5. RUTAREA CIRCUITELOR CU RESURSE LIMITATE DE RUTARE

5.1 Introducere

În procesul de proiectare automatã a circuitelor VLSI sau de implementare a sistemelor digitale prin circuite FPGA, etapa urmãtoare celei de plasare a modulelor este *rutarea*. Rutarea este executatã cu ajutorul programelor de rutare, a cãror sarcinã este definirea precisã a cãilor pentru interconexiunile dintre pinii care sunt echivalenți din punct de vedere electric.

Rutarea necesitã în jur de 30% din timpul de proiectare, iar interconexiunile necesitã un procent ridicat din suprafața circuitelor [146]. Primii algoritmi de rutare automatã au fost dezvoltați pentru proiectarea circuitelor imprimate. Unele idei de bazã ale rutării automate rezultate de aici sunt valide în continuare, fiind adaptate pentru circuitele VLSI și circuitele FPGA.

Ca și în cazul problemei de plasare, toate formulãrile matematice ale rutării conduc la probleme NP-complete. Rutarea este descompusã de aceea într-o ierarhie de probleme mai simple, care se pot soluționa uneori într-un timp polinomial, sau sunt suficient de simple pentru ca metoda enumerării complete a soluțiilor sã fie practicã. Aceastã descompunere se realizeazã însã cu costul pierderii optimalității globale.

Pentru rutare se adoptã de obicei o abordare în douã etape: se executã mai întâi *rutarea globalã*, urmatã apoi de *rutarea detaliatã*. Obiectivul rutării globale este de a se elabora un plan de rutare astfel încât fiecare conexiune sã fie asignatã unor regiuni particulare de rutare, în timp ce se încearcã minimizarea unei funcții obiectiv date (de obicei o estimare a lungimii totale a conexiunilor). Rutarea detaliatã se aplicã apoi pentru fiecare regiune de rutare, și fiecãrei conexiuni i se asigneazã piste particulare de rutare.

5.2 Definirea problemei de rutare

Fiind dat un set de celule și porturile acestora (pinii de intrare, ieșire, ceas, alimentare și masã), un set de conexiuni (seturi de puncte care trebuie conectate împreunã din punct de vedere electric), și locațiile celulelor (obținute în urma procesului de plasare), rutarea constã în determinarea cãilor adecvate pentru interconexiunile dintre seturile de pini. Aceste cãi adecvate minimizează funcția obiectiv datã, supusã unor restricții. Restricțiile pot fi impuse de proiectant, de procesul de implementare, de tipul circuitului sau de stilul de proiectare. Exemple de restricții sunt distanța minimã dintre conexiunile adiacente corespunzãtoare semnalelor diferite, lãțimea minimã a conexiunilor de rutare, numãrul de straturi disponibile pentru rutare, întârzieri maxime

etc. Ca exemple de funcøii obiectiv se pot aminti reducerea lungimii totale a interconexiunilor, sau evitarea problemelor datorate întârzierilor semnalelor.

Conexiunile utilizate pentru rutarea circuitelor VLSI se obøin prin depunerea uniformã a unui metal pe un suport de siliciu, şi apoi eliminarea metalului nedorit pentru a se obøine firele de interconectare. În cazul circuitelor VLSI, pe lângã metal se mai utilizeazã şi polisiliciul pentru interconectare, pe un strat separat de stratul de metal. Cele douã straturi sunt separate printr-un strat de oxid izolant. Anumite tehnologii VLSI permit utilizarea a trei straturi pentru rutare. Douã dintre straturi conøin conductori de metal (numite de obicei metal-1 şi metal-2), iar al treilea strat conøine conductori de polisiliciu. Se utilizeazã orificii de trecere pentru conectarea conductorilor din straturi diferite.

În cazul circuitelor FPGA, conexiunile sunt predefinite, fiind formate din linii segmentate, legãturile dintre segmente fiind programabile. Majoritatea circuitelor FPGA dispun de linii de interconectare organizate în canale. Unele circuite au arhitecturi de rutare la care liniile de interconectare nu sunt segmentate, flexibilitatea fiind de aceea mult mai redusã. Rutarea acestor tipuri de circuite este dificilã. Implementãrile efectuate în cadrul tezei au fost realizate pentru aceste circuite FPGA cu resurse limitate de rutare.

5.3 Funcții de cost şi restricții

5.3.1 Funcții de cost şi restricții pentru rutarea globală

Rutarea globalã este diferitã pentru diferitele tipuri de circuite. În cazul *reøelelor de porøi*, regiunile de rutare constau din canale orizontale şi verticale. Canalele sunt regiuni dreptunghiulare cu pini amplasaøi pe marginile opuse ale regiunii. Capacitãtile de rutare disponibile în cadrul canalelor sunt fixe. O soluøie de rutare globalã nu trebuie sã depãşeascã capacitãøile canalelor. Dintre soluøiile posibile se alege cea care optimizeazã funcøia de cost datã. Funcøia de cost este de obicei o mãsurã a lungimii tuturor conexiunilor şi/sau o mãsurã a performanøei globale (întârzierile de interconectare pe cãile critice). Deoarece spaøiul de rutare este fix, obiectivul rutãrii globale este în acest caz verificarea fezabilitãøii rutãrii detaliate.

Pentru circuitele cu *celule standard*, regiunile de rutare sunt canale orizontale cu pini amplasaøi pe marginile de sus şi de jos ale canalelor. Rutarea globalã constã în asignarea conexiunilor acestor canale astfel încât sã se minimizeze congestia canalelor şi lungimea totalã a conexiunilor. Rutarea între canale este asiguratã prin celule de trecere inserate în cadrul rândurilor de celule. În acest caz, canalele nu au capacitãøi fixate în prealabil.

În cazul circuitelor cu *macro-celule*, dimensiunea şi forma celulelor este variatã. Aceasta conduce la regiuni de rutare neregulate. Aceste regiuni pot fi descompuse în canale orizontale şi verticale, şi uneori blocuri de comutare (regiuni dreptunghiulare cu pini pe toate cele patru margini). Identificarea acestor regiuni este un prim pas esenøial pentru rutarea globalã. Nici în acest caz, regiunile de rutare nu au capacitãøi fixate.

Atât pentru circuitele cu celule standard, cât şi pentru cele cu macro-celule, obiectivul rutãrii globale este minimizarea spaøiului de rutare necesar şi a lungimii totale a conexiunilor, asigurând în acelaşi timp succesul rutãrii detaliate ulterioare. Funcøia de cost este de aceea o mãsurã a spaøiului total de rutare. Restricøiile pot fi o limitã a numãrului maxim de piste pe canal şi/sau restricøii asupra performanøelor.

O problemã importantã care apare în toate cazurile este identificarea seturilor de rute cele mai scurte pentru conectarea pinilor conexiunilor individuale. Pentru conexiuni cu doi pini, problema este simplã şi se reduce la gãsirea cãii celei mai scurte

care conectează cei doi pini. Pentru conexiuni cu pini multipli, problema constă în găsirea celui mai scurt arbore Steiner care acoperă toți pinii unei conexiuni. Problemele arborelui Steiner sunt în general NP-complete [146]. Există însă algoritmi euristici care permit obținerea unor rezultate apropiate de cele optime într-un timp rezonabil.

5.3.2 Funcții de cost şi restricții pentru rutarea detaliată

Obiectivul principal al unui algoritm de rutare detaliată este obținerea rutării complet automate, fără intervenție manuală sau cu o intervenție minimă. Pentru implementarea unui circuit într-un spațiu minim, este esențială reducerea spațiului ocupat de interconexiuni. Lungimea conexiunilor individuale trebuie de asemenea redusă pentru a se satisface criteriile de performanță. Deci, obiectivul celor mai multor programe de rutare este realizarea rutării complete utilizând conexiuni de lungime cât mai redusă.

Modificările direcției căilor de rutare determină necesitatea utilizării orificiilor de trecere. Dacă se utilizează rutarea cu două straturi, toate segmentele orizontale sunt asignate unui strat, iar cele verticale celuilalt strat. Legătura între segmentele care aparțin aceleiași conexiuni este realizată prin orificii de trecere. Eficiența procesului de fabricație scade odată cu creșterea numărului orificiilor de trecere. Întârzierile datorate impedanței acestor orificii reduc de asemenea viteza. De aceea, reducerea numărului orificiilor de trecere este importantă.

Performanțele sunt afectate datoritã întârzierilor de semnal. Întârzierile introduse de porțile logice s-au redus considerabil, astfel încât întârzierile datorate interconexiunilor nu mai pot fi ignorate. De fapt, uneori întârzierile datorate interconexiunilor sunt dominante. Obiectivul programului de rutare este deci nu numai de a reduce lungimea totală a conexiunilor, dar şi păstrarea întârzierii maxime a fiecărei conexiuni sub o anumită valoare limită.

În sfârşit, programele de rutare trebuie să accepte circuite de dimensiuni foarte mari. De aceea, este esențial ca algoritmii utilizați să fie eficienți din punct de vedere al timpului de execuție şi dimensiunea memoriei utilizate să fie cât mai redusă.

Algoritmii de rutare detaliată trebuie să țină cont de un set dat de restricții. Principalele tipuri de restricții sunt: a) restricții de plasare, b) numărul straturilor de rutare, şi c) restricții geometrice.

a) Restricții de plasare. Numeroase programe de rutare se bazează pe o plasare fixă. Celulele sunt deci în locații predefinite şi nu se pot muta. Această metodă are ca rezultat soluții imperfecte şi poate determina ca unele conexiuni să nu poată fi rutate. Aceste restricții se aplică plăcilor imprimate, rețelelor de porți şi, într-o anumită măsură, celulelor standard.

În cazul rețelelor de porți, toate celulele au aceeaşi dimensiune. Celulele sunt plasate în locații fixe aranjate în rânduri, între celule fiind disponibile spații orizontale şi verticale pentru tutare. Spațiul de rutare este deci fix în acest caz. Ca şi în cazul rețelelor de porți, la circuitele cu celule standard există rânduri de celule cu înălțime fixă. Dimensiunea suprafeței de plasare nu este predeterminată. Problema este de a efectua toate conexiunile între celule utilizând lățimea minimă a spațiilor orizontale dintre rânduri, spații numite *canale*.

b) Numărul straturilor de rutare. Rutarea cu un singur strat este cea mai economică. În acest caz sunt necesari algoritmi de rutare în plan. Complexitatea acestor algoritmi este însă ridicată, şi în general nu pot asigura rutarea completă.

În cele mai multe aplicații, rutarea se realizează în două straturi. Metoda utilizată de obicei este rutarea H-V, în care un strat conține numai interconexiuni pe direcția orizontală, iar celălalt strat conține numai interconexiuni pe direcția verticală.

Utilizând acest concept, se garanteazã cã toate conexiunile unui circuit pot fi realizate cu condiøia ca spaøiul de rutare sã fie suficient de mare şi ca numãrul orificiilor de trecere sã nu fie limitat. Dezavantajul rutãrii H-V este cã numãrul orificiilor de trecere este mare. Eliminarea orificiilor redundante se realizeazã în cele mai multe aplicaøii de un post-procesor. Acesta asigneazã stratului orizontal segmentele de conexiuni verticale care nu sunt intersectate de segmente orizontale ale altor conexiuni, şi stratului vertical segmentele de conexiuni orizontale care nu sunt intersectate de segmente verticale ale altor conexiuni.

Tehnologiile avansate utilizeazã un numãr mai mare de straturi. Datoritã costurilor de producøie şi consideraøiilor de eficienøã, în cazul tehnologiilor MOS standard se preferã rutarea cu douã straturi, cu un strat de metal şi unul de polisiliciu.

c) Restricøii geometrice. În timpul rutãrii, trebuie respectate anumite distanøe, ca lãøimea minimã şi spaøierea cãilor de rutare, care sunt impuse de procesul tehnologic. Programele de rutare automatã trebuie sã ia în considerare toate restricøiile geometrice, astfel nefiind necesarã verificarea regulilor de proiectare. De obicei, aceasta se asigurã prin utilizarea unei grile echidistante. Interconexiunile sunt reprezentate prin linii şi sunt restrânse la poziøiile liniilor grilei. De aceea, lãøimea interconexiunilor şi separarea dintre acestea sunt constante.

5.4 Sinteza metodelor de rutare

Metodele utilizate sunt diferite pentru rutarea globalã şi pentru rutarea detaliatã. De asemenea, metodele de rutare globalã şi detaliatã diferã pentru diversele tipuri de circuite: reøele de porøi, celule standard, macro-celule, circuite FPGA. Problema de rutare a fost intens studiatã în literaturã [2], [3], [4], [5], [15], [19], [20], [21], [22], [27], [28], [34], [48], [49], [53], [54], [55], [60], [71], [81], [82], [93], [110], [112], [115], [123], [126], [140], [151], [159], [161], [166], [172], [188], [189].

5.4.1 Prezentarea sintetică a metodelor de rutare globală

În cazul *rutării globale*, metodele raportate în literaturã se pot împãrøi în patru categorii principale:

1) Metode secvenøiale;
2) Metode de cãutare aleatoare, ca de exemplu cãlirea simulatã;
3) Metoda programãrii liniare;
4) Metoda descompunerii ierarhice.

