban kiadott publikációját) nevéhez fűződik.

A természetben mindig több egyed születik, mint amennyinek egyáltalán esélye lenne a túlélésre az elérhető erőforrásokat tekintve. Ez azt jelenti, hogy az azonos fajból származó egyedek (és bizonyos mértékig a különböző fajokból származó egyedek is) állandó versenyben vannak egymással a létfenntartáshoz és a reprodukcióhoz szükséges erőforrásokért, mint például az élelmiszer (zsákmány), az ivóvíz, a lakhely, vagy akár az ellentétes nemű egyedek. A természetes kiválasztódás szerint amelyik egyed jobban alkalmazkodott a környezetéhez, az általában több utódot fog létrehozni, mint más egyedek.

Világos, hogy a természetes kiválasztódásnak öröklődéssel kell párosulnia, azaz az utódoknak bírniuk kell szüleik jó (és rossz) tulajdonságaik egy részével, különben az evolúciós folyamat minden generációnál előlről kezdődne.

Az is hasonlóan egyszerűen látható, hogy fejlődés csak akkor történik, ha a populáció (azaz az egyedek összeségének) környezethez való adaptáltságának foka, más szóval *fitness*-e nem homogén, tehát az egyedek más és más mértékben adaptálódtak környezetükhöz. Ezt az egyedek *varianciájának* is nevezik. Mintapéldái a variancia fontosságának egyes baktériumtörzsek, amelyek sok évmillió óta is azonos génanyaggal bírnak, lényegében semmiben sem különböznek a mai példányok az őskoriaktól.

Ma már közismert számít, hogy az öröklődés a gének segítségével történik, amely meg is mutatkozik a genetikus algoritmusok terminológiájában.

A mai mérnöki alkalmazásokban használatos genetikus algoritmusok tipikusan az ivaros szaporodással reprodukáló fajokat modellezik. Ilyen esetekben a szaporodás szinte kivétel nélkül két egyed részvételével zajlik le. Mindkét szülő génanyaga megtalálható az utódban, azonban a két génanyag között bizonyos részek kölcsönösen kicserélődtek. Ennek a jelenségnek *crossover* a neve. A szaporodás közben, részben a gének másolása közben fellépő hibák, részben külső hatások (sugárzás, mutagének, stb.) miatt ún. *mutáció* léphet fel, azaz a génanyag egy kis része roncsolódik, véletlenszerűen módosul.

A genetikus algoritmus. A *genetikus algoritmusok* egy széles körben alkalmazott heurisztikacsaládot alkotnak. A módszer mérnöki alkalmazásokban a 70-es évektől fordul elő [8], bár szerzők ennél korábban is foglalkoztak az evolúció és a numerikus optimalizálás kapcsolatával. A módszer kezdeti, mára már klasszikusnak nevezhető változata, az *egyszerű genetikus algoritmus (simple genetic algorithm – SGA)* lényegében az előző szakaszban vázolt gondolatmenetet tükrözi. Mielőtt az eljárást ismertetjük, a megértés segítése végett álljon itt egy rövid leírás a használt szakkifejezésekről, valamint ezek angol megfelelőiről:

fitness (érték) – *fitness (value)*: Egy egyed „jóságát”, az optimumhoz közeliségét kifejező

érték. Fontos, hogy egy egyed fitness értéke nem minden esetben határozható meg pusztán az egyed tulajdonságainak ismeretében. Függhet az értéke például a populáció méretétől, egyéb egyedek tulajdonságaitól, stb. is.

fitness függvény – *fitness function*: A fitness függvény értelmezési tartománya a lehetséges egyedek, a megoldások halmaza, értékkészlete pedig egy olyan típus, amin definiált egy rendezés, általában valós számok. Utóbbi kitétel azért lényeges, mert két különböző fitness-értékű egyed közül el kell tudnunk dönteni, melyik reprezentálja a jobb megoldást. A rendezés alatt az esetek döntő többségében teljes rendezést értünk, azonban ez nincs mindig így: lehetőségünk van például egy többváltozós fitness függvény esetén *Pareto-optimalizálásra*.[6]

gén – *gene*: Egy megoldás reprezentációjńak alkotóelemei. Általában önmagukban is van valmilyen jelentésük, illetve gének sorozata egy (rész)megoldást alkot.

kromoszóma – *chromosome*: A gének összesége, tehát egy egyed teljes génanyaga a kromoszóma. Önmagában teljesen kódolja a megoldást, amit az egyed reprezentál.

egyed – *entity*: Az egyed a probléma egy (nem feltétlenül helyes) megoldását reprezentálja, tehát tartalmazz a teljes génanyagát, a fitness-értékét, illetve bizonyos genetikus algoritmus implementációknál még egyéb segédértékeket[7] is.

megoldás – *solution*: Az optimalizálni kívánt probléma egy (nem feltétlenül helyes) megoldása. Lényeges különbség a megoldás és az egyed között: míg az egyedben a kromoszóma a genetikus operátorok számára kedvezően kódolt formában, az algoritmus által használt segédadatokkal kiegészítve, addig általában a megoldás más, esetleg tömörített formában kerül ábrázolásra. A genetikus algoritmus a megoldással általában csak az optimalizáló szoftver határfelületein, a probléma, esetleg kezdeti megoldás megadásakor, illetve az optimalizált megoldás visszaadásakor kerül kapcsolatba.

szülő – *parent*: Egy egyed, amely génanyaga közvetlenül részt vett egy másik egyed létrehozásában, azaz a szülő kivásztásra került a kiválasztási operátor által és létrejött utódja.

leszármazott – *offspring*: Egy egyed, amelynek egy másik egyed a szülője[8] .

populáció – *population*: Az egyedek összesége. A *statikus generációs modellt* használó genetikus algoritmusoknál a populáció minden egyede azonos generációból való, más modelleknél ez általában nincs így.

kezdeti populáció – *starting population*: Az optimalizáció kezdetekor, általában véletlen módon meghatározott populáció: lényeges, hogy amennyire lehetséges, a keresési tér különböző pontjait reprezentáló megoldások legyenek tagjai. A gyakorlatban ezt véletlen választással költséghatékonyan lehet közelíteni.

generáció – *generation*: Egy egyed, illetve statikus generációs modellt használó genetikus algoritmus esetén egy populáció generációs száma, azaz generációja a kezdeti populációtól való távolságát méri, generációkban kifejezve, azaz, hogy hány szülőn keresztül vezet el tőle a kezdeti populáció (egy egyedéhez) az út.

[6] Ekkor egy egyed csak akkor számít jobbnak egy másiknál, amennyiben a fitness-értékének minden komponense nagyobb (kisebb) a másikénál (tehát a fitness-vektora dominálja a másik egyed fitness-vektorát). Ez a fajta optimalizálás alkalmas például egymásnak ellentmondó paraméterek egyidejű optimalizálására.

[7] Pl. ha az algoritmus nem statikus generációs modellel *(generation based GA)* dolgozik, az egyes egyedek élettartamát, stb.