O etapã importantã de pregãtire a rutării globale este *determinarea regiunilor de rutare*. Au fost descrise diferite metode pentru rezolvarea acestei probleme. Cai şi Otten [34] au descris un algoritm pentru conversia joncøiunilor de tip + în joncøiuni de tip T în cazul circuitelor cu macro-celule. Cai şi Wong [35] au elaborat un algoritm pentru definirea regiunilor de rutare ca şi canale dreptunghiulare şi blocuri de comutare, astfel încât numãrul blocurilor de comutare sã fie minimizat. Aceeaşi autori au prezentat în [36] un algoritm pentru descompunerea regiunilor de rutare în canale dreptunghiulare, minimizând în acelaşi timp numãrul de canale în formã de L.

O altã problemã importantã întâlnitã în timpul rutării globale, ca şi la rutarea detaliatã, este cea a construirii unui *arbore Steiner* pentru fiecare conexiune. Existã numeroase lucrãri asupra acestui subiect, fiind descrise metode euristice pentru gãsirea unui arbore Steiner apropiat de cel optim, într-un timp rezonabil [3] [45] [60] [84] [91] [99] [100] [148].

Sarrafzadeh şi Wong [148] au descris o strategie ierarhicã de jos în sus pentru construirea unui arbore Steiner. La fiecare nivel al ierarhiei, este enumeratã o colecøie de arbori Steiner parøiali utilizând un algoritm pentru determinarea arborelui de acoperire minim. La fiecare nivel superior al ierarhiei, arborii nivelului precedent sunt uniøi, eliminând muchiile duplicate şi evitând crearea ciclurilor. Arborele corespunzãtor nivelului din vârful ierarhiei este arborele Steiner dorit pentru conexiunea curentã.

O altã metodã elaboratã de Cong *et al.* [60] utilizeazã o modificare a algoritmilor pentru construirea arborelui de acoperire minim şi a arborelui Steiner în scopul construirii unui arbore de rutare cu lungime limitatã şi întârziere de interconectare limitatã. Kahng şi Robins [99] utilizeazã o euristicã iterativã pentru determinarea unor puncte Steiner optime. Aceste puncte sunt determinate şi adãugate unul câte unul, într-un mod *greedy*, pânã când nu se mai poate obøine o reducere a lungimii totale a arborelui. Rezultatele experimentale au arãtat cã arborele Steiner construit astfel are un cost care este mãrginit la $\frac{4}{3}$ din costul arborelui Steiner optim.

Chiang *et al.* [54] au propus utilizarea *arborilor Steiner ponderaøi* pentru rutarea globalã. Suprafaøa de rutare este divizatã în regiuni, fiecare având o pondere datã de numãrul de terminale din regiune, densitatea acesteia şi capacitatea marginilor regiunii. Ponderile regiunilor se modificã în mod dinamic. În fiecare pas al rutãrii se construieşte în plan un arbore Steiner ponderat. Metoda de rutare propusã minimizeazã simultan lungimea conexiunilor şi densitatea regiunilor.

Aceeaşi autori au descris în [53] o metodã de rutare globalã bazatã pe *arbori Steiner min-max*, care au muchiile de pondere maximã minimizate. Conexiunile sunt ordonate mai întâi pe baza criteriilor de lungime, multiplicitate şi importanøã. Pentru fiecare conexiune se construieşte apoi un arbore Steiner min-max.

Metoda de cãlire simulatã a fost utilizatã şi pentru rutarea globalã, mai întâi de Vecchi şi Kirkpatrick, iar apoi de Sechen şi Sangiovanni-Vincentelli [151], în cadrul pachetului de programe TimberWolf. Acest pachet este destinat pentru plasarea şi rutarea globalã a circuitelor cu celule standard. Dupã o fazã de plasare iniøialã, se executã rutarea globalã, iar apoi se încearcã îmbunãtãøirea plasãrii prin interschimbarea aleatoare a unor celule vecine. Dupã fiecare interschimbare, se rerutează conexiunile afectate de interschimbare.

Pentru rutarea circuitelor VLSI au fost utilizate metode de *programare liniarã*. Aceste metode au însã douã limitãri importante. În primul rând, dimensiunea problemelor este importantã; acestea pot conøine mii (sau chiar sute de mii) de restricøii şi variabile. În al doilea rând, pentru soluøionarea acestor probleme se utilizează de obicei o metodã simplex, care necesitã pentru soluøionarea problemei de optimizare un numãr de iteraøii egal cu numãrul de restricøii. Din acest motiv, timpul de execuøie devine excesiv de lung. Vannelli [172] a descris o modificare a metodei punctului interior a lui Karmakar pentru rezolvarea problemei de rutare globalã formulatã ca o problemã de programare liniarã. Spre deosebire de abordãrile bazate pe metoda simplex, la care dimensiunea problemei rãmâne aceeaşi pentru toate iteraøiile, metoda propusã permite reducerea semnificativã a dimensiunii problemei pe mãsura evoluøiei algoritmului.

În scopul reducerii complexitãøii problemei de rutare globalã, au fost propuse *metode ierarhice*, care efectuează o descompunere ierarhicã a problemei în mai multe subprobleme, acestea fiind soluøionate individual. Soluøiile parøiale sunt combinate apoi pentru a produce o soluøie a problemei globale originale. În literaturã au fost descrise numeroase formulãri ierarhice ale problemei de rutare globalã. Metodele propuse independent de Marek-Sadowska şi Lauther sunt similare, soluøiile generate fiind superioare celorlalte metode ierarhice raportate [146]. Metoda propusã de Marek-Sadowska şi Lauther este utilizatã de programele PROUD [170] şi BEAR.

Viteza circuitelor este influenøatã în mare mãsurã de întârzierile datorate interconexiunilor. Aceasta a determinat apariøia unor sisteme de proiectare în care accentul se pune pe *maximizarea performanøelor*, în special pe reducerea întârzierilor de inter-

conectare. Pentru a se evita întârzierile mari pe cãile critice, au existat încercãri pentru a controla etapa de rutare globalã de cerinøele de timp ale circuitelor. Minimizarea lungimii totale de interconectare nu va conduce neapãrat la întârzieri minime de interconectare. Numãrul schimbãrilor de direcøie ale interconexiunilor, ca şi alte caracteristici electrice ale acestora sunt la fel de importante ca şi lungimea conexiunilor.

Jackson, Kuh şi Marek-Sadowska au raportat o metodã ierarhicã pentru rutarea controlatã de restricøiile de timp [60]. O altã metodã de rutare globalã controlatã de cerinøele de reducere a întârzierilor a fost raportatã de Prasitjutrakul şi Kubitz, în care rutarea globalã este integratã cu un analizor de timp [146]. Procesul de rutare este controlat de întârzierile de interconectare.

Cong *et al.* [60] au raportat o metodã de rutare globalã cu minimizarea simultanã a lungimii totale de interconectare şi a întârzierii maxime de interconectare. Aceastã metodã este bazatã arbori de rutare cu razã limitatã, pentru construcøia acestora utilizându-se euristici bazate pe algoritmul lui Prim pentru arbori de acoperire minimi. Aceeaşi autori au elaborat o metodã care minimizeazã simultan lungimea totalã de interconectare (indicatã de costul arborelui) şi întârzierea maximã (indicatã de raza arborelui).

Shragowitz şi Keal au formulat rutarea globalã cu ajutorul unui model de *reøea de flux* [146]. Soluøia este obøinutã în douã etape. În prima etapã, este soluøionatã problema fãrã a øine cont de restricøii. În a doua etapã, se aplicã o procedurã iterativã în care, la fiecare iteraøie, este rerutatã conexiunea pentru care rezultã o reducere maximã a fluxului.

În metoda propusã de Clow, se utilizeazã un algoritm de cãutare a liniilor pentru rutarea circuitelor cu celule generale, evitându-se astfel utilizarea unei grile [146]. Algoritmul este o generalizare a algoritmului de cãutare *A** des utilizat în inteligenøa artificialã. Se determinã o cale între o sursã *s* şi o destinaøie *d* prin adãugarea câte unui segment de linie la un moment dat, începând de la sursa *s*, pânã când se ajunge la destinaøia *d*. Pentru aceasta se construieşte un graf cu sursa *s* şi destinaøia *d*, adãugând câte o muchie, pânã când se stabileşte o cale minimã de la *s* la *d*. Muchiile corespund segmentelor de linie, iar vârfurile corespund punctelor de intersecøie ale liniilor. Fiecãrei muchii i se asignează un cost egal cu lungimea segmentului de lini corespunzãtor. Cãutarea cãii celei mai scurte de la sursã la destinaøie se executã utilizând algoritmul *A**.

În mod tradiøional, rutarea globalã a fost utilizatã pentru elaborarea unui plan de rutare care va fi executat de rutarea detaliatã. O altã utilizare a rutării globale este pentru *verificarea soluøiilor obøinute în urma plasării*. În timpul plasării, rutarea globalã este utilizatã pentru controlul procedurii de plasare în sensul producerii unei plasări rutabile. Una din lucrările elaborate în acest sens se datorează lui Shragowitz *et al.*, în care se descrie un algoritm constructiv de plasare controlat de rutarea globalã [146]. Circuitul se considerã împărøit în felii, iar soluøia plasării este construitã pentru câte o felie la un moment dat, pânã când toate celulele sunt plasate. De fiecare datã când este plasat un grup de celule, se executã rutarea globalã pentru conectarea celulelor plasate recent cu cele plasate anterior şi pentru a rezerva resursele de rutare necesare. În cadrul pachetului de programe TimberWolf [151], este de asemenea combinatã plasarea cu rutarea globalã.

Pentru rezolvarea problemei de rutare globalã au fost utilizate şi *metode netradiøionale:* reøele neuronale artificiale, algoritmi paraleli, evoluøia simulatã. Shih *et al.* [154] au utilizat o reøea neuronalã artificialã cu douã straturi. Un strat este utilizat pentru minimizarea lungimii conexiunilor şi pentru obøinerea unei distribuøii uniforme a conexiunilor în cadrul canalelor de rutare. Al doilea strat este utilizat pentru a impune restricøiile privind capacitãøile canalelor.

Rutarea este în general un proces mare consumator de timp. Majoritatea soluøiilor propuse permit o implementare paralelã. Rose [140] a descris un algoritm de rutarea globalã pentru celule standard, *LocusRoute*, şi implementarea paralelã a acestui

algoritm. În implementarea paralelã raportatã, Rose a descris douã strategii pentru paralelizarea procesului de rutare. Prima strategie constã în rutarea mai multor conexiuni în paralel. În a doua strategie, se evaluează în paralel mai mulți arbori de rutare pentru fiecare conexiune. A fost raportatã o creștere de vitezã semnificativã cu ambele strategii.

5.4.2 Prezentarea sintetică a metodelor de rutare detaliată

Rutarea detaliatã poate fi împărțitã în *rutare detaliatã generalã* și *rutare detaliatã cu restricții*. Rutarea detaliatã cu restricții pote fi împărțitã la rândul ei în rutare prin *canale* și rutare prin *blocuri de conexiune*.

Una din metodele cele mai utilizate de *rutare detaliatã generalã* este *rutarea labirint*, algoritmul cel mai cunoscut fiind cel elaborat de Lee. Acest algoritm garantează gãsirea cãii între douã puncte, dacã o asemenea cale existã. În plus, calea gãsitã va fi cea mai scurtã. Dezavantajul acestui algoritm este necesarul ridicat de memorie și timp de execuție. Pentru eliminarea acestor dezavantaje au fost propuse diferite tehnici. Hadlock a sugerat o nouã tehnicã de etichetare a celulelor bazatã pe numere de deturnare [146]. Creșterea de vitezã obținutã este semnificativã. Soukup a sugerat adãugarea unei cãutãri în adâncime. Algoritmul obținut este de 10-50 de ori mai rapid decât algoritmul Lee, dar nu se garantează gãsirea cãii celei mai scurte.

O altã metodã de rutare detaliatã generalã este rutarea prin *cãutarea liniilor*. Prin aceastã metodã se eliminã necesarul ridicat de memorie al algoritmului Lee, deoarece spațiul de rutare nu este memorat ca o matrice, ci este reprezentat prin segmente de linii. Spre deosebire de algoritmul Lee, în acest caz se efectuează cãutarea în adâncime. Principalii algoritmi au fost propuși de Mikami și Tabuchi, respectiv Hadlock.

În literaturã au fost descrise diferite arhitecturi hardware specializate pentru rezolvarea problemelor de proiectare fizicã. Au fost raportate tablouri de procesoare dedicate pentru implementarea algoritmilor de rutare prin cãutarea cãii celei mai scurte. Iosupovici a propus un tablou bidimensional de procesoare SIMD pentru implementarea algoritmilor de rutare bazați pe reprezentarea suprafeței de rutare sub forma unei grile. Au fost propuse de asemenea tablouri de procesoare mai generale care pot fi programate pentru implementarea rutării, ca și pentru alte probleme de proiectare fizicã [146].

Pentru *rutarea prin canale*, Deutch a propus un algoritm pentru evitarea buclelor în graful constrângerilor verticale și pentru reducerea densității canalului. Aceasta se realizează prin divizarea segmentelor orizontale ale unei conexiuni, cu efectul minimizării numãrului de piste orizontale. Divizarea orizontalã a unei conexiuni poate fi realizatã numai în pozițiile terminalelor. Algoritmul Deutch divizează fiecare conexiune multipin în segmente orizontale individuale.

O euristicã de tip *greedy* pentru rutarea prin canale a fost propusã de Rivest și Fiduccia. Acest algoritm rutează canalul coloanã cu coloanã începând din stânga. În fiecare coloanã algoritmul aplicã o secvențã de euristici inteligente pentru maximizarea numãrului de piste disponibile în urmãtoarea coloanã. Nu se utilizează constrângeri orizontale sau verticale, deciziile fiind luate local, la nivelul coloanei. Rutarea este terminatã întotdeauna, uneori fiind necesare coloane adiționale la sfârșitul canalului.

Un algoritm de rutare prin canale bazat pe *sortare* a fost propus de Chaudry și Robinson [47]. Aceastã metodã presupune cã interconexiunile pot fi rutate nu numai orizontal și vertical, ci și la 45° și 135°. Algoritmul utilizează sortarea bubble. Într-un pas al acestei metode de sortare se interschimbã o pereche de numere o singurã datã, în cazul în care acestea nu sunt în ordinea corectã.

Pentru rutarea prin canale cu straturi multiple au fost propuse diferite tehnici. Asemenea tehnici au fost propuse de Braun *et al.* [24], și acestea au fost implementate într-un program de rutare numit Chameleon. Acest program constã din douã etape, un

program de partiøionare şi unul de rutare detaliatã. Rolul programului de partiøionare este de a diviza problema în subprobleme pentru două şi trei straturi astfel încât suprafaøa globalã a canalului sã fie minimizatã.

O altă abordare a problemei de rutare prin canale cu trei straturi, bazatã pe ideea transformării unei soluøii cu două straturi într-una cu trei straturi a fost descrisã de Cong *et al.* [61]. Această transformare constă din mai multe etape care pot fi formulate ca probleme de planificare, rutare labirint, respectiv problema căii celei mai scurte. Deoarece aceste probleme au o complexitate polinomială, rutarea cu trei straturi poate fi rezolvatã într-un mod optim. Cele mai multe din aceste tehnici pot fi extinse şi pentru patru straturi.

O euristică pentru rutarea prin canale care asignează interconexiunile pistă cu pistă într-un mod *greedy* a fost propusă de Ho *et al.* [92]. Structura de date şi strategia utilizatã sunt simple şi pot fi generalizate pentru obøinerea unei clase de euristici pentru rutarea prin canale. Acest algoritm are posibilitatea de revenire prin care cresc şansele de efectuare a rutării complete cu un număr minim de piste.

5.5 Rutarea globală

Rutarea globală este etapa pregătitoare pentru rutarea detaliatã. De obicei, rutarea globală este executatã în scopul elaborării unui plan de rutare pentru rutarea detaliatã. Rutarea globală este formulatã în mod diferit pentru diferitele tipuri de circuite: reøele de porøi, celule standard, macro-celule. În această secøiune se prezintă unele metode de rutare globală pentru aceste tipuri de circuite. Rutarea circuitelor FPGA este prezentatã în secøiunea 5.7.

În secøiunea 5.5.1 se prezintă definirea şi ordonarea canalelor de rutare. Este prezentatã de asemenea reprezentarea regiunilor de rutare. În secøiunea 5.5.2 se prezintă metode de rutare secvenøială: rutarea globală prin metoda parcurgerii labirintului, rutarea globală prin arbori Steiner ponderaøi, rutarea globală orientatã pe performanøe. În secøiunea 5.5.3 se prezintă rutarea globală prin metoda călirii simulate. În secøiunea 5.5.4 se prezintă rutarea globală prin metoda programării întregi.

5.5.1 Regiuni de rutare

Definirea regiunilor de rutare constă în partiøionarea spaøiului de rutare într-un set de regiuni dreptunghiulare care nu se intersectează, numite *canal*e. Pot exista două tipuri de canale: orizontale şi verticale. În cele mai multe cazuri, canalele orizontale şi verticale pot veni în contact prin intersecøii în formă de T.

Definirea şi ordonarea canalelor este o parte esenøială a procesului de proiectare fizică, reprezentând legătura între plasare, rutarea globală şi rutarea detaliatã.

5.5.1.1 Conversia joncţiunilor canalelor

În cazul circuitelor formate din macro-celule, pot apare trei tipuri de joncøiuni ale canalelor: de tip L, de tip T şi de tip +. Joncøiunile de tip L apar la coløurile suprafeøei de rutare. Pentru asemenea joncøiuni, ordonarea nu are un impact asupra rutării detaliate. Pentru joncøiunile de tip T, canalul care reprezintă baza literei T trebuie rutat înaintea canalului care reprezintă linia transversală a literei T. Joncøiunile de tip + sunt mai complexe şi necesită utilizarea programelor de rutare pentru blocuri de comutare (switchbox). Pe de altă parte, în cazul joncøiunilor de tip L şi T se pot utiliza programe de rutare detaliatã pentru canale, care sunt unele din cele mai intens inves-

tigate metode de rutare. De aceea, este avantajoasã transformarea tuturor joncøiunilor de tip + în joncøiuni de tip T.

Existã douã moduri de conversie a unei joncøiuni + într-o joncøiune T (Figura 5.1):

a) conversie orizontalã, când se divizeazã canalul vertical, şi
b) conversie verticalã, când se divizeazã canalul orizontal.

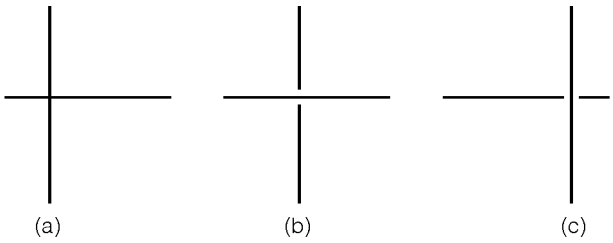

Figura 5.1. Conversia joncøiunilor. (a) Joncøiune +. (b) Conversie orizontalã. (c) Conversie verticalã.

Conversia joncøiunilor + în joncøiuni T trebuie efectuatã cu atenøie, pentru a nu crea cicluri în graful corespunzãtor al restricøiilor de ordonare [146]. Acest lucru este ilustrat în Figura 5.2.

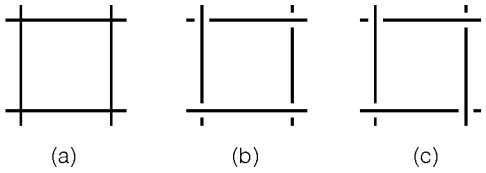

Figura 5.2. Conversia intersecøiilor. (a) O configuraøie de canale. (b) O conversie fãrã cicluri. (c) O conversie care introduce cicluri.

Dupã conversia tuturor joncøiunilor de tip + în canale de tip T, restricøiile de ordonare ale canalelor sunt puse în evidenøã printr-un graf direcøionat numit *graful restricøiilor de ordonare* (Figura 5.3).

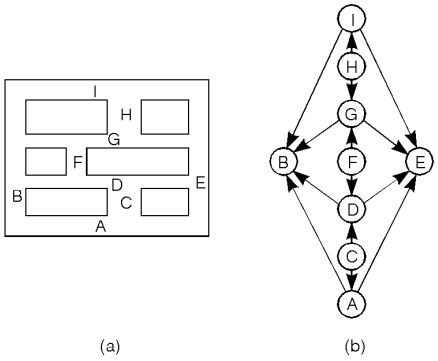

Figura 5.3. Graful restricøiilor de ordonare. (a) Structura canalelor. (c) Graful restricøiilor de ordonare.

Pentru a minimiza efectele negative pe care le pot avea conversiile canalelor asupra rutabilitãøii pe ansamblu, se utilizează urmãtoarele douã criterii [34]:

1. *Criteriul izolãrii cãilor critice.* Obiectivul acestui criteriu este de a proteja cãile critice ale grafurilor de poziøie ale canalelor de canalele învecinate. Un graf de poziøie al canalelor verticale (orizontale) este un graf bipartit reprezentând adiacenøele verticale (orizontale) între blocuri şi canalele de rutare. Graful de poziøie al canalelor verticale (orizontale) are câte un vârf pentru fiecare bloc şi fiecare canal orizontal (vertical). Există o muchie de la vârful *b* la vârful *c* dacã şi numai dacã marginea de jos (din dreapta) a blocului *b* se învecinează cu canalul *c*. În graful de poziøie al canalelor orizontale (verticale) fiecãrui vârf corespunzãtor unui bloc i se asignează o pondere pozitivã indicând lãøimea (înãløimea) blocului corespunzãtor. Fiecãrui vârf corespunzãtor unui canal i se asignează o pondere pozitivã indicând lãøimea canalului respectiv. Lungimea cãii critice din graful de poziøie al canalelor verticale (orizontale) este egalã cu înãløimea (lãøimea) zonei de rutare a circuitului.

   Criteriul cãii critice înceracã executarea conversiei în direcøia cãii critice, în scopul reducerii lungimii canalelor învecinate perpendiculare pe direcøia cãii critice, divizând astfel aceste canale (Figura 5.4). Un asemenea criteriu va determina de asemenea reducerea lãøimii canalelor de-a lungul cãii critice.

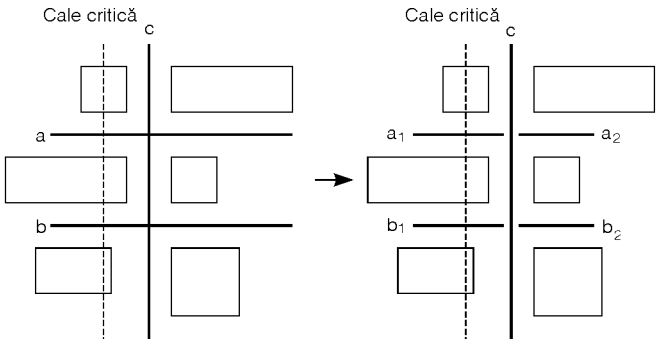

Figura 5.4. Ilustrarea criteriului cãilor critice.

2. *Criteriul fluxului major.* Conversia canalelor este executatã dupã rutarea globalã. Astfel, numãrul de interconexiuni din fiecare canal este cunoscut înaintea începerii procesului de conversie. În scopul minimizãrii numãrului de schimbãri ale direcøiei interconexiunilor de-a lungul canalelor, se divizează canalul mai îngust (Figura 5.5).

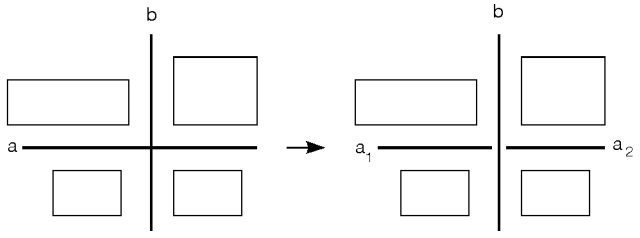

Figura 5.5. Ilustrarea criteriului fluxului major. Numãrul de conexiuni ale canalului *b* este presupus mai mare decât cel al canalului *a*.

Pentru fiecare joncøiune de tip +, se utilizează cele douã criterii pentru a se calcula valoarea unei funcøii de câştig. Această funcøie recompensează conversiile joncøiunilor care favorizează criteriul de izolare a cãii critice şi cel al fluxului maxim. Ast-

fel, pentru fiecare joncøiune de tip + adiacentã cu un segment de canal care se aflã pe o anumitã cale criticã se adaugã un premiu pentru conversia în direcøia cãii respective. De asemenea, pentru fiecare joncøiune de tip + se adaugã un premiu pentru conversia în direcøia canalului cu fluxul maxim al interconexiunilor. Structura optimã a canalelor este cea cu suma cea mai mare a premiilor de conversie a joncøiunilor.

Cai şi Otten [34] au descris un algoritm pentru conversia joncøiunilor de tip + în canale de tip T.

5.5.1.2 Ordonarea canalelor

Dupã asignarea tuturor conexiunilor unor canale individuale, etapa finalã a rutãrii constã în asignarea conexiunilor unor piste individuale din cadrul fiecãrui canal, deci executarea rutãrii detaliate. Canalele sunt rutate de obicei unul câte unul, într-o ordine specificã. Ordonarea canalelor este un pas intermediar important executat înaintea rutãrii detaliate şi dupã rutarea globalã. Acest pas este necesar pentru a specifica ordinea în care trebuie rutate canalele de cãtre programul de rutare detaliatã.

Ordinea în care trebuie rutate canalele este impusã de faptul cã poziøia pinilor trebuie fixatã înaintea rutãrii detaliate a canalului respectiv. Astfel, dintre cele douã canale ale unei joncøiuni de tip T canalul care reprezintã baza literei T trebuie rutat înaintea canalului care reprezintã linia transversalã a literei T, din urmãtoarele douã motive [146]:

1. Pentru rutarea canalului care reprezintã linia transversalã, sunt necesare informaøii despre pinii din joncøiunea T, deci despre conexiunile care trec prin joncøiune. Aceasta necesitã rutarea prealabilã a canalului care reprezintã baza literei T.
2. Atunci când se rutează canalul care reprezintã baza literei T, este posibil sã se constate cã trebuie mutate blocuri de la stânga (din partea de sus) şi/sau de la dreapta (din partea de jos) a acelui canal în scopul asigurãrii unui spaøiu pentru piste suplimentare. Aceasta va schimba poziøiile pinilor din cadrul canalului canalul care reprezintã linia transversalã a literei T. Acesta este un alt motiv pentru rutarea în ordinea indicatã.

Pentru ordonarea canalelor, se construieşte un graf al restricøiilor de ordonare, dupã cum urmeazã. Fiecare canal este reprezentat printr-un vârf. În cadrul grafului există un arc (i, j) dacã şi numai dacã canalele i şi j se întâlnesc într-o joncøiune T în care i este canalul care reprezintã baza literei T, iar j este canalul care reprezintã linia transversalã (Figura 5.3).