[8] Nem minden egyed ilyen: tipikusan a kezdőpopuláció egyedeinek nincsenek szüleik.

crossover operátor- *crossover operator*: A crossover operátor a genetikus algoritmusok egyik legfontosabbnak tartott része. A crossover operátor az, ami igazán megkülönbözteti a genetikus algoritmust a véletlenszerű kereséstől. Az operátort a kiválasztás után, a két szülő kromoszómájának másolatára alkalmazzák, minek hatására az a két kromoszóma bizonyos részeit kölcsönösen kicseréli egymással. A kicserélt részek mérete általában, bár nem minden esetben egyenlő, ugyanis a legtöbb ábrázolásnál a kromoszóma mérete fix.[9]

crossover valószínűség – *crossover probability*: A crossover operátor alkalmazásának valószínűsége egy leszármazott-párnál.

mutációs operátor- *mutation operator*: A crossover operátor mellett a másik általánosan elterjedt operátor az egyedek génanyagának módosítására. Általában a kromoszómának csak egy kis részét módosítja, véletlen módszerrel.

mutációs valószínűség – *mutation probability*: A mutációs operátor alkalmazásának valószínűsége egy leszármazottnál.

kiválasztás – *selection*: A populációból azon egyedek kiválasztása a kiválasztási operátor segítségével, amelyekből szülők lesznek.

kiválasztási operátor – *selection operator*: A genetikus algoritmus ezt az operátort használja a populációból egyedek kiválasztására, akikből szülők lesznek. Megválasztásánál figyelni kell arra, hogy az egyedek közül lehetőleg „igazságosan" válasszon, azaz a fitnessüknek megfelelően, azonban lehetőleg ne dominálhassa egy vagy néhány egyed a populációt.[10]

elitizmus – *elitism*: A kiválasztási operátor segédoperátora, hatására általában a populáció legjobb egyede módosítás nélkül átkerül a következő generációba. A módszer a keresés konvergenciáját gyorsítja, ami bizonyos esetekben kedvezőtlen is lehet.

megállási paraméterek – *stopping parameters*: Az optimalizálás megállási kritériuma(i). Segítségükkel megállapítható, hogy tovább kell-e futtatnunk a heurisztikát, vagy az algoritmus valószínűleg megtalálta az eredményt (ill. annak egy jó közelítését). Tipikus változatai: generációs szám, legmagasabb, átlagos fitness érték a populációban, ezen értékek két generáció közötti változása, illetve ezek kombinációja.

Az algoritmus leírása:

Adott egy kezdeti, általában véletlenszerű *egyedekből* felépített *populáció*, az ún. *kezdő* vagy *kezdeti populáció*. Az algoritmus ebből a kezdeti populációból jut el egy optimális, vagy optimumközeli megoldáshoz, tehát olyan egyedhez, ill. egyedekhez, amely(ek) génanyaga egy megfelelő megoldást kódol. Az algoritmus egy lépésben az i-edik generációból az $i+1$-ediket állítja elő, a következőképpen: Lefoglal egy, a populációt tartalmazó adatszerkezet méretével megegyező méretű ideiglenes tárhelyet, az $i+1$-edik generáció egyedeinek. Ezután a kiválasztási operátorral az i-edik generáció egyedei közül kiválaszt kettőt, amely egyedek lesznek a szülők. A szülők kromoszómáit duplikálja, majd ezután meghívja rájuk a crossover és a mutációs operátorokat. Az így létrejött két új egyed az $i+1$-edik generációt tartalmazó ideiglenes tárhelyre

[9] Talán a legszemléletesebb példa erre, ha egy gráf *Hamilton körét* szeretnénk kódolni, mégpedig a meglátogatott csomópontok számából a meglátogatásuk sorrendje szerint képzett sorozattal.

[10] Ezt egyébként a fitness függvény és a mutációs valószínűség helyes megválasztásával is elő lehet segíteni.

kerül. Az algoritmus ezeket a lépéssorozatokat addig ismétli, amíg teljesen fel nem tölti a $i+1$-edik generáció populációját az új egyedekkel. Amennyiben a populáció mérete nem osztható az egy kiválasztáskor keletkező új egyedek számával, az algoritmus az új generáció egyedeiből valamilyen módszerrel eldob annyit, hogy az új populáció mérete is megegyezzen a régivel. Végül az algoritmus felülírja az i-edik generáció egyedeit az $i+1$-edik generáció egyedeivel.

Az algoritmus addig lép generációról generációra, amíg a megállási kritérium(ok) nem teljesül(nek). Megállás után az algoritmus eredményként az utolsó generáció legjobb egyedét adja vissza.

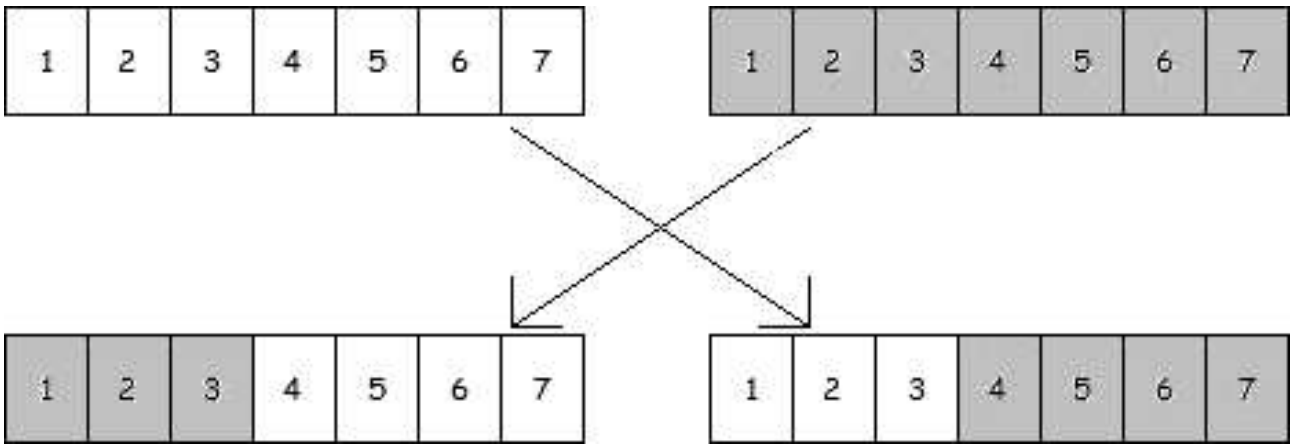
Természetesen az egyes operátoroknak illetve az algoritmusnak magának is nagyon sokféle változata van, kezdve a paraméterek variálásától esetleges új operációk bevezetésén keresztül a generációs modell megváltoztatásáig. Itt nem célunk ezek összefoglalása, ezért a teljesség igénye nélkül, pusztán tájékoztató jelleggel írunk a genetikus operátorok leggyakrabb implementálási formáiról.

ábrázolási módok. Kétféle ábrázolási módot tárgyalunk, a jelsorozat, valamint a fa ábrázolási módot. A két ábrázolási mód sokszor a genetikus operációk, a crossover és a mutáció jelentősen különböző implementálását követeli meg, illetve az ábrázolási módból következhetnek bizonyos konzisztenciafeltételek, amelyek betartását ellenőrizni kell. A jelsorozat ábrázolási mód tipikusan jól alkalmazható Hamilton kör keresésére, rádiós vevőállomások elrendezésének optimalizálására, míg a jelen dolgozatban implementált genetikus algoritmus lényegében a fás ábrázolási módot alkalmazza.