5.5.1.3 Reprezentarea regiunilor de rutare

Dupã definirea regiunilor de rutare, se construieşte un *graf de rutare*. Există trei metode generale pentru construirea acestui graf.

1. Utilizarea unui graf de conectivitate al canalelor $G = (V, E)$, unde fiecare canal este reprezentat printr-un vârf. Fiecare muchie modelează adiacenøa dintre canalele corespunzãtoare (Figura 5.6). Vârfurilor li se pot asigna ponderi pentru a indica numãrul de conexiuni care trec prin canal şi/sau numãrul de piste disponibile în acel canal. De notat cã graful de conectivitate al canalului este graful restricøiilor de ordonare în care se eliminã direcøiile arcelor.
2. Utilizarea unui graf al gâtuirilor $G = (V, E)$, unde se modelează prin vârfuri numai blocurile de interconectare. Există o muchie $(u, v) \in E$ dacã şi numai dacã blocurile de interconectare corespunzãtoare se aflã pe pãrøile opuse ale aceluiaşi canal orizontal sau vertical. Gâtuirile sunt reprezentate de canalele de rutare.

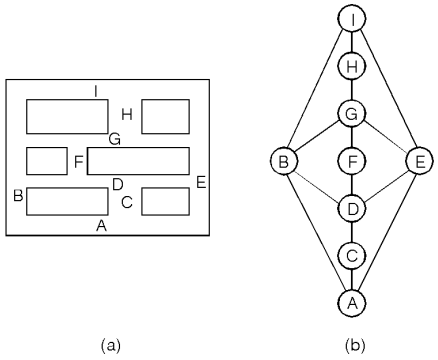

Figura 5.6. Graf de conectivitate al canalelor. (a) O amplasare pentru macro-celule. (b) Graful corespunzător de conectivitate.

Conceptul blocurilor de interconectare și a regiunilor de gâtuire este ilustrat în Figura 5.7(a), unde blocurile de interconectare sunt hașurate. Graful corespunzător este prezentat în Figura 5.7(b). Muchiile din acest graf modelează canale orizontale și verticale.

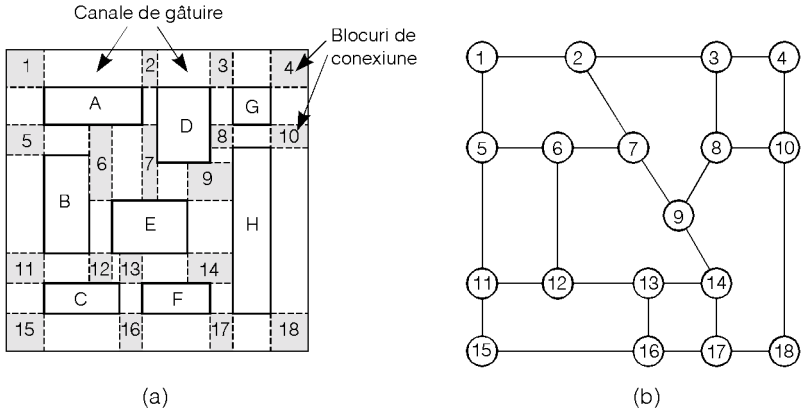

Figura 5.7. Graf al gâtuirilor. (a) O amplasare pentru macro-celule. (b) Graful corespunzător de gâtuire.

3. Utilizarea unui graf de caroiaj $G = (V, E)$, unde vârfurile modelează celule globale, iar muchiile modelează adiacențele dintre aceste celule (Figura 5.8). Pentru rutarea cu două straturi, fiecărui vârf i se asignează două cifre indicând numărul de piste orizontale și verticale disponibile.

5.5.2 Rutarea globală secvențială

Rutarea globală secvențială este metoda cea mai des utilizată. După ce au fost identificate canalele de rutare și a fost construit graful de rutare corespunzător, rutarea globală se continuă astfel. Pentru fiecare conexiune, se marchează vârfurile grafului de conectivitate al canalelor în care conexiunea respectivă are pini. Deci, rutarea conexiunii se reduce la identificarea unui arbore (de preferință al celui mai scurt) care acoperă vârfurile marcate.

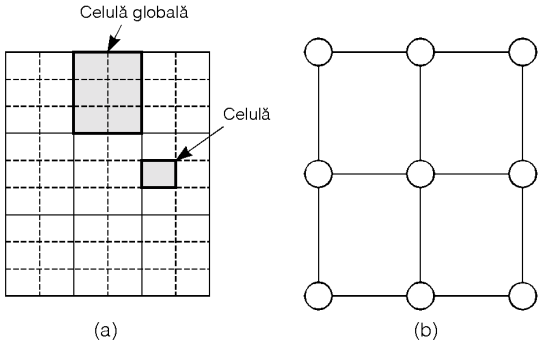

Figura 5.8. Graf de caroiaj. (a) Un caroiaj bidimensional. (b) Graful de caroiaj corespunzător unde fiecare celulă globală este modelată printr-un vârf.

Dacă conexiunea are pini doar în două vârfuri ale grafului, problema se reduce la determinarea căii celei mai scurte între vârfurile marcate. Dacă graful este un graf de caroiaj, se poate utiliza algoritmul lui Lee, care va fi prezentat în secțiunea 5.6. Pentru toate cele trei modele de grafuri, se poate utiliza algoritmul lui Dijkstra pentru determinarea căii celei mai scurte. Acest algoritm este prezentat în Figura 5.9.

Algorithm *Dijkstra* (*s*, *G*);
/* *s*: un vârf sursă; */
/* *G*: un graf ponderat; */
/* D_i: distanța cea mai scurtă estimată de la *s* la nodul *i*; */
/* d_{ij}: pondere a muchiei (*i*, *j*) sau distanța între nodurile *i* și *j*;*/
/* *M*: set de noduri marcate permanent. */
begin
1. /* Inițializare */
$M \leftarrow s$;
$D_s \leftarrow 0$;
forall ($j \in V(G)$) **do** $D_j \leftarrow d_{sj}$ **endfor**;
2. /* Determinarea nodului următor cel mai apropiat */
Se caută un nod $i \notin M$ astfel încât $D_i = \min_{j \notin M} D_j$;
$M \leftarrow M \cup i$;
if (*M* conține toate nodurile) **then** STOP; **endif**;
3. /* Actualizare */
forall ($j \notin M$) **do** $D_j \leftarrow \min_i (D_j, D_i + d_{ij})$ **endfor**;
goto 2;
end.

Figura 5.9. Algoritmul Dijkstra pentru determinarea căii celei mai scurte.

În general, conexiunile au mai mult de doi pini. Determinarea căii celei mai scurte care acoperă trei sau mai multe noduri constituie problema arborelui Steiner. Această problemă este de importanță deosebită pentru rutarea secvențială și este prezentată în secțiunea următoare.

5.5.2.1 Problema arborelui Steiner

Fie *M* un set de vârfuri marcate. Un arbore care conectează toate vârfurile din *M* ca și alte vârfuri din *G* care nu sunt în *M* este numit un *arbore Steiner*. Un *arbore Steiner minim* este un arbore Steiner cu o lungime minimă.

Problema arborelui Steiner este NP-completă. De aceea, în locul determinării unui arbore Steiner minim, se utilizează euristici pentru identificarea cât mai rapidă a unui arbore de lungime rezonabilă, și nu neapărat de lungime minimă. În literatură au fost publicate diferite metode euristice pentru determinarea arborelui Steiner [3] [45] [60] [84] [91] [99] [100] [148].

Cele mai multe euristici utilizează o variantă modificată a algoritmului Dijkstra pentru determinarea căii celei mai scurte sau o variație a algoritmului Lee pentru rutarea de tip labirint. De obicei, o asemenea euristică este de tip greedy și se execută astfel. Se selectează mai întâi unul din vârfurile marcate. Se identifică apoi calea cea mai scurtă la oricare din vârfurile marcate rămase. Apoi este selectat unul din vârfurile marcate rămase și este identificată calea cea mai scurtă de la acest vârf la oricare din vârfurile arborelui parțial. Acest proces continuă până când se procesează toate vârfurile marcate. O descriere formală a acestei euristici este prezentată în Figura 5.10.

```
Algorithm Arbore_Steiner;
begin
        M ← set de noduri marcate /* noduri în care conexiunea are pini */;
        s ← un nod selectat din M;
        Se aplică algoritmul Dijkstra pentru a determina π(s,e), calea cea mai scurtă
                de la s la un anumit nod e din M;
        M ← M - {e};
        V ← {s, e}; /* noduri ale arborelui Steiner */
        while (M ≠ φ) do
                e ← next (M); /* se alege un alt nod din M */
                Se aplică algoritmul Dijkstra pentru a determina π(e,x), calea cea mai scurtă
                        de la e la un anumit nod x din V;
                V (π(e,x)) ← noduri acoperite de π(e,x);
                V ← V ∪ V (π(e,x));
                /* se elimină nodurile marcate acoperite de calea π(e,x) */
                M ← M - M ∩ V (π(e,x));
        endwhile
end.
```

Figura 5.10. Exemplu de euristică pentru determinarea arborelui Steiner.

O altă euristică pentru determinarea unui arbore Steiner sub-optimal se bazează pe o variație a unui algoritm pentru determinarea arborelui de acoperire minim al lui Kruskal. Fie *M* setul de noduri (vârfuri) în care conexiunea are pini. Se determină mai întâi căile cele mai scurte între toate perechile de noduri din *M*. Există $\binom{n}{2}$ asemenea căi, unde *n* este egal cu numărul de noduri din *M*. Aceste căi sunt sortate în ordinea crescătoare a lungimii lor, și sunt procesate una câte una. Calea cea mai scurtă identifică prima ramură a arborelui Steiner. Fiecare din căile următoare identifică o altă ramură a arborelui. Algoritmul se termină atunci când toate nodurile au fost acoperite și conectate. Este probabil ca acest lucru să se întâmple înaintea procesării tuturor căilor. O descriere formală a acestei euristici este prezentată în Figura 5.11.

În cazul în care graful este un graf de caroiaj, se poate utiliza algoritmul Lee modificat pentru conexiuni multipin în scopul rutării conexiunilor între celulele globale. Acest algoritm va fi descris în secțiunea următoare.

```
Algorithm Aproximare_Steiner;
begin
        M ← set de noduri marcate /* noduri în care conexiunea are pini */;
        Se determină căile cele mai scurte între toate perechile de noduri din M;
        P ← secvența căilor sortate în ordinea crescătoare a lungimii lor;
        V ← φ;
        E ← φ;
        while (V ≠ M) do
                cale ← next (P);  /* se preia următoarea cale cea mai scurtă */
                                  /* și se elimină din P */
                forall (i, j) ∈ E (cale) do
                        if ( (i, j) nu crează un ciclu în graful G (V, E) ) then
                                V ← V ∪ {i, j};
                                E ← E ∪ {(i, j)};
                        endif
                endfor
        endwhile
end.
```

Figura 5.11. Euristică de aproximare a arborelui Steiner utilizând o variație a algoritmului pentru arborele de acoperire minim.

5.5.2.2 Rutarea globală prin metoda parcurgerii labirintului

Primul pas al rutării globale este modelarea regiunilor de rutare. Zonele orizontale și verticale de rutare sunt definite prin extinderea muchiilor orizontale și verticale ale celulelor plasate până la marginea suprafeței. Regiunile de rutare ale acestui model reprezintă intersecțiile dintre zonele orizontale și verticale de rutare. În acest caz, nu se garantează ca regiunile de rutare să fie canale.

După identificarea canalelor (regiunilor) de rutare, etapa următoare este asignarea de conexiuni acestor regiuni. Pentru aceasta, canalele sunt modelate printr-un *graf de conectivitate al canalelor*. Pentru rutarea cu două straturi, fiecărui nod i se asignează două ponderi reprezentând capacitatea orizontală (lățimea) și capacitatea verticală (lungimea) canalului corespunzător (Figura 5.12).

Metoda secvențială este cea mai simplă și cea mai des utilizată metodă pentru rutarea globală. Această metodă constă în selectarea câte unei conexiuni la un moment dat și determinarea unui arbore Steiner optimal (sau sub-optimal) care acoperă toți pinii conexiunii. Există două posibilități în acest caz: (1) metoda dependentă de ordine și (2) metoda independentă de ordine.

În cazul rutării globale *independente de ordine*, fiecare conexiune este rutată independent de toate celelalte conexiuni. Se identifică apoi zonele congestionate și conexiunile afectate sunt rerutate, penalizând căile care trec prin aceste zone. Această metodă nu necesită ordonarea conexiunilor și reduce în mod considerabil complexitatea spațiului de căutare, deoarece singurele obstacole sunt celulele. Totuși, poate fi necesar un număr mare de iterații pentru găsirea unei soluții fezabile a problemei de rutare globală.

În cazul rutării globale *dependente de ordine*, mai întâi se ordonează conexiunile conform anumitor criterii. Apoi conexiunile sunt rutate în ordinea rezultată, actualizând spațiul de rutare disponibil după fiecare conexiune. Căutarea este mai complexă, deoarece numărul de obstacole este mai mare (celule și conexiuni deja rutate). Ambele metode sunt oarecum similare prin faptul că ele încearcă identificarea unui arbore Steiner pentru o conexiune la un moment dat. În continuare se va descrie metoda dependentă de ordine.

Fiecare margine a unui bloc este atašatã unui canal unic. Deci, fiecare pin este asociat în mod unic cu un canal. Astfel, nodurile grafului de conectivitate al canalelor în care o conexiune are pini sunt determinate în mod neambiguu. Rutarea globalã a unei conexiuni se reduce deci la determinarea unui arbore care acoperã toate nodurile în care conexiunea are pini.

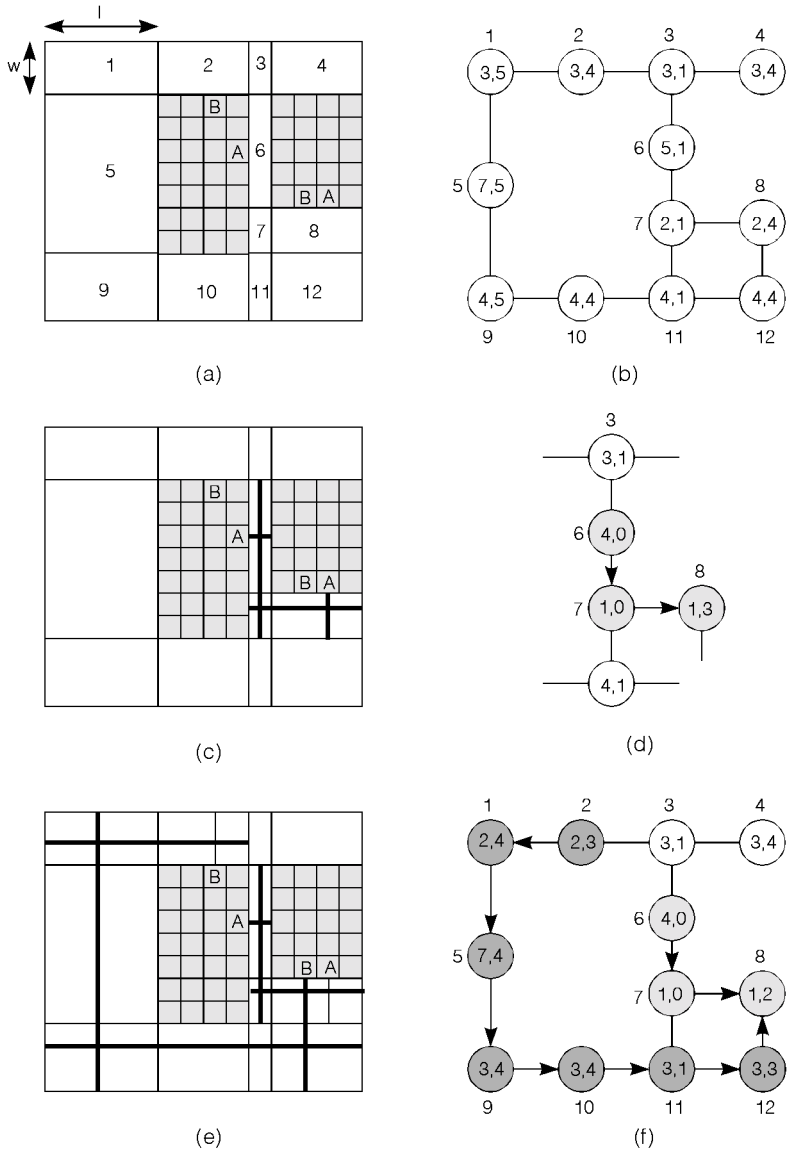

Figura 5.12. Rutarea globalã. (a) Modelul canalului. (b) Graful de conectivitate al canalelor. (c) Rutarea globalã *A–A'*. (d) Graful canalelor cu *A–A'* rutat. (e) Rutarea globalã *B–B'*. (d) Graful canalelor cu *A–A'* øi *B–B'* rutate.

Se considerã cazul simplu în care conexiunea are pini numai în douã noduri. În acest caz, asignarea unei conexiuni canalelor este realizatã prin cãutarea unei cãi în graful canalelor. Este cãutatã o cale între canalul care conøine nodul sursã la cel care conøine nodul destinaøie. Procedura de cãutare este similarã cu cea utilizatã în algoritmul Lee. Pentru simplitate se presupune cã lungimea tuturor muchiilor în graful de conectivitate al canalelor este o unitate. Pornind de la un nod etichetat cu k, nodurile adiacente sunt etichetate cu $k + 1$. Procedura de etichetare continuã pânã când se ajunge la nodul destinaøie. Calea cea mai scurtã în graful de conectivitate al canalelor este indicatã printr-o secvenøã de noduri cu etichete descrescãtoare. Dupã ce este gãsitã aceastã cale, conexiunea este asignatã canalelor, øi pentru toate nodurile (canalele)

din cale ponderile capacitãøilor *w* şi *l* sunt micşorate conform cu lãøimea şi lungimea conexiunii care trebuie rutatã.

Figura 5.12(a) indicã o plasare a celulelor, fiind necesarã rutarea a douã conexiuni, *A* - *A*' şi *B* - *B*'. Figura 5.12(b) prezintã graful corespunzãtor al canalelor. Pinul *A* este adiacent cu canalul numãrul 6, iar pinul *A'* cu canalul 8. Calea cea mai scurtã din graf de la nodul 6 la nodul 8 (6-7-8) este indicatã în Figura 5.12(d).

Se observã cã ponderile nodurilor în aceastã parte a grafului din Figura 5.12(d) sunt actualizate. Lãøimea şi lungimea canalului sunt reduse cu o valoare egalã cu spaøiul utilizat de segmentul de interconectare. În cazul în care calea trece doar vertical (orizontal), este redusã numai ponderea corespunzãtoare lungimii (lãøimii) sale. Aceasta înseamnã cã o întreagã linie şi o întreagã coloanã a fiecãrui canal (6, 7 şi 8) sunt asignate acestei conexiuni, dupã cum se indicã în Figura 5.12(c).

Urmãtorul pas este asignarea unor canale pentru conexiunea *B* - *B'*. Pinul *B* este adiacent cu canalul 2, iar pinul *B'* este adiacent cu canalul 8. Se considerã douã cãi posibile care interconectează nodul 2 cu nodul 8. Prima este calea 2-3-6-7-8, iar a doua este 2-1-5-9-10-11-12-8. Canalele 6 şi 7 au o lãøime de o unitate şi au fost asignate conexiunii *A* - *A'*, deci capacitãøile lor orizontale au devenit zero. Calea cea mai scurtã disponibilã este deci 2-1-5-9-10-11-12-8. Ponderile actualizate din graf sunt indicate în Figura 5.12(f).

O altã aplicaøie a tehnicii prezentate anterior este de a determina separarea necesarã între celule în scopul asigurãrii rutabilitãøii circuitului de cãtre programul de rutare detaliatã. Rutarea globalã este utilizatã deci numai pentru a determina separarea necesarã între celule. Atunci când se executã rutarea detaliatã pentru întregul circuit, nu este necesarã urmãrirea exactã a canalelor asignate conexiunilor de programul de rutare globalã.

Modificarea metodei anterioare pentru determinarea separãrii între celule constã în considerarea unei separãri iniøiale egale cu zero, ceea ce va reprezenta ponderea iniøialã a nodurilor. În continuare, de fiecare datã când se gãseşte o cale, ponderile nodurilor (canalelor) corespunzãtoare sunt incrementate. La sfârşitul acestei proceduri, plasarea relativã a celulelor va fi menøinutã, dar separarea minimã între celule va fi cea datã de ponderile orizontale şi verticale ale nodurilor grafului de conectivitate al canalelor.

5.5.2.3 Rutarea globală utilizând arbori Steiner ponderați

Fie un set de conexiuni cu terminale multiple $\eta = \{N_1, \ldots, N_n\}$ pentru care trebuie executatã rutarea globalã. Presupunem cã suprafaøa de rutare este divizatã în regiuni dreptunghiulare (Figura 5.13). Fiecare conexiune *N* cu *k* terminale este specificatã printr-un *k*-tuplu $(R_1, \ldots, R_k)$, unde R_i, $1 \le i \le k$, sunt regiunile conøinând terminalele conexiunii *N*. Regiunile R_i nu sunt neapãrat distincte, deoarece o conexiune poate avea mai multe terminale într-o regiune. Pentru rutarea globalã, pentru fiecare conexiune trebuie specificatã o secvenøã de regiuni prin care va trece conexiunea respectivã.

Considerând o soluøie a rutãrii globale, fie $d(i, j)$ numãrul de conexiuni care intersectează marginea a douã regiuni adiacente R_i şi R_j, iar $c(i, j)$ capacitatea marginii regiunilor R_i şi R_j (numãrul maxim de conexiuni care pot intersecta această margine). O soluøie a rutãrii globale este specificatã printr-un set η de conexiuni cu terminale multiple, o funcøie de capacitate *C* şi o colecøie de regiuni. O conexiune cu *k* terminale are *multiplicitatea k*. Multiplicitatea setului η este egalã cu numãrul maxim de terminale pe conexiune, dintre toate conexiunile setului η. Problema de rutare globalã pentru o anumitã pereche (η, C) constã în determinarea unei secvenøe de regiuni astfel încât $d(i, j) \le c(i, j)$ pentru fiecare douã regiuni adiacente R_i şi R_j. De notat cã deşi

obiectivul principal este satisfacerea restricøiilor de capacitate, existã un numãr de obiective secundare importante, ca de exemplu minimizarea lungimii conexiunilor.

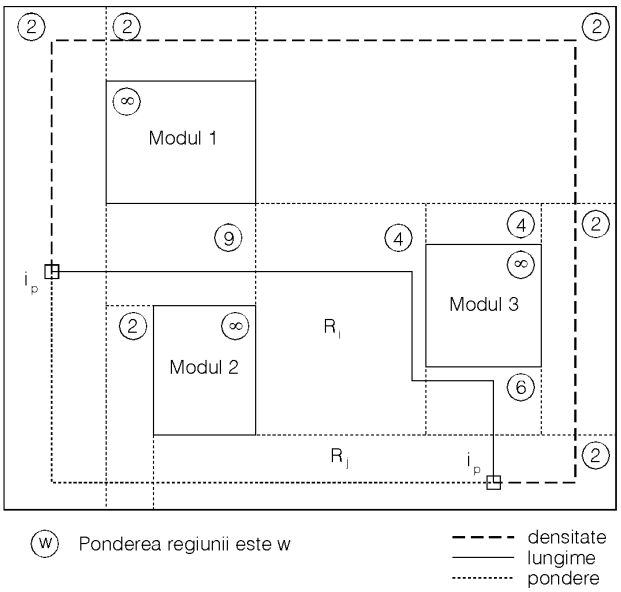

Figura 5.13. Trei tipuri de arbori Steiner.

Considerãm cã regiunilor din Figura 5.13 li se asociazã ponderi, ponderea unei regiuni depinzând de suprafaøa acesteia, numãrul de conexiuni care trec prin regiune, şi numãrul de terminale pe care le conøine. Ponderea regiunilor corespunzãtoare modulelor este setatã la ∞, pentru a indica faptul cã nici o conexiune nu poate trece prin acestea. Existã diferite tipuri de arbori Steiner care se pot defini pentru o anumitã conexiune. În Figura 5.13, considerând conexiunea între terminalele notate cu *p*, sunt definiøi trei arbori Steiner: pentru densitate, lungime şi pondere. Considerãm arborele Steiner pentru pondere, numit *arbore Steiner ponderat*. Un *arbore Steiner cu pondere minimã* este un arbore Steiner cu ponderea totalã minimã, unde o muchie a arborelui cu lungimea ℓ_i din regiunea R_i cu ponderea w_i are ponderea totalã $\ell_i w_i$.

Pentru conexiunea între terminalele notate cu *p* din Figura 5.13, arborele Steiner cu lungimea minimã nu este adecvat, deoarece trece prin regiuni critice, care conøin module. Dupã modificarea traseului, calea de rutare poate trece în continuare prin regiuni critice, de exemplu cea cu ponderea 9. De asemenea, prin aceastã modificare se poate viola optimalitatea lungimii. În aceste cazuri, un arbore Steiner cu pondere minimã reprezintã o conexiune eficientã, fiind posibilã minimizarea simultanã a lungimii conexiunilor şi a densitãøii regiunilor. O asemenea soluøie a fost propusã de Chiang *et al.* [54], care au descris şi o metodã de construire a unui arbore Steiner ponderat. În continuare se va descrie aceastã metodã.

Se considerã o divizare a planului într-o colecøie de regiuni $\mathcal{R} = \{R_1, \ldots, R_m\}$. Regiunii R_i îi este asignatã o pondere pozitivã w_i. Se considerã o cale $\mathcal{P}$ care conectează douã puncte p_a şi p_b, trecând prin diferite regiuni. Se notează cu ℓ_i lungimea cãii $\mathcal{P}$ în regiunea R_i, deci $\ell_i = |\mathcal{P} \cap R_i|$. Ponderea cãii $\mathcal{P}$ este $W(\mathcal{P}) = \Sigma \ell_i w_i$.

Fiind dat un set de puncte P din $\mathcal{R}$, problema este de a obøine un arbore Steiner cu pondere minimã care interconectează punctele din P. Se considerã numai cãi rectiliniare. Aceasta este problema *arborelui Steiner rectilinar ponderat* (*weighted rectilinear Steiner tree – WRST*). Problema *WRST* este NP-completã, deoarece o clasã restrânsã a acestei probleme, şi anume cea în care $w_i = 1$ pentru fiecare i, este problema

tradiøionalã a arborelui Steiner rectiliniar cu lungime minimã, care este NP-completã [146].