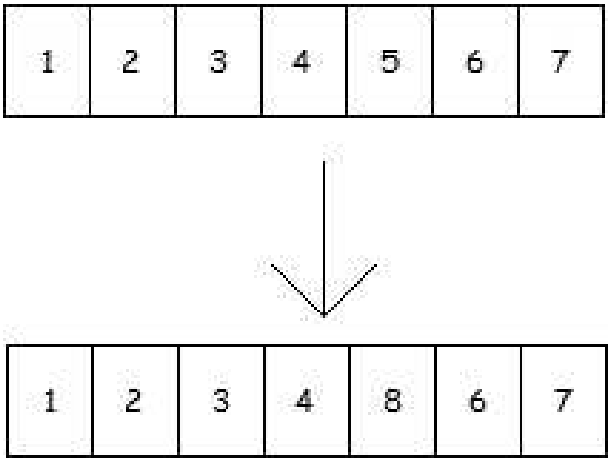
crossover operátor. A crossover operátor feladata, hogy két egyed kromoszómáinak részeit kölcsönösen kicserélje egymással. Ezzel az öröklődést modellezi, miszerint mindkét gyermek mindkét szülőtől örököl tulajdonságokat. A jelsorozattal ábrázolt kromoszómáknál jellemzően kétféle crossover operátor létezik: *egypontos (single point)* ill. *kétpontos (double point) crossover.* Az egypontos crossover operátor kiválaszt egy pozíciót a kromoszómában és a két kromoszómának a pozíción túl eső részét megcseréli. A kétpontos crossover operátor két pozíciót választ ki a kromoszómában és az *először és másodszor kiválasztott pozíciók közti géneket cseréli meg.* Ez akkor fontos, ha az elsőre kiválasztott pozíció nagyobb, mint a másodjára kiválasztott: ebben az esetben az első pozíciótól a kromoszóma végéig, majd a kromoszóma elejétől a második pozícióig kerülnek kicserélésre a gének. A fás reprezentációnál a crossover tipikusan részfák cseréje. Ennél a fajta cserénél figyelembe kell venni, hogy a részfák cseréje folytán az egyes kromoszómák csomópontjainak száma megváltozhat, illetve az ezzel járó esetleges konzisztenciaproblémákat kezelni kell.[11] E két crossover-családon kívül az operáció implementálásának még számos más lehetséges változata is van, köztük olyanok is, amelyeknél nem csak két szülő van. Ilyen módszer például a *random keverés*, amelynél az egyed minden gént egy véletlenszerűen kiválasztott szülőtöl kap.

mutációs operátor. A mutációs operátor feladata, hogy új, a populációban eddig jelen nem lévő génekkel bővítse az egyedek génanyagát. Ezt a mutációs operátorok tipikusan véletlen

[11] Például ún. *kromoszóma-javító eljárásokkal.*



9. ábra. single point crossover



10. ábra. single point mutation

számok segítségül hívásával teszik. A jelsorozatos ábrázolásnál tipikus *pontmutáció (single point mutation)* esetében az operátor kiválaszt egy pozíciót a kromoszómában, majd a pozíción lévő gént mutálja, egy véletlen értékkel helyettesíti. Az új értéknek érvényes kódnak kell lennie, azonban elképzelhető, hogy a kromoszómára érvényes konzisztenciafeltételeknek[12] nem tesz eleget, ilyenkor a hibát kezelni kell, például a mutáció visszavonásával, vagy a kromoszóma módosításával. A fás ábrázolásnál szokásos mutációs operátorok például a fán belüli részfák kicserélése, illetve a fa részfáinak törlése és véletlen generált részfákkal való helyettesítése. Az operációból adódó esetleges konzisztenciaproblémákat természetesen itt is kezelni kell.

kiválasztási operátor. A kiválasztási operátorok nagymértékben függetlenek a választott ábrázolástól és általában a fitness értékkel vannak szoros kölcsönhatásban. Itt csak két, elterjedt kiválasztási metódust említünk meg, a *rulettkerék kiválasztást – (roulette wheel selection)* és a *körmérkőzéses kiválasztást (tournament selection).*

A rulettkerék kiválasztás algoritmusa a következő: képezünk egy véletlen számot 0 és a generáció összes egyede fitness-ének összege között, amit egy segédváltozóban tárolunk, majd ebből a számból addig vonjuk ki az egyes egyedek fitness-értékeit, amíg a segédváltozó értéke

[12]Egy egyszerű példa erre egy Hamilton kör keresésére alkalmas genetikus algoritmus, ahol a kromoszóma a meglátogatott gráfpontok sorozatából áll. Ilyenkor a mutáció után előfordulhat, hogy egy gráfpont kétszer fordul elő a kromoszómában, ami nyilvánvalóan inkonzisztenciát jelent.

0 vagy negatív nem lesz.[13] Amelyik egyed fitness-ét utóljára vontuk ki az összesített értékből, az kerül kiválasztásra. A módszer hátránya, hogy a nagy fitness-értékű egyedek túl erősen dominálhatják a populációt. Ennek elkerülésére használják a *fitness kalibrálást (fitness scaling)*, azaz a fitness értékeknek a populációtól függő transzformálását.

A körmérkőzéses kiválasztásnál ez utóbbi probléma nem merül fel. A kiválasztás úgy zajlik, hogy a rulettkerék-kiválasztással kiválaszt egy előre meghatározott számú egyedet, majd az így kiválasztott szubpopulációból a legnagyobb fitness-ű egyed kerül kiválasztásra. A második szülőt hasonló módon választja ki az algoritmus.

Fontos megemlíteni még a kiválasztási operátorok egy opcionális segédoperátorát, az *elitizmus operátort*, amely minden kiválasztás elején a generáció legjobb egyedét másolja át a következő generációba. Ezzel megakadályozható, hogy egy korábban megtalált optimális vagy optimumközeli megoldás elvesszen.

megállási kritériumok. A megállási kritériumok általában az egész folyamat, vagy a legjobb egyed tulajdonságaiból próbálják meg eldönteni, hogy kielégítőek-e már az eredmények, abba lehet-e hagyni az optimalizációt. Szokásos megállási kritériumok a következők:

- egy bizonyos számú generáció elérése
- egy bizonyos fitness érték elérése a legjobb egyednél
- egy bizonyos átlagos fitness érték elérése az egész populációt tekintve
- a maximális vagy átlagos fitness érték generációnkénti változásának egy bizonyos határ alá esése

illetve ezek kombinációi.

Természetesen az egyes operációknak nagyon sokféle változata van, illetve léteznek olyan alkalmazások, amelyeknél új operációk is bevezetésre kerülnek. Jó áttekintést nyújt a témáról [10].

6.2. Az implementáció

Ebben a fejezetben a heurisztika szoftver megvalósítását tárgyaljuk. Szó esik az implementált algoritmusról, a szoftver szerkezetéről, valamint a szoftver és a keretrendszer kapcsolatáról.

Az implementált algoritmus. Az algoritmus egy *egyszerű genetikus algoritmus*, objektumos ábrázolással. Az algoritmus kritikus pontja, ami általában is megnehezíti a genetikus algoritmusok $VLSI$ problémákra való alkalmazását, hogy a crossover operátorok alkalmazása során könnyen hibás megoldások keletkezhetnek. Ezt a problémát többféleképpen kezeljük. Egyrészt az egyedek fitnessében tükröződik, hogy a megoldás helyes-e vagy sem. Azonban kizárólag helyes megoldások engedélyezése esetén a populáció esetleg nem tud elmozdulni, csak a keresési tér egy szűk részét járhatja be. Ezért minden egyed, amelynek a szülei helyesek voltak, ő azonban már nem az, ún. *próbaidőt* kap: k generációig nem tükröződik a fitness értékében, hogy helytelen. Amennyiben eddig utódai egy helyes megoldásba konvertálódnak, a próbaidő a

[13] A lebegőpontos aritmetika pontatlansága miatt célszerű 0 helyett valamilyen kicsi pozitív értékre tesztelni.

helyes utódokra természetesen nem lesz érvényes. Amennyiben nem, úgy a túl kései helytelen utódok fitness értéke meredeken csökken. A problémát máshogy kezeli az elitizmus operátor: az elitizmus ebben az esetben nem veszi figyelembe a próbaidőt, ezért amennyiben van legalább egy jó megoldás a populációban, azt megtartja, átmenti a következő populációba, így nehezítve meg, hogy az algoritmus olyan területére tévedjen a keresési térnek, ahol nem, vagy csak nagyon rossz megoldást tud találni.