Se aratã mai întâi modul în care se poate gãsi o cale cu pondere minimã între douã puncte p_a øi p_b din R. Pentru aceasta se construieşte un graf *G*, numit *graful de cãutare* al R. Acest graf se obøine prin extinderea marginilor fiecãrei regiuni pânã când ele ajung la marginea exterioarã sau la un obstacol (o regiune cu pondere infinitã). Noua configuraøie este notatã cu D. Intersecøiile segmentelor de linie din D definesc vârfurile grafului *G*, iar liniile care le interconectează definesc muchiile grafului.

Fiind dat un set de puncte *Q*, se poate obøine un graf de cãutare G_Q în (R, *Q*) dupã cum urmează. Se începe cu D, cãruia i se adaugã setul de puncte *Q*. Se extind apoi segmentele de linii orizontale şi verticale de la fiecare punct din *Q* pânã când fiecare segment ajunge la un obstacol sau la margine. Noua configuraøie este notatã D_Q. Intersecøiile segmentelor de linie din D_Q definesc vârfurile grafului G_Q, iar segmentele de linie care le interconectează definesc muchiile grafului G_Q.

Pentru gãsirea unei cãi de pondere minimã între douã puncte p_a şi p_b se poate utiliza urmãtoarea Lemã [54].

> **Lema 5.1.** Existã o cale de pondere minimã între douã puncte p_a şi p_b, care este o cale în graful $G_{\{p_a, p_b\}}$.

P fiind un set de puncte, graful G_P are *e* muchii. Pentru gãsirea unei cãi de pondere minimã între douã puncte p_a şi p_b din *P*, este suficient sã se determine o cale cu pondere minimã în graful *G*, într-un timp *O* (*e* log *e*).

Un *arbore de acoperire minim* (ponderat) (*minimum spanning tree – MST*) T_P al unui set de puncte *P* este un arbore de pondere minimã care interconectează toate punctele. Se poate construi un *MST* al setului *P* prin determinarea unui *MST* pentru punctele grafului G_P, deoarece G_P conøine toate cãile minime între aceste puncte.

Relaøia dintre ponderea unui arbore de acoperire minim şi ponderea unui arbore Steiner optim este datã de urmãtoarea Lemã [54].

> **Lema 5.2.** Raportul dintre ponderea unui arbore de acoperire minim şi ponderea unui arbore Steiner optim satisface urmãtoarea inegalitate:
>
> $$\frac{W_{MST}}{W^*_{WRST}} \le 2\left(1-\frac{1}{l}\right) \tag{5.1}$$
>
> unde *l* este numãrul minim de frunze din *WRST*.

Lema 5.2 sugerează faptul cã orice algoritm care porneşte de la un *MST* şi îl modificã, fãrã a creşte ponderea acestuia, asigurã obøinerea limitei ponderii definite în lemã. Au fost propuşi diferiøi algoritmi care utilizează această procedurã. Chiang *et al.* [54] au propus un algoritm mai eficient, care va fi descris în continuare.

Pentru construirea unui *WRST* se porneşte de la un *MST* ponderat T_P al setului de puncte *P*. Pe baza Lemei 5.1, T_P este un subgraf al grafului de cãutare G_P. Dacã ponderea *WRST* rezultat este mai micã decât ponderea *MST* original, limita definitã de Lema 5.2 este respectatã.

Muchiile $e_1, \ldots, e_{t-1}$, $t = |P|$, ale T_P vor fi procesate una câte una. Ordinea de procesare a muchiilor influenøează rezultatul final. Muchiile sunt procesate în ordine crescãtoare a ponderilor, deci ponderea muchiei e_1 este cea mai micã. Se consideră procesarea unei muchii e_j din T_P. Se presupune cã punctele de pe muchiile $e_1, \ldots, e_{j-1}$ au fost interconectate, deci s-a obøinut o colecøie de arbori Steiner ponderaøi. Reuni-

unea acestor interconectãri se notează cu L_{j-1}, ponderea acestei reuniuni respectând inegalitatea:

$$W(L_{j-1}) \le \sum_{i=1}^{j-1} W(e_i) \tag{5.2}$$

În general, există mai multe căi cu pondere minimă care conectează două puncte. Fie $CALE_1(e_j)$, …, $CALE_q(e_j)$ un subset al acestora, unde q este un parametru de proiectare. Fiecare din căile $CALE_i(e_j)$ este unită cu L_{j-1} în felul următor. Fie h_1 şi h_2 capetele muchiei e_j. Dacă h_1 şi h_2 sunt conectate deja în L_{j-1}, procesarea muchiei e_j se opreşte şi se continuă cu procesarea muchiei e_{j+1}. În caz contrar, se începe de la h_1 şi se conectează h_1 cu h_2 utilizând $CALE_i(e_j)$. Există două cazuri:

- *Cazul 1:* $CALE_i(e_j)$ intersectează numai h_1 şi h_2. În acest caz nu sunt necesare alte operaøii.
- *Cazul 2:* $CALE_i(e_j)$ intersectează L_{j-1}, începând cu h_1, în punctele $h_1 = z_1, z_2, \ldots, z_{f-1}, h_2 = z_f$, unde $f > 2$. Se testează subcalea din $CALE_i(e_j)$ de la z_v la z_{v+1}, pentru fiecare v, $1 \le v \le f$-1, în această ordine, începând cu $v = 1$. Dacă subcalea de la z_v la z_{v+1} creează un ciclu, se elimină întreaga subcale. În caz contrar, ea rămâne neschimbată.

Se repetă procesul începând de la h_2. Cele două rezultate sunt comparate şi este selectat rezultatul cel mai bun. L_j este deci actualizat pentru fiecare din cele q căi. Procedura descrisă mai sus este numită *MERGE* (L_{j-1}, $CALE_i(e_j)$). Rezultatul L_j este o colecøie de arbori (fără cicluri). Diferenøa dintre ponderile L_j şi L_{j-1} este mai mică sau egală cu ponderea muchiei e_j, deci $W(L_j) - W(L_{j-1}) \le W(e_j)$. Astfel, L_{j-1} este un arbore Steiner rectiliniar ponderat cu ponderea totală mai mică sau egală cu ponderea lui T_P.

O descriere formală a algoritmului este prezentată în Figura 5.14.

```
Procedure WRST (R, P);
begin
        T_P ← un arbore de acoperire minim al setului P;
        for (j = 1 to t - 1) do              /* t este numărul de terminale */
                for (i = 1 to q) do          /* q este numărul de căi */
                        L_i = MERGE (L_{j-1}, CALE_i(e_i));
                endfor;
        endfor;
        return (min_i L_i);
end
```

Figura 5.14. Algoritmul pentru construirea arborelui Steiner rectiliniar ponderat.

În continuare se descrie algoritmul de rutare globală care utilizează arbori Steiner ponderaøi. Primul pas al algoritmului constă în asignarea unui număr distinct, numit *număr de ordine*, fiecărei conexiuni. În general, numărul de ordine al unei conexiuni depinde de prioritatea, ponderea şi multiplicitatea conexiunii. De exemplu, conexiunilor de alimentare li se poate asigna prioritatea 1, celor de ceas li se poate asigna prioritatea 2, iar conexiunilor critice li se poate asigna prioritatea 3. Ponderea unei conexiuni este proporøională cu ponderea arborelui de acoperire minim (ponderat) al acesteia. Multiplicitatea conexiunii este proporøională cu numărul de terminale.

În algoritmul descris, ordonarea conexiunilor este realizată numai pe baza ponderii. Fie W_i ponderea *MST* pentru conexiunea N_i şi o funcøie de ordonare Π pe setul de ponderi W_i. Conexiunile sunt ordonate conform funcøiei Π, deci conexiunea

cu ponderea minimă este rutată prima. Motivul acestei ordonări este că în primele etape ale rutării se dorește evitarea regiunilor cu densitate mare, fiind acceptabile conexiunile mai lungi produse ca rezultat. Însă, în etapele finale ale algoritmului trebuie construiți arbori de lungime minimă, pentru a se evita saturarea capacității regiunilor.

Înaintea începerii rutării, ponderea unei regiuni depinde numai de capacitatea acesteia și de numărul de terminale pe care le conține. Ponderea unei regiuni s-a ales egală cu numărul de terminale din regiune împărțit cu suma capacităților marginilor regiunii. Pe măsura rutării conexiunilor, capacitatea unor regiuni crește, și deci ponderea acestora va crește de asemenea.

Al doilea pas al algoritmului este obținerea, pentru fiecare conexiune, a unui arbore Steiner ponderat, a cărei pondere totală este minimizată. Fie N_i conexiunea curentă care se procesează. Considerăm perechea ($\mathbb{R}$, W_i), unde $\mathbb{R}$ este setul de regiuni și W_i este ponderea regiunilor înaintea procesării conexiunii N_i. Ponderea unei regiuni crește pe măsură ce prin această regiune trec mai multe conexiuni. Un arbore Steiner ponderat pentru ($\mathbb{R}$, W_i) impune o rutare globală care minimizează simultan densitatea conexiunilor în cadrul regiunilor și lungimea conexiunii curente.

Pentru a controla importanța ponderii și a lungimii pe parcursul algoritmului, se introduc constantele λ^j. La iterația j, se obține un *WRST* pentru conexiunea N_i care minimizează valoarea $\sum_i \lambda_i^j w_i^j \ell_i^j$, unde w_i^j este ponderea regiunii R_i prin care trece N_i, iar ℓ_i^j este lungimea conexiunii N_i în R_i. Valoarea λ_i^j este selectată astfel încât $\lambda_i^j w_i^j$ scade pe măsură ce j crește, devenind apropiată de 1, astfel încât în ultimele iterații ale algoritmului să crească rolul lungimii conexiunii relativ la cel al ponderii. Valoarea utilizată este $\lambda_i^j = (1/j + 1/w_i^j)$, iar numărul total de iterații (valoarea maximă pentru j) este 20.

La iterația j al algoritmului, dacă ponderea totală a rutării conexiunii N_i este mai mică decât ponderea totală a rutării conexiunii N_i de la iterația j - 1, rutarea este acceptată; în caz contrar, ea este respinsă. O descriere formală a algoritmului este prezentată în Figura 5.15.

```
Algorithm Rutare_WRST (R, η, s);
        /* R este setul regiunilor de rutare                  */
        /* η este setul conexiunilor                          */
        /* s este numărul de conexiuni din setul η            */
begin
        W ← funcția ponderilor inițiale ale R ;
        for (i = 1) to s do
                W (N_i) = ∞;            /* se inițializează ponderile conexiunilor */
        endfor;
        for (j = 1) to 20 do
                N_i = conexiunea curentă;  /* conform funcției de ordonare Π */
                temp = WRST (R, N_i);
                if (temp) ≤ W (N_i) then
                        N_i = temp;        /* se acceptă rutarea lui N_i */
                        actualizare W;
                endif;
        endfor;
end.
```

Figura 5.15. Algoritmul de rutare globală pe baza arborilor Steiner ponderați.

Timpul de procesare pentru fiecare conexiune este $O(t^2 + e \log e)$, unde t este numărul de terminale ale conexiunii, iar e este numărul de muchii din graful de căutare. Algoritmul de rutarea globală descris este mult mai rapid față de alți algoritmi care utilizează metoda călirii simulate (de exemplu, TimberWolf) sau algoritmi care utilizează rețele de flux [54].

5.5.2.4 Rutarea globală orientată pe performanțe

Odată cu progresele înregistrate în tehnologia de fabricație a circuitelor VLSI, întârzierile datorate interconexiunilor au devenit semnificative pentru viteza circuitelor. De aceea, interconexiunile din cadrul unui circuit integrat și dintre circuite au un rol major în determinarea performanțelor sistemelor digitale. Aceasta a determinat apariția unor sisteme de proiectare în care accentul se pune pe maximizarea performanțelor. Deși majoritatea cercetărilor în acest domeniu au ca temă problema de plasare, există unele lucrări și în domeniul rutării globale.

Jackson, Kuh și Marek-Sadowska au raportat o metodă ierarhică pentru rutarea controlată de restricțiile de timp [60]. O altă metodă de rutare globală controlată de cerințele de reducere a întârzierilor a fost raportată de Prasitjutrakul și Kubitz, în care rutarea globală este integratã cu un analizor de timp [146]. Rutarea globală elaboratã de acești autori pentru circuite cu macro-celule se bazează pe algoritmul euristic de căutare *A**. Metodele amintite nu sunt însă suficient de flexibile pentru a considera simultan întârzierile de interconectare și costurile de rutare.

Cong *et al.* [60] au descris un algoritm de rutare globală orientat pe performanțe pentru celule standard și macro-celule. Această metodă este bazată arbori de rutare cu razã limitatã, pentru construcția acestora utilizându-se euristici bazate pe algoritmul lui Prim pentru arbori de acoperire minimi. Metoda permite proiectantului un control al importanței relative a costului rutării și a întârzierilor de interconectare. Aceeași autori au elaborat o metodă care minimizează simultan lungimea totală de interconectare (indicată de costul arborelui) și întârzierea maximă (indicată de raza arborelui).

Fiind dat un *graf de rutare* $G = (V, E)$, care este un graf ponderat conectat, o conexiune N este un subset al nodurilor acestui graf. O soluție de rutare a unei conexiuni N este un arbore din G, numit *arbore de rutare*, care conectează toate terminalele/nodurile din N.