Az egyedek ábrázolása a keretrendszer *Routing* osztályával történik. Az egyed ezen kívül még nyilvántartja az általa reprezentált megoldás fitness értékét is. A kromoszóma így gyakorlatilag egy objektum, illetve egy objektum sorozat, azonban mivel egy Routing osztály minden nethez egy (részleges) Steiner-fát ír le, a kromoszóma tekinthető fák uniójának is.[14]

Az algoritmus statikus generációs modellt használ. Ez azt jelenti, hogy mindig csak a legutolsó generáció egyedeit tárolja, ezekre alkalmazza a kiválasztás operátort a következő generáció képzésekor. Generációváltás után az előző generáció egyedei elvesznek.

Az algoritmus egy előre megadott méretű populáción dolgozik, aminek mérete az optimalizáció futása során konstans marad.

Az algoritmus a kezdőpopulációt mohó módon határozza meg: minden netet beköt, majd amennyiben a vízszintes sávok között talál szabad sávot, oda húzza be a vezetéket. A kezdőpopulációban ütközések, azaz egymást metsző vezetékpárok megengedettek.

Az algoritmus körmérkőzéses kiválasztást használ, a szubpopuláció mérete paraméterezhető.

A heurisztika megállási kritériumait a felhasználó paraméterezheti, jelenleg a következő lehetséges értékeket lehet figyelni:

- egy bizonyos számú generáció elérése
- egy bizonyos fitness érték elérése a legjobb egyednél
- egy bizonyos átlagos fitness érték elérése az egész populációt tekintve
- a maximális vagy átlagos fitness érték generációnkénti változásának egy bizonyos határ alá esése

az egyes kritériumok küszöbértéke szintén beállítható.

A crossover operátor a Steiner-fák egy-egy részfáját cseréli ki a két kromoszóma között. A csere leírásához szükség lesz a fák, azaz a netek vezetékei hosszának definiálására. Jelölje l a vezetékek által érintett rácspontok számát. Egy net vezetékeinek hossza alatt az $l-1$ mennyiséget értjük. Az operátor a két egyed között n részfát cserél ki, ahol n egy véletlen szám 1 és a netek száma között. Mivel a szoftver a huzalozást objektumok sorozataként tárolja, ezért az operátor implementációjakor erre tekintettel kellett lenni. Az operátor a következőképpen jár el (egy-egy részfa cseréje esetén) :

1. Képzi a kicserélendő Steiner-fák vezetékeinek hosszát. Ez legyen k_1 és k_2.
2. Generál egy véletlen számot 0 és k_i között, ez legyen r_1 és r_2.
3. A két szám különbsége, $k_i - r_i$ adja meg a kicserélendő részfák nagyságát. A csere után a kromoszómák mérete $r_1 + k_2 - r_2$, ill. $r_2 + k_1 - r_1$ lesz. Amennyiben valamelyik kromoszóma

[14] Például a crossover operátor ezt a szemléletet követi.

új mérete kisebb lesz, mint az eredeti méret 50%-a, vagy nagyobb, mint az eredeti méret 200%-a, a csere nem történik meg.

4. A fa egy tetszőlegesen kiválasztott terminájától r lépést megy a vezetéken, ahol egy lépés egy egységnyi hosszú, tetszőleges irányú vezetéken való átlépést jelent. Amennyiben kereszteződéshez ér a kezetéken, véletlenszerűen kiválaszt egy irányt, valamint a másik irányban lévő vezetékek hosszát is hozzáadja a már meglépett lépések számához.
5. Az így elért pontok (be nem járt irányban fekvő) részfáit cseréli ki az algoritmus.

Amennyiben a csere nem történt meg, az operátor új véletlen számokkal próbálkozik, összesen három alkalommal. Ha ezek után sem járt sikerrel, akkor az operáció az eredeti állapotot meghagyva befejeződik.

A mutációs operátor véletlenszerűen kiválaszt két pozíciót a kromoszómában, majd a köztük lévő vezetékdarabot törli és egy véletlenszerűen generált, de a Manhattan-modell kikötéseinek eleget tevő drótdarabbal kicseréli.

A program az egyed fitness értékét az alábbi szempontok figyelembevételével határozza meg: a fitness célja hogy a helyes és helytelen megoldásokat élesen elkülönítse egymástól. Két helytelen megoldás fitness értéke között az ütközések száma, illetve a be nem kötött netek százalékos aránya disztingvál. A probléma eldöntési jellegéből fakadóan amint találunk egy helyes megoldást, be lehet fejezni a heurisztika futtatását.

A későbbiekre tervezzük egy *evolúciós ugrás* operátor bevezetését is, amely egyrészt hibás egyedeket lenne képes megjavítani, másrészt nem teljes huzalozásokat, amennyiben egyszerű módon kiegészíthetők, lenne képes befejezni.

A fitness számítása, valamint az elitizmus jelenleg is fejlesztés alatt áll. Tervbe van véve továbbá a crossover operátor módosítása úgy, hogy paraméterezhető legyen, hány fát cserél ki a két egyed között és készül egy alternatív, a kromoszómát csupán egy ponton megváltoztató mutációs operátor is.

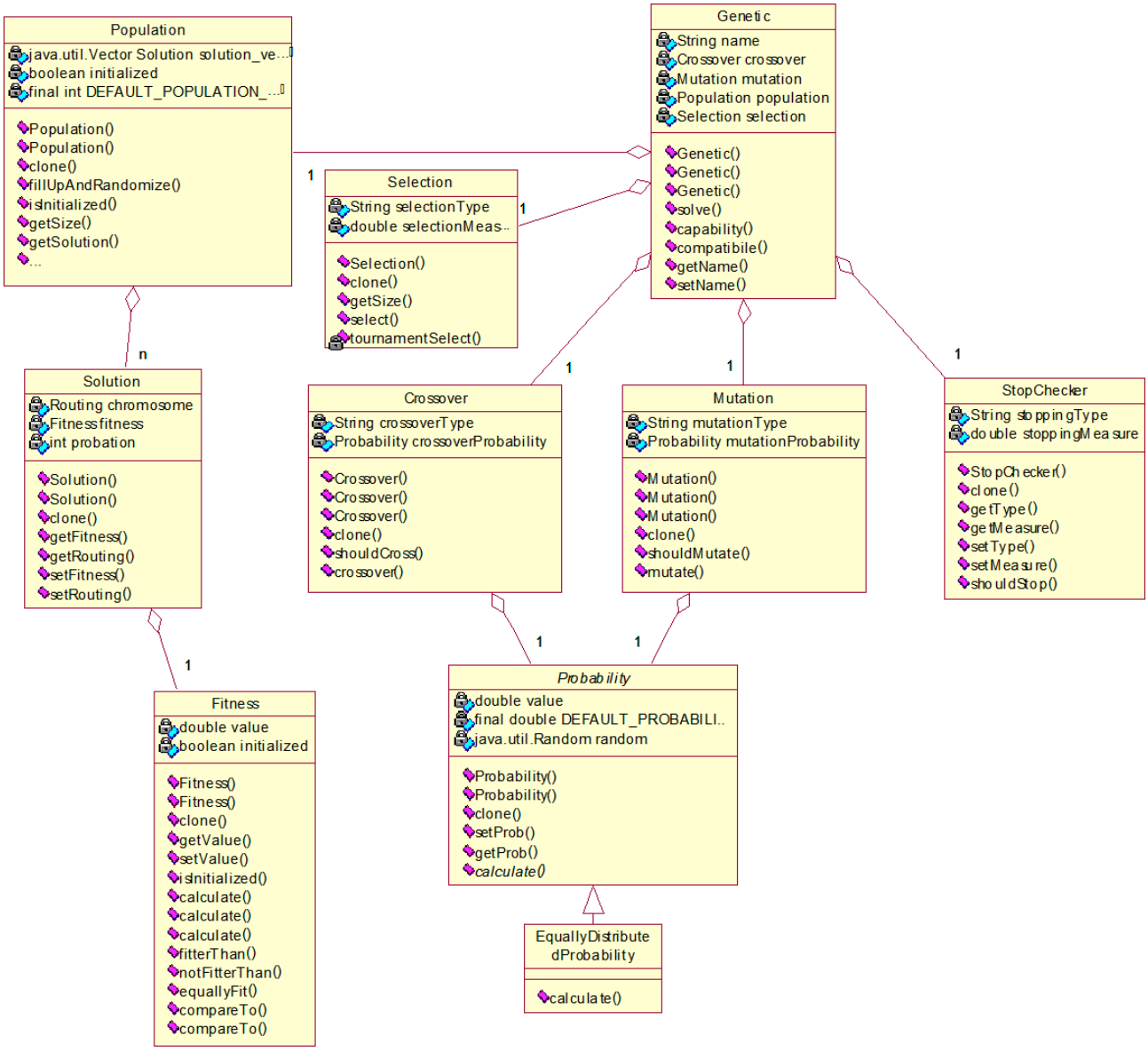
A szoftver szerkezete. A szoftver szerkezetének kialakításakor nem csak a szokásos struktúrális dekompozíciót vettük figyelembe, hanem –lévén az algoritmust végrehajtó objektumnak nagyon sok felelőssége volt– a funkcionálisat is. Így az algoritmus tulajdonképpeni végrehajtó osztálya, a Genetic felelősségkörét sikerült lényegében a generációs modell kezelésére korlátozni. Ez olyan szorosan összefügg a heurisztika tulajdonképpeni végrehajtásával, hogy nem láttuk indokoltnak külön objektumban elhelyezni. A genetikus operátorok, valamint a megállási kritériumok vizsgálata is külön objektumokban kaptak helyet.

A szoftver flexibilis lehetőségeket biztosít a heurisztika futtatására, ahol nem hagytuk figyelmen kívül a későbbi bővítés lehetőségeit sem. Erre szolgál a vezérlőosztály a Genetic többféle értékkel paraméterezhető konstruktora például, amellyel a heurisztika futása egy lemezre mentett köztes állapotból újra elindítható, akkor is, ha az egyes operátorok viselkedése időben változik. Az egyes osztályok is hasonlóan robosztus konstruktorokkal rendelkeznek, ami megkönnyíti többek között, hogy a későbbiekben más kezdőpopuláció-generáló függvényeket is implementáljunk. Az osztályok természetesen teljesítik az egységbe zárás követelményeit is.

A következőkben megadjuk az egyes osztályok rövid leírását.

- Genetic: A tulajdonképpeni vezérlőosztály, amely a keretrendszer *hu.bme.vlsi.framework.Algorithm* interfészt implementálja. Része a generációs modell, jelenleg statikus modellel dolgozik.
- Population: A populációt tároló osztály. Az egyedeket rajta keresztül lehet lekérdezőfüggvényekkel elérni, az osztály felelőssége egy egyedből illetve az üres egyedből egy véletlenszerű populációt készíteni.
- Solution: Egy egyedet tároló osztály, részei jelenleg a kromoszóma és a fitness.
- Fitness: A fitness értéket tároló, valamint annak kiszámolásáért felelős osztály.
- Selection: A kiválasztási operátorok tárolóosztálya. Jelenleg az osztály a körmérkőzéses kiválasztást implementálja.
- StopChecker: A megállási kritériumokat implementáló osztály. Jelenleg a következő kritériumok vannak implementálva:
  - egy bizonyos számú generáció elérése
  - egy bizonyos fitness érték elérése a legjobb egyednél
  - egy bizonyos átlagos fitness érték elérése az egész populációt tekintve
  - a maximális vagy átlagos fitness érték generációnkénti változásának egy bizonyos határ alá esése
- Crossover: A crossover operátort implementáló osztály.
- Mutation: A mutációs operátort implementáló osztály.
- Probability: A crossover és a mutációs operátorok végrehajtása során szükség van bizonyos eloszlás szerint bekövetkező események beteljesülésének eldöntésére. Erre szolgál a Probability absztrakt osztály és leszármazottai.
- EquallyDistributedProbability: A Probability absztrakt osztály leszármazottja, egyenletes valószínűségi eloszlást valósít meg.

A szoftver és a keretrendszer kapcsolata. A szoftverrendszer igazi próbája az illesztés volt a már létező keretrendszerhez. Egy genetikus algoritmus merőben más szerkezetű, mint a keretrendszerben eddig implementált algoritmusok, heurisztikák. Gondos mérlegelés után úgy határoztunk, hogy az algoritmus számára nem konvertáljuk át az adatokat, hanem a genetikus operátorokat úgy implementáljuk, hogy azok a már meglévő keretrendszerbeli ábrázoláson dolgozzanak, tehát az algoritmus és a keretrendszer szorosan együttműködnek. Ezzel elkerültük egy új keretrendszer kifejlesztésének fádadtságát és veszélyeit, valamint a meglévő rendszer összes előnyét élvezhettük.



11. ábra. a modell statikus struktúra-diagramja

7. Tesztelés

A tesztkörnyezet fejlesztésekor a BME Számítástudományi és Információelméleti Tanszékén kifejlesztett keretrendszerből ([2]) indultunk ki. Ennek a már meglévő, Java nyelven írt keretrendszernek a fő funkciói:

- Grafikus felhasználó felületet biztosít, melyben VLSI huzalozási problémákat lehet megfogalmazni számos különböző modellben.
  Ezt a funkciót csak kipróbáltuk, de tesztelési célokra nem használtuk. A tesztproblémákat a megfelelő adatstruktúrákat közvetlenül Java nyelven megfogalmazott saját programrésszel építettük fel.
- Három huzalozási feladat-megoldó algoritmust implementál: egy Gallai-féle ([3]), 3 rétegű, Manhattan modellben optimális challel-huzalozót; egy Szeszlér-féle ([24]) Manhattan modellbeli közelítő switchbox-huzalozót; és egy Recski-féle ([12]) 2 rétegű, Manhattan modell-beli közelítő channel-huzalozót.
  Mivel ezen algoritmusok közül csak a közelítő channel-huzalozó dolgozik az átalunk implementált módszerekkel azonos modellben, ezért csak ezt futtatuk, és eredményét összehasonlítottuk a mi algoritmusaink eredményeivel.
- Képes megjeleníteni a problémát és megoldását 2 és 3 dimenzióban.
  A keretrendszer ezen képességeit nem használtuk ki.
- Képes a problémát és a megoldásának 2 dimenziós ábrázolását FIG formátumban kimenteni, és ez a kimenet ábraként egy dolgozatba beilleszthető.
  A TDK dolgozatunkban szereplő síkbeli ábrák nagy része ezzel a keretrendszerrel készült.