Un arbore de rutare poate fi considerat ca un arbore *RC* distribuit. Pentru aproximarea întârzierilor de interconectare au fost utilizate diferite modele, ca de exemplu întârzierea Elmore sau limitele superioare și inferioare ale întârzierilor într-un arbore *RC*. Deși aceste modele sunt utile pentru simulare și verificarea întârzierilor, ele implică sume de termeni cuadratici, care sunt dificil de calculat și optimizat în procesul de proiectare fizică. De aceea, pentru o aproximare mai simplă a întârzierilor de interconectare se utilizează de obicei un model *RC* liniar, în care întârzierea de interconectare dintre un terminal sursă și unul destinație este proporțională cu lungimea interconexiunii dintre cele două terminale.

Costul unei căi din graful de rutare G este suma ponderilor muchiilor din acea cale. *Calea cea mai scurtă* în G dintre două terminale $x, y \in N$, $cale\_min_G(x, y)$, este o cale care conectează x și y, cu un cost minim. Într-un arbore de rutare T, calea cea mai scurtă dintre x și y, $cale\_min_T(x, y)$, este singura cale dintre terminalele x și y. În cazul geometric, costul căii celei mai scurte este distanța geometrică, notată cu $dist(x, y)$. Pentru un graf ponderat, acest cost este notat cu $dist_G(x, y)$.

Raza R a unei conexiuni este costul căii celei mai scurte din graful G de la o sursă la destinația cea mai îndepărtată, deci $\max_{x \in N} dist_G(s, x)$. *Raza* unui arbore de rutare T, notată cu $r(T)$, este costul căii celei mai scurte din T de la sursa s la destinația cea mai îndepărtată. Pentru oricare arbore de rutare, $r(T) \geq R$.

Dacă se utilizează modelul de întârziere *RC* liniar, întârzierile de interconectare se minimizează prin minimizarea razei arborelui de rutare, care măsoară întârzierea maximă de interconectare între sursă şi oricare destinaţie. Pentru aceasta se poate utiliza *arborele căii celei mai scurte* (*Shortest Path Tree - SPT*) al unei conexiuni, obţinut prin uniunea căilor celor mai scurte de la o sursă la toate destinaţiile, determinate prin algoritmul lui Dijkstra. Deşi *SPT* are raza cea mai mică posibilă *r* (*SPT*) dintre toţi arborii de rutare, costul *SPT* poate fi foarte ridicat. Pentru a se lua în considerare atât întârzierile de interconectare, cât şi costul rutării, problema de rutare globală controlată de restricţii de timp poate fi formulată după cum urmează.

Problema arborelui de rutare minim cu rază limitată: Fiind dat un parametru $\varepsilon \geq 0$ şi o conexiune cu raza *R*, să se determine un arbore de rutare *T* cu cost minim şi cu raza $r(T) \leq (1 + \varepsilon) \cdot R$.

Parametrul ε controlează importanţa relativă a razei şi a costului arborelui. Dacă $\varepsilon = 0$, se minimizează raza arborelui de rutare şi deci se obţine arborele căii celei mai scurte pentru conexiunea respectivă. Pe de altă parte, dacă $\varepsilon = \infty$, se minimizează costul total al arborelui şi se obţine un arbore de acoperire minim.

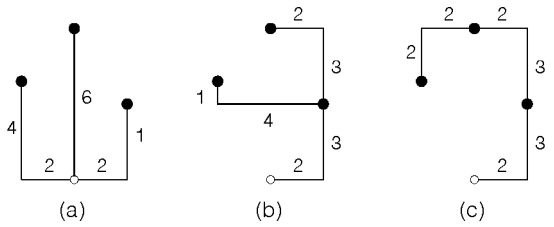

Figura 5.16. Un exemplu în planul Manhattan al modului în care creşterea valorii ε determină scăderea costului arborelui, dar creşterea razei $r(T)$: (a) $\varepsilon = 0$, cost(T) = 17, $r(T) = 6$; (b) $\varepsilon = 1$, cost(T) = 15, $r(T) = 10$; (c) $\varepsilon = \infty$, cost(T) = 14, $r(T) = 14$.

În Figura 5.16 se prezintă un exemplu în care se obţin trei arbori de acoperire distincţi pentru valori diferite ale parametrului ε. Figura 5.16(a) arată arborele de acoperire cu raza minimă corespunzător cazului $\varepsilon = 0$, cu lungimea căii maxime $r(T) = 6$ şi costul *cost* (T) = 17. Figura 5.16(b) arată o soluţie corespunzătoare cazului $\varepsilon = 1$, cu $r(T) = 10$ şi *cost* (T) = 15. Figura 5.16(c) arată arborele de acoperire minim corespunzător cazului $\varepsilon = \infty$, cu $r(T) = 14$ şi *cost* (T) = 14.

Pentru circuite cu celule standard sau macro-celule, distanţele dintre noduri reprezintă distanţe geometrice, graful de rutarea fiind $G = (V, E)$ cu $V = N$. În acest caz, numeroase metode de rutare globală sunt bazate pe construirea unui arbore de acoperire pentru fiecare conexiune. Astfel, problema arborelui de rutare minim cu rază limitată devine problema *arborelui de acoperire minim cu rază limitată* (*bounded radius minimum spanning tree - BRMST*). Cong *et al.* [60] au descris un algoritm euristic pentru construirea unui arbore de acoperire cu rază limitată, pe baza algoritmului clasic al lui Prim pentru construirea unui arbore de acoperire minim. Acest algoritm este descris în continuare.

Algoritmul construieşte un arbore $T = (V', E')$ care conţine iniţial numai sursa *s*. La fiecare pas, se alege $x \in V'$ şi $y \in N - V'$ astfel încât $dist(x, y)$ este minimă. Dacă adăugarea muchiei (x, y) nu violează restricţia razei, deci $dist_T(s, x) + dist(x, y) \leq (1 + \varepsilon) \cdot R$, se include această muchie în *T*. În caz contrar, se parcurge calea inversă de la *x* la *s* pentru a găsi primul terminal *x'* astfel încât muchia (x', y) să fie corespunzătoare, deci $dist_T(s, x') + dist(x', y) \leq R$, şi se adaugă (x', y) la arborele *T*. În cazul cel mai defavorabil, parcurgerea inversă se va termina cu $x' = s$.

Acest algoritm este numit algoritmul Prim limitat (*PRIML*) şi este descris în Figura 5.17. Algoritmul are avantajul cã raza arborelui rezultat nu este mai mare decât raza arborelui de acoperire minim, ori de câte ori acesta din urmã este unic [60].

Algorithm *PRIML* (N, s, R, ε);
begin
  $T = (V', E') = (\{s\}, \phi)$;
  while ($|V'| < |N|$) **do**
    Se selectează două terminale $x \in V'$ şi $y \in N - V'$ care minimizează $dist\ (x, y)$;
    if ($dist_T\ (s, x) + dist\ (x, y) \le (1 + \varepsilon) \cdot R$) **then**
      $x' = x$
    else
      Se caută primul terminal x' pe calea din T de la x la s astfel încât
        $dist_T\ (s, x') + dist\ (x', y) \le R$;
    endif
    $V' = V' \cup \{x'\}$;
    $E' = E' \cup \{(x', y)\}$
  endwhile
end.

Figura 5.17. Algoritmul *PRIML* pentru construirea unui arbore de acoperire cu razã limitatã, T, pentru setul de terminale N, cu sursa $s \in N$ şi raza R, utilizând parametrul ε.

Construcøia arborelui cu razã limitatã poate fi aplicatã şi altor metode de generare a arborilor de acoperire, diferite de algoritmul lui Prim. Pot exista diferite variante, în funcøie de modul de selecøie a perechii de terminale x şi y. Anumite variante asigurã o creştere a performanøelor faøã de algoritmul *PRIML*, de exemplu:

- *H1* – Se determinã x şi y ca şi în cazul algoritmului *PRIML*, şi se selectează terminalul x' pe calea din T de la x la s care produce o muchie corespunzãtoare de lungime minimã (x', y); se adaugã (x', y) la T.
- *H2* – Se cautã un terminal $y \in N - V'$ care minimizează $dist\ (x, y)$ pentru orice $x \in V'$, şi se selectează terminalul $x' \in$ care produce o muchie corespunzãtoare de lungime minimã (x', y); se adaugã (x', y) la T.
- *H3* – Se cautã o pereche de terminale $x \in V'$ şi $y \in N - V'$ care produce o muchie corespunzãtoare de lungime minimã (x, y); se adaugã (x, y) la T.

Complexitatea algoritmului *PRIML*, ca şi a variantelor *H1* şi *H2*, este $O\ (n^2)$, iar a variantei *H3* este de $O\ (n^3)$. În cazurile cele mai defavorabile, acešti algoritmi pot avea însã un raport de performanøã mai redus decât cazul optim, raport pentru care nu se poate stabili o limitã sub forma unei constante. Cong *et al.* [60] au elaborat de asemenea un algoritm de rutare orientat pe performanøe pentru care o asemenea limitã poate fi definitã. Acest algoritm se bazează pe o combinaøie a construirii unui arbore de acoperire minim şi a unui arbore cu cale minimã.

Ideea principalã a algoritmului este de a se construi un subgraf Q care acoperã N, având atât un cost total redus, cât şi o razã redusã. Atunci arborele cãii celei mai scurte al subgrafului Q va avea de asemenea un cost redus şi o razã redusã, şi va corespunde unei soluøii eficiente de rutare. Algoritmul poate fi descris astfel:

- Se determinã arborele cãii celei mai scurte SPT_G (*shortest path tree*) al grafului de rutare G, şi se determinã arborele de acoperire minim MST_G al grafului G. Se iniøializează graful Q ca fiind egal cu MST_G.
- Fie L secvenøa de vârfuri corespunzãtoare unei traversãri în adâncime a arborelui MST_G, unde fiecare muchie a MST_G este traversatã de două ori. Costul total al muchiilor pentru această traversare este dublul costului arborelui MST_G.

- Se traversează L, actualizând de fiecare dată costul total S al muchiilor traversate. La fiecare nod L_i, se testează dacă S este mai mare decât $\varepsilon \cdot dist_G(s, L_i)$. Dacă da, se resetează S la 0 şi se adaugă $cale\_min_G(s, L_i)$ la Q. Se continuă traversarea L, repetând acest proces.
- Arborele de rutare final este SPT_Q, arborele căii minime al grafului Q.

O descriere formală a algoritmului este prezentată în Figura 5.18.

Algorithm *BRMST* (G, s, R, ε);
begin
  Se determină arborele de acoperire minim MST_G şi arborele căii celei mai scurte SPT_G;
  $Q = MST_G$;
  L = traversare în adâncime a arborelui MST_G;
  $S = 0$;
  for ($i = 1$) **to** $|L| - 1$ **do**
    $S = S + cost(L_i, L_{i+1})$;
    if ($S \geq \varepsilon \cdot dist_G(s, L_{i+1})$) **then**
      $Q = Q \cup cale\_min_G(s, L_{i+1})$;
      $S = 0$;
    endif;
  endfor;
  T = arborele căii celei mai scurte al grafului Q;
end.

Figura 5.18. Algoritmul *BRMST* pentru construirea unui arbore de acoperire cu rază limitată, T, pentru $G = (V, E)$, cu sursa $s \in V$ şi raza R, utilizând parametrul ε.

În [60] se arată că pentru orice ε fixat algoritmul generează un arbore de rutare cu raza şi costul total limitate simultan de un factor constant faţă de cazul optim, pe baza următoarelor două teoreme.

Teorema 5.1. Pentru orice graf ponderat G şi orice parametru ε, arborele de rutare T construit de algoritmul *BRMST* are o rază $r(T) \leq (1 + \varepsilon) \cdot R$.

Teorema 5.2. Pentru orice graf ponderat G şi orice parametru ε, arborele de rutare T construit de algoritmul *BRMST* are un cost $cost(T) \leq (1 + (2/\varepsilon)) \cdot cost(MST_G)$.

Metoda descrisă în algoritmul *BRMST* se poate generaliza pentru formularea problemei de rutare prin arbori Steiner, obţinându-se o limită a lungimii conexiunilor mai mare de $2 \cdot (1 + (2/\varepsilon))$ ori decât costul arborelui Steiner optim, menţinându-se de asemenea limita de $(1 + \varepsilon) \cdot R$ pentru rază.

În planul Manhattan, limita pentru lungimea conexiunilor se poate îmbunătăţi la $(3/2) \cdot (1 + (1/\varepsilon))$ faţă de cea optimă, iar în planul euclidian această limită se poate îmbunătăţi suplimentar la $(2/\sqrt{3}) \cdot (1 + (1/\varepsilon))$ faţă de cazul optim.

5.5.3 Rutarea globală prin metoda călirii simulate

Prima aplicare a metodei de călire simulată pentru rutarea globală a fost raportată de Vecchi şi Kirkpatrick, autorii formulând problema ca o programare liniară întreagă, unde capacităţile fiecărei muchii sunt egale cu unu [146]. Sunt considerate numai conexiuni cu două terminale. Funcţia de cost utilizată este suma pătratelor încărcărilor tuturor regiunilor de rutare.