A tesztelés során kétrétegű, Manhattan-modell-beli, 2-net channel huzalozást végző algoritmusokat vizsgáltunk. Az alábbi algoritmusokat kívántuk összehasonlítani:

- a keretrendszerben már meglevő channel-huzalozó
- a saját fejlesztésű genetikus algoritmus
- visszavezetés SAT kielégíthetőségi problémára, a SAT megoldása saját fejlesztésű, heurisztikus algoritmussal
- visszavezetés SAT kielégíthetőségi problémára, a SAT megoldása a *zChaff* programmal ([27])
- visszavezetés SAT kielégíthetőségi problémára, a SAT megoldása a *SICStus Prolog* CLP(B) könyvtárával
- visszavezetés CSP kielégíthetőségi problémára, a CSP megoldása a *SICStus Prolog* CLP(FD) könyvtárával ([28])

Az alábbi tulajdonságokat vizsgáltuk:

- helyes-e az implementáció?
- a mért futási idő ipari benchmark-ok esetén
- a mért futási idő véletlenszerűen generált feladat esetén
- a megtalált megoldás szélessége

– memóriaigény

7.1. Előzetes megfontolások

Channel huzalozás esetén a huzalozás számára rendelkezésre álló téglalap szélessége nem rögzített. Csak azt tudjuk, hogy a két átellenes oldalon hogyan helyezkednek el a terminálok. A feladat az, hogy a huzalozást minél kisebb szélességgel oldjuk meg, és ezáltal minél kisebb, egyszerűbb és olcsóbb legyen a VLSI áramkör gyártása. Az optimális channel-huzalozási feladat NP-nehéz, így a gyakorlatban közelítő algoritmusokat használnak, amelyek nem feltétlenül minimális szélességgel oldják meg a problémát.

Ilyen közelítő algoritmus a keretrendszerben már meglévő channel-huzalozó, és természetéből fakadóan közelítő megoldást ad a saját fejlesztésű genetikus algoritmus is. A kielégíthetőségi problémákra való visszavezetés esetén viszont a szélességet előzetesen ismerni kell. Ezt intervallumfelezéssel hidaltuk át, vagyis az algoritmust többször futtattuk, és az alapján, hogy talált, vagy nem talált érvényes huzalozást, megfeleztük a minimális szélesség még szóba jövő értékeinek tartományát. A kezdeti tartományt pedig vagy a channel-huzalozó által visszzaadott felső becslés alapján vettük fel, vagy 1-től indulva addig dupláztunk, amíg megoldás nem született.

Minden algoritmus lefutása után egy külön rutinnal ellenőrizzük a talált megoldás helyességét. Természetesen semmit sem tudunk mondani akkor, ha az algoritmus nem talál megoldást: elképzelhető, hogy tényleg nincs megoldás, de az is lehet, hogy a program implementációja hibás. Azt sincs módunk ellenőrizni, hogy a megoldás optimális-e, hiszen ehhez várhatóan a huzalozással összemérhető bonyultságú, új algoritmusra lenne szükség. A helyesség ellenőrzése abból áll, hogy megvizsgáljuk, hogy az előírás szerint azonos netbe tartozó terminálok tényleg össze vannak-e kötve, és a különböző netbe sorolt terminálok között tényleg nem fut vezeték.

A vizsgált algoritmusok nem használnak a bemenet méreténél lényegesen több memóriát. Mivel az iparban használatos channel huzalozási problémák mérete ($\approx$ 100 terminál) jóval kisebb a manapság kapható olcsó memóriamodulok kapacitásánál, ezért a számítás során a memóriahasználat nem kritikus. Úgy becsültük, hogy a tárigény extrém esetben sem fogja elérni a 10 MB-ot, ezért a számítást végző folyamatoknak 20 MB-ra korlátoztuk a felhasználható tár méretét. Ha ezt a tárat egy program túllépné, implementációs hibára kell gyanakodnunk.

Az egyes algoritmusok legfontosabb összehasonlítási alapja a futás során felhasznált idő. Ez az gépidő költségként megjelenik a VLSI alkatrész gyártási folyamatában. Fontos, hogy a részletes huzalozás számítására fordított idő a teljes tervezés hosszával összemérhető legyen. Épp ezért minden feladathoz egy időkorlátot határoztunk meg, és megszakítottuk azon algoritmusok futását, amelyek ennyi idő alatt nem adtak eredményt. A korlát értékét a program fejlesztési ciklusához igazodva 15 percben határoztuk meg. A huzalozási feladatok megoldásánál a feladat méret-idő függvényében rejlő konstans és lineáris szorzó általában nem számít: ezek újabb hardver elemek vásárlásával ellensúlyozhatók. Azt érdemes inkább vizsgálni, hogy milyen méretre kezd el robbanni a komplexitás.

7.2. A hardver- és szoftverkörnyezet

A teszteket egy személyi számítógépen futtattuk, melyet ez idő alatt másra nem használtunk. A számítógépben 1 db 1200 MHz-es AMD Athlon processzor és 768 MB memória volt. A processzor 256 kB cache memóriával rendelkezett. A gépen Linux operációs rendszer és 2.4.19-es verziószámú kernel futott. A swap-elést (memórialapok cseréje a háttértárra) időlegesen letiltottuk. A kernel alapszolgáltatásain kívül (`kapmd`, `keventd`, `kreiserfsd`, `kseriod`, `ksoftirqd_CPU0` és `kswapd`) más daemon-ok nem futottak. A teszteket egy `bash` parancsértelmezőből, kézzel indítottuk.

A Java virtuális gép a Sun 1.4.1-es JDK-jának *Java HotSpot(TM) Client VM*-je volt. A SAT megoldására a *zChaff* program Z2001.2.17-es verzióját és a SICStus Prolog 3.9.1-es változatának CLP(B) könyvtárát használtuk, a CSP probléma megoldását pedig ugyanennek a Prolog változatnak a CLP(FD) könyvtára végezte.

7.3. Az idő mérése

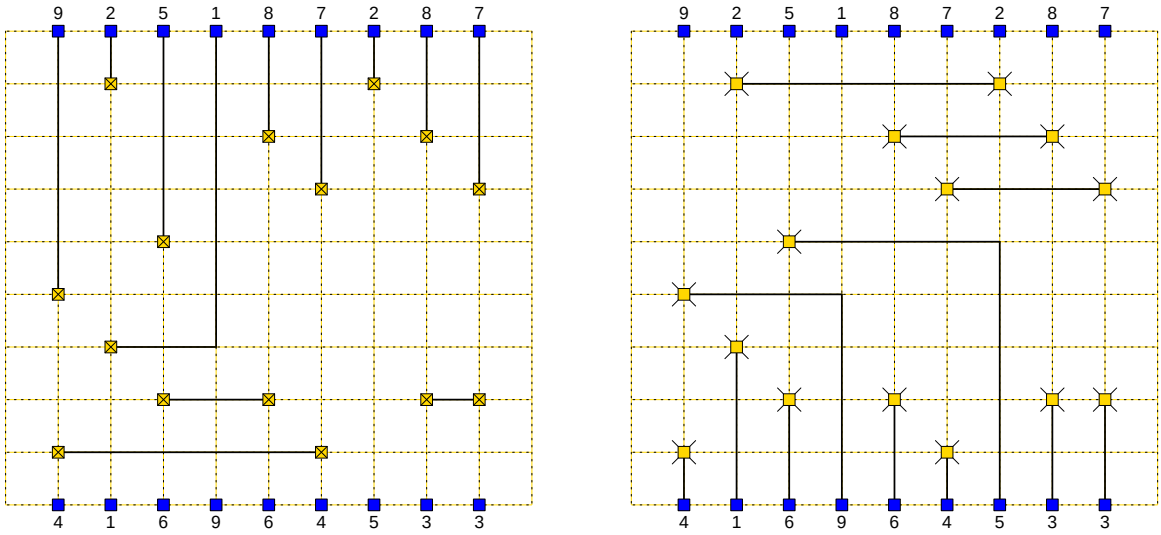
Multiprogramozott környezetben futó folyamatok futásidejének mérése során alapvetően kétfajta időt szokás mérni. A *user time* a ténylegesen számítással töltött idő, vagyis a processzor használatának mértéke. Ennél valamivel hosszabb a *real time*, a való világban a program indításától a befejeződéséig eltelt idő. Ebbe beleértendő a user time-on kívül lemezműveletekre fordított idő, az egyéb külső eseményre várakozások ideje (pl. a várakozás arra, hogy a felhasználó leüssön egy billentyűt), a szemétgyűjtéssel és egyéb memóriakezelési feladatokkal töltött idő; az egyéb, párhuzamosan futó programokra fordított idő, és az operációs rendszer programok közötti váltással, ütemezéssel eltöltött ideje.

Jól látható, hogy a két időfogalom közül a user time tükrözi jobban egy algoritmus gyorsaságát, mivel a real time számos olyan külső tagot tartalmaz, ami nem függ a vizsgált algoritmustól. Sőt, a mért real time két egymás utáni futtáskor lényegesen eltérő is lehet. Mi mégis úgy döntöttünk, hogy a kevésbé jellemző real time-ot mérjük. Ennek az volt a fő oka, hogy a Java programon belülről csak ezt lehetséges ezredmásodperc pontossággal lekérdezni, továbbá a Java nem biztosít lehetőséget arra, hogy csak bizonyos szálak vagy folyamatok idejét mérjük. Arra számítottunk, hogy real és a user time közti eltérés elhanyagolhatóan kicsi lesz az algoritmusok aszimptotikus viselkedésében tapasztalható eltéréshez képest.

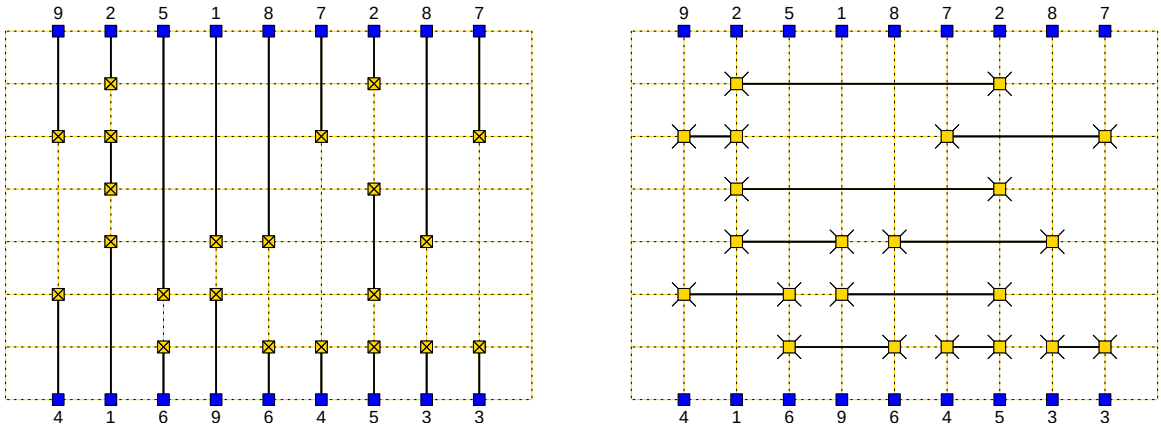
Ezért tehát olyan hardver- és szoftverkörnyezetet igyekeztünk teremteni, ahol a két időérték nem tér el lényegesen egymástól.

7.4. Eredmények

A teszteléshez a Recski-Strzyzewsky [12] és a *zChaff* SAT solvert [27] használó algoritmust használtuk. Mivel az előbbi algoritmus nem optimumkeresésre volt kihegyezve, elég könnyen talált az általunk kifejlesztett megoldó jobb megoldást. Ilyen megoldást mutatunk most be. Látható a 12. és a 13. ábrán, hogy a Recski-Strzyzewsky algoritmusnál akár kettővel is kisebb szélességet igényelt a huzalozási feladat megoldása.



(a)

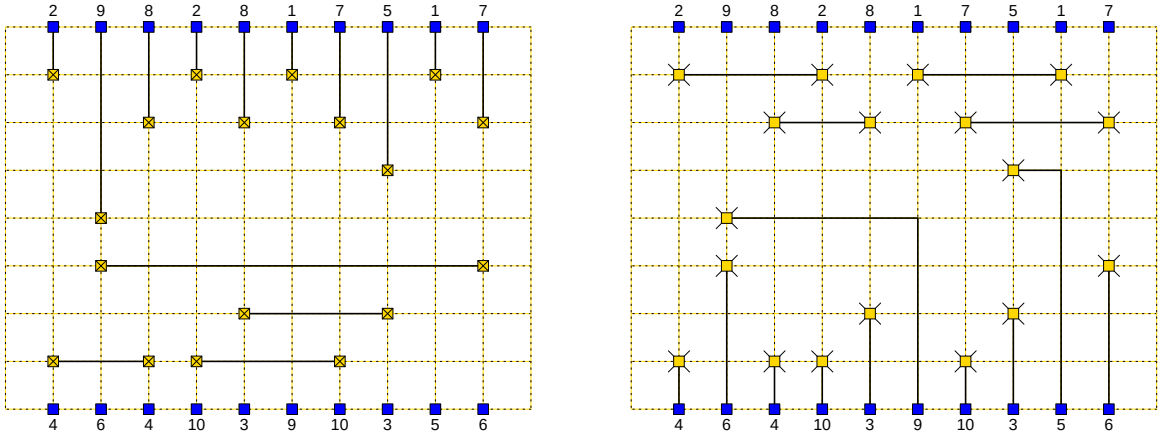


(b)

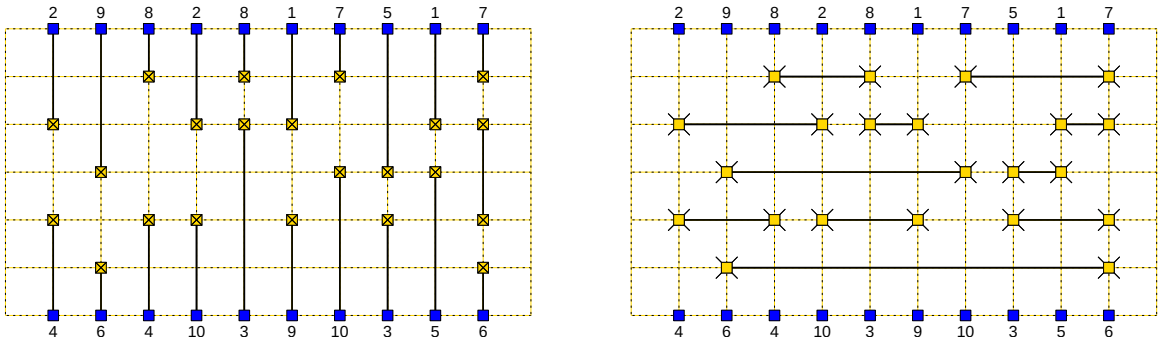
12. ábra.

8 hosszú csatornahuzalozási probléma megoldása

(a) Recski-Strzyzewsky algoritmussal (b) SAT solverrel



(a)



(b)

13. ábra.
9 hosszú csatornahuzalozási probléma megoldása
(a) Recski-Strzyzewsky algoritmussal (b) SAT solverrel

8. Konklúzió

Nagy bonyolultságú áramkörök egy fontos feladata a részletes VLSI huzalozás. Mi ehhez a problémakörhöz tettünk hozzá. Munkánk során kifejlesztettünk új algoritmusokat, és implementáltuk ezek egy részét, és egy már meglévőt is egy rendszerbe ([2]), ezzel növelve a rendszer erejét. A rendszer az általunk implementált algoritmusokkal szélesebb körű tesztelésre használható.

8.1. Kifejlesztett algoritmusok

Készítettünk egy genetikus algoritmust, ami megoldja a csatorna-huzalozás problémát kétrétegű modellben.

Egy másik algoritmust is kifejlesztettünk a huzalozási probléma megoldására, amely nagy hálózatok esetén nem a leghatásosabb, viszont kisebb tesztesetekre pontosabban megkapjuk, hogy milyen problémák oldhatók meg, és ezzel a gyors heurisztikák tökéletes megoldáshoz való közelségét is mérni tudjuk. Ez az algoritmus a kielégíthetőségi problémára való visszavezetésre épül, így – mivel napjainkban többszáz változós logikai függvényeket is hatásosan megoldanak a SAT solverek – akár 10–15 hosszúságú csatorna-huzalozásra is meg tudja adni az optimális (vagy legalábbis közel optimális, ha a solver nem garantál teljes keresést) választ.

Formálisan bizonyítottuk a kielégíthetőségre visszavezetés helyességét.

8.2. Implementációk

Készítettünk a programhoz egy ellenőrzőt, amely megmondja, hogy jó-e egy megoldás. Tesztelésnél alapvető fontosságú, hogy biztos jól működjenek a feladat megoldók, és ne csak sokkal később derüljön ki a hiba.

Egy teszteset generáló programot is mellékeltünk. File-okba írja a teszteseteket, így minden tesztelés futtatást meg lehet többször is ismételni.

Implementáltuk a Gallai algoritmus csatorna-huzalozásra használt változatát. Az általunk kifejlesztett SAT solver alapú algoritmust is implementáltuk, és az ötletet használtuk Prolog alapú algoritmus készítésére is.

Részben elkészítettük a Rivest-Fiduccia switchbox és háromrétegű csatorna-huzalozó algoritmust, de a tesztelésüket még nem tudtuk elkezdeni.

8.3. Továbbfejlesztési lehetőségek

A közeljövőben be kívánjuk fejezni az algoritmusok implementálást és részletes tesztelesét, az időeredményeket is összehasonlító grafikonokat kívánunk készíteni. Azt tapasztaltuk, hogy a kielégíthetőségre való visszavezetés már az iparban gyakori feladatoknál jóval kisebb problémákra is használhatatlanul lassú. Meg kívánjuk állapítani a pontos méretet, amelyre ez a módszer még elfogadható időn belül lefut, és meg kívánjuk vizsgálni, hogy a SAT-megoldók paramétereinek hangolása hogyan befolyásolja a keresési időt.

Irodalomjegyzék

[1] Recski, A.: Some polynomially solvable subcases of the detailed routing problem in VLSI design. Discrete Applied Math 155. (2001) 199–208

[2] Golda B., Laczay B., Megyeri Cs.: Huzalozó algoritmusok implementációja és vizsgálata, BME TDK, 2001.

[3] Gallai T. His unpublished results have been announced in Hajnal, A. and J. Surányi: 1958: Über die Anflösung von Graphen in vollständige Teilgraphen, Ann. Univ. Sci. Budapest Eötvös 1, 115–123.

[4] Szymanski, T. G. 1985. Dogleg channel routing is NP-complete, IEEE Trans. Computer-Aided Design of Integrated Circ. Syst. CAD-4, 31–41.

[5] Yoshimuta, T. and Kuh, E.S. "Efficient algorithms for channel routing", IEEE Trans. Computer-Aided Design of Integrated Circ. & Syst., vol. I. no. I. pp. 25-35, 1982.

[6] Kureichik V., Davidenko V. Genetic algorithm for restrictive channel routing. Proceeding of the 7th international conference on Genetic Algorithms, USA, MSU, East Lansing, 1997, p. 636-642.

[7] Altenberg, L. 1995. "The Schema Theorem and Prices Theorem" Foundations of Genetic Algorithms 3. ed. Darrell Whitley and Michael Vose. Morgan Kaufmann, San Francisco, pp. 23-49.

[8] Holland, J. 1975. "Adaptation In Natural and Artificial Systems" University of Michigan Press.

[9] Mitchell, M. 1998. "An Introduction to Genetic Algorithms" MIT Press.

[10] Goldberg, D. E. 1989. "Genetic Algorithms in Search, Optimization and Machine Learnin" Addison-Wesley

[11] *Java Virtual Machine Profiler Interface.* Sun Microsystems, 1999.

[12] Recski A. and F. Strzyzewski. *Integer programming and combinatorial optimization*, chapter "Vertex-disjoint channel routing on two layers", pages 397–405. University Waterloo Press, 1990.

[13] M. L. Brady and D. J. Brown. VLSI routing: Four layer's suffice. *Advances in Computing Research*, 2:245–258, 1984.

[14] Otto G. Folberth and Warren D. Grobman, editors. *VLSI: Technology and Design*, chapter "Device, Circuit, and Technology Scaling to Micron and Submicron Dimensions", pages 3–18. IEEE Press, New York, NY 10017, 1984.

[15] Otto G. Folberth and Warren D. Grobman, editors. *VLSI: Technology and Design*, chapter "VLSI Design Automation: An Introduction", pages 19–23. IEEE Press, New York, NY 10017, 1984.

[16] A. Frank. Disjoint paths in a rectilinear grid. *Combinatorica*, 2:361–371, 1982.

[17] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, Massachusetts, 1994.

[18] S. E. Hambrusch. *Channel routing in overlap models.* IEEE Trans. Computer-Aided Design of Integrated Circ. & Syst. CAD-4, 1985.

[19] T. C. Hu and Ernest S. Kuh. *VLSI Circuit Layout: Theory and Design*, chapter "Theory and Concepts of Circuit Layout", pages 3–19. IEEE Press, New York, NY 10017, 1985.

[20] M. Koebe and P. Dupont. Single-layer channel routing. *J. Inf. Process. Cybern.*, 7/8:339–354, 1988.

[21] A. S. LaPaugh. A polynomial time algorithm for optimal routing around a rectangle. *Proc. 21st FOCS Symp.*, pages 282–293, 1980.

[22] R. N. Noyce. Microelectronics. *Sci. Amer.*, 237, 1977.

[23] A. Recski. Minimax results and polynomial algorithms in VLSI routing. *Ann. Discrete Math*, pages 261–273, 1992.

[24] D. Szeszlér. Switchbox routing in the multilayer manhattan model. *Ann. Univ. Sci. Budapest Eötvös Sect. Math.*, 40:155–164, 1997.

[25] D. Szeszlér. Nagy bonyolultságú hálózatok huzalozása. Kézirat, 2000.

[26] Dorothea Wagner and Karsten Weihe. An animated library of combinatorial VLSI-routing algorithms. In *Symposium on Computational Geometry*, pages C28–C29, 1995.

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an Efficient SAT Solver At *39th Design Automation Conference, Las Vegas*, June 2001. `<http://ee.princeton.edu/~chaff/zchaff.php>`

[28] SICStus Prolog Homepage. `<http://www.sics.se/sicstus/>`

[29] P. Bruell, P. Sun, A "Greedy" Three Layer Channel Router At ICCAD-85 (*IEEE International Conference on Computer Aided Design*), November 18-21, 1985.

[30] R. L. Rivest, C. M. Fiduccia, A Greedy Channel Router Computer Aided Design Vol. 15, No. 3. (1983), pp 135-140

[31] J. R. Stenstrom, R. M. Mattheyses, Switch-Box Routing the Greedy Way At ICCAD-85 (*IEEE International Conference on Computer Aided Design*), November 18-21, 1985.