În aceastã secøiune, se va descrie rutarea globalã utilizatã în pachetul de programe TimberWolf, elaborat de Sechen øi Sangiovanni-Vincentelli [151]. Acest pachet este destinat plasãrii øi rutãrii globale a circuitelor cu celule standard, cu douã straturi metalice.

După faza de plasare iniøialã, programul TimberWolf continuã cu rutarea globalã. Rutarea globalã este soluøionatã în douã etape. Obiectivul primei etape este asignarea unor conexiuni canalelor de rutare orizontale astfel încât sã se minimizeze densitãøile canalelor. La sfârøitul acestei etape, sunt identificate toate conexiunile care pot fi asignate unui canal adiacent. Scopul etapei a doua este de a încerca reducerea densitãøii canalelor prin modificarea asignãrii canalelor pentru conexiunile identificate anterior.

Dupã rutarea globalã, TimberWolf executã rafinarea plasării prin interschimbarea aleatoare a celulelor învecinate. Dupã fiecare interschimbare, se executã ambele etape ale rutării globale pentru rerutarea conexiunilor afectate de interschimbare. Metoda călirii simulate este utilizatã numai în a doua etapã a rutării globale, ca øi în faza de rafinare a plasării.

A. Prima etapã

În această etapã se utilizează o metodă secvenøială independentă de ordine. Conexiunile sunt luate în considerare una câte una øi sunt rutate.

Se descrie mai întâi terminologia utilizatã în cadrul algoritmului.

Un *grup de pini* reprezintă pinii unei celule date care sunt conectaøi intern. Pinii aceluiaøi grup sunt echivalenøi. Coordonata *x* a unui grup de pini *P* este egală cu media coordonatelor *x* ale pinilor constituenøi, deci:

$$x(P) = \frac{1}{|P|} \times \sum_{i \in P} x(i) \tag{5.1}$$

Acest concept este ilustrat în Figura 5.19.

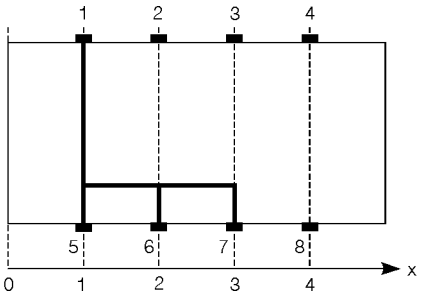

Figura 5.19. Ilustrarea conceptului grupului de pini: $P = \{1, 5, 6, 7\}$ øi $x(P) = \frac{1+1+2+3}{4} = \frac{7}{4}$.

O porøiune a unei conexiuni între două grupuri de pini, de exemplu P_1 øi P_2, se numeøte un *segment de conexiune*. Dacă P_1 øi P_2 aparøin la două celule diferite plasate pe acelaøi rând øi ambele au pini pe marginile de sus øi de jos ale celulelor, atunci segmentul de conexiune dintre cele două grupuri este numit un *segment comutabil*. Deci, un segment comutabil poate fi asignat canalului de rutare aflat deasupra sau dedesubtul rândului de celule. Decizia de asignare a fiecărui segment de conexiune unui anumit canal este bazată pe funcøia de cost.

Funcøia de cost utilizatã este egală cu suma densitãøilor totale ale canalelor, deci

$$D = \sum_{\forall c} d(c) \tag{5.2}$$

unde $d(c)$ este densitatea canalului c, care este egală cu numărul conexiunilor asignate canalului.

Algoritmul de rutare globală din prima etapă este compus din patru paşi distincţi care sunt executaţi pentru fiecare conexiune.

1. *Iniţializare*. Se identifică toate grupurile de pini ale conexiunii, şi se determină coordonatele x ale acestora. Grupurile de pini sunt sortate apoi după coordonatele x ale acestora, în ordine crescătoare.
2. *Construirea unui graf al grupurilor de pini*. Nodurile acestui graf modelează grupurile de pini, iar muchiile modelează conexiunile posibile (segmentele de conexiune) între grupurile de pini corespunzătoare.
3. *Construirea unui arbore de acoperire minim.* În acest pas, se utilizează algoritmul Kruskal pentru construirea unui arbore de acoperire minim al grafului grupurilor de pini.
4. *Identificarea segmentelor de conexiune*. În acest pas, sunt selectaţi pini individuali din cadrul grupurilor de pini pentru a forma segmentele de conexiune. Dacă segmentul de conexiune este comutabil, sunt selectate două perechi de pini, o pereche pentru rândul de sus, iar cealaltă pentru rândul de jos.

B. Etapa a doua

În această etapă, se utilizează tehnica de căutare a călirii simulate pentru rafinarea soluţiei rutării globale determinate prin algoritmul secvenţial din prima etapă. Sunt luate în considerare pentru rerutare numai segmente de conexiuni comutabile. Algoritmul metodei de călire simulată a fost descris în Capitolul 3. În această secţiune se va descrie funcţia de cost utilizată şi modul în care se generează soluţiile.

Obiectivul acestei etape este minimizarea densităţilor totale ale canalelor, conform ecuaţiei (5.2). Funcţia *generate* utilizată pentru obţinerea unor noi soluţii se execută astfel. Se selectează mai întâi un segment de conexiune comutabil, în mod aleator. Apoi, se comută asignarea canalului pentru segmentul selectat, de la canalul său curent la canalul opus. Dacă prin această comutare se reduce valoarea funcţiei de cost, comutarea este acceptată. Dacă noua soluţie are acelaşi cost ca şi cea precedentă, segmentul comutabil este asignat canalului cu densitatea cea mai mică. Scopul acestei decizii este de a se facilita rutarea detaliată.

5.5.4 Rutarea globală prin metoda programării întregi

Rutarea globală este formulată ca o problemă de programare întreagă 0-1. Suprafaţa de rutare este modelată ca un graf de caroiaj, unde fiecare nod reprezintă o celulă de caroiaj (super-celulă) [146]. Se presupune că marginea dintre două celule de caroiaj adiacente l şi k are o capacitate de $c_{l,k}$ piste. Aceasta corespunde unei ponderi pozitive $c_{l,k}$ a muchiei de legătură dintre nodurile l şi k din graful de caroiaj. Pentru fiecare conexiune i, trebuie identificate modurile diferite de rutare ale conexiunii. Se presupune că pentru fiecare conexiune i există n_i arbori posibili de rutare $t_1^i, t_2^i, \ldots, t_{n_i}^i$. Fiecărui arbore t_j^i îi asociem o variabilă $x_{i,j}$, cu semnificaţia următoare:

$$x_{i,j} = \begin{cases} 1 & \text{dacă conexiunea } i \text{ este rutată conform arborelui } t_j^i \\ 0 & \text{în caz contrar} \end{cases} \tag{5.3}$$

Fiecărei conexiuni i îi asociem o ecuaţie pentru a impune ca un singur arbore să fie selectat pentru acea conexiune:

$$\sum_{j=1}^{n_i} x_{i,j} = 1 \tag{5.4}$$

Astfel, pentru un graf de caroiaj cu M muchii şi T arbori, arborii de rutare ai tuturor conexiunilor se pot reprezenta printr-o matrice 0-1 $A_{M\times T} = [a_{i,p}]$, unde

$$a_{i,p} = \begin{cases} 1 & \text{dacă muchia } i \text{ aparţine arborelui } t_j^i \text{ şi lui } p, \text{ conform ecuaţiei (5.6)} \\ 0 & \text{în caz contrar;} \end{cases} \tag{5.5}$$

$$p = \sum_{m=1}^{l-1} n_m + k\,, \tag{5.6}$$

şi

$$T = \sum_{i=1}^{N} n_i \tag{5.7}$$

N fiind numărul de conexiuni.

Este necesar un al doilea set de ecuaøii pentru a se asigura cã pentru fiecare muchie i, $1 \le i \le M$, nu se depãşeşte capacitatea, deci

$$\sum_{k=1}^{N}\sum_{l=1}^{n_k} a_{i,p} \times x_{l,k} \le c_i \tag{5.8}$$

În sfârşit, dacã fiecãrui arbore t_j^i i se asociazã un cost $g_{i,j}$, o funcøie obiectiv posibilã care trebuie minimizatã este

$$F = \sum_{i=1}^{N}\sum_{j=1}^{n_i} g_{i,j} \times x_{i,j} \tag{5.9}$$

Astfel, o formulare posibilã a rutării globale ca o problemã de programare întreagã este:

$$\begin{cases} \min \sum_{i=1}^{N}\sum_{j=1}^{n_i} g_{i,j}x_{i,j} & \text{cu condiţiile:} \\ \sum_{j=1}^{n_i} x_{i,j} = 1 & 1 \le i \le N \\ \sum_{k=1}^{N}\sum_{l=1}^{n_k} a_{i,p}x_{l,k} \le c_i & 1 \le i \le M,\ p \text{ definit în (5.6)} \\ x_{k,j} = 0,1 & 1 \le k \le N, \text{ şi } 1 \le j \le n_k \end{cases} \tag{5.10}$$

Formularea rutării globale ca o problemã de programare întreagã este elegantã şi eliminã necesitatea ordonării conexiunilor. Metoda programării întregi poate gãsi o asignare optimã globalã a conexiunilor la regiunile de rutare. Aceastã metodã are însã mai multe dezavantaje:

1. Este necesarã identificarea mai multor arbori Steiner pentru fiecare conexiune, ceea ce poate necesita un timp ridicat. De asemenea, este dificilã enumerarea unui anumit numãr de arbori Steiner pentru fiecare conexiune.
2. Arborii trebuie selectaøi astfel încât sã se garanteze fezabilitatea problemei.
3. Existã un numãr mare de coeficienøi $a_{i,j}$, ceea ce conduce la un numãr mare de restricøii.
4. Poate rezulta un sistem cu un numãr foarte mare de ecuaøii.

Pentru rezolvarea unora din problemele amintite, metoda programării întregi a fost combinatã cu diferite metode ierarhice. În cazul acestora, problema de rutare

globalã este descompusã într-o ierarhie de subprobleme care sunt soluøionate individual. Atunci când metoda programării întregi se combinã cu o metodã de descompunere ierarhicã, se pot obøine timpi de execuøie acceptabili şi soluøii de calitate superioarã celor obøinute prin alte metode.

5.6 Rutarea detaliată

Rutarea detaliatã este etapa urmãtoare rutării globale. În aceastã etapã se asigneazã fiecãrei conexiuni piste particulare din cadrul regiunilor de rutare. Rutarea detaliatã se poate împãrøi în rutare detaliatã *generalã* şi rutare detaliatã *restricøionatã*. În cazul rutării generale se impun un numãr foarte redus de restricøii asupra problemei de rutare, şi este rutatã o singurã conexiune la un moment dat. Pe de altã parte, rutarea restricøionatã necesitã anumite constrângeri asupra problemei de rutare, ca de exemplu zone rectangulare libere cu toøi pinii aflaøi la periferie. Rutarea detaliatã restricøionatã se poate împãrøi la rândul ei în rutare prin *canale* şi rutare prin *blocuri de comutare*.

În secøiunea 5.6.1 se prezintã rutarea detaliatã generalã: metoda rutării prin labirint şi metoda cãutării liniilor. În secøiunea 5.6.2 se prezintã rutarea prin canale. Este descris algoritmul marginii din stânga şi algoritmii elaboraøi de Yoshimura şi Kuh. Se prezintã de asemenea alte metode de rutare prin canale.

5.6.1 Rutarea detaliată generală

5.6.1.1 Rutarea labirint

O clasã de metode de rutare cu scop general care utilizeazã un model de grilã este reprezentatã de rutarea labirint. Suprafaøa de rutare este împãrøitã în celule printr-o reøea de grile. Toate porturile şi conexiunile sunt aliniate pe aceastã grilã. Conectarea a douã puncte se realizeazã prin determinarea unei secvenøe de celule adiacente de la un punct la celãlalt. Se conectează o singurã pereche de puncte la un moment dat.

Unul din cei mai utilizaøi algoritmi pentru gãsirea unei cãi între douã puncte pe o grilã cu obstacole este algoritmul introdus de Lee. O caracteristicã a acestui algoritm este cã dacã existã o cale între douã puncte, aceasta va fi gãsitã, şi se garanteazã cã aceasta este calea cea mai scurtã. Existã trei faze în cadrul execuøiei algoritmului.

Prima fazã constã în etichetarea grilei, fiind numitã faza de umplere sau de propagare a undelor. Perechea de celule care trebuie conectate este etichetatã cu *S* şi *T*. În aceastã fazã, în timpul pasului *i*, celulele libere aflate la distanøa Manhattan *i* de la celula *S* sunt etichetate cu *i*. Etichetarea continuã pânã când celula destinaøie *T* este marcatã în pasul *L*, unde *L* este lungimea cãii celei mai scurte, iar fiecare celulã de pe calea respectivã contribuie cu o unitate de lungime. Procesul de etichetare continuã pânã când în pasul *i*:

1. Se ajunge la celula *T*; sau
2. Nu se ajunge la celula *T* şi în pasul *i* nu existã celule libere adiacente cu celulele etichetate *i* - 1; sau
3. Nu se ajunge la celula *T* şi *i* devine egal cu *M*, unde *M* este limita superioarã a lungimii unei cãi.

Dacã se ajunge la celula *T* în pasul *i*, atunci existã o cale de lungime *i* între punctele *S* şi *T*. Pe de altã parte, dacã în pasul *i* nu existã celule libere adiacente cu celulele etichetate cu *i* - 1, atunci calea cerutã nu este gãsitã. Se poate fixa o limitã su-

perioarã asupra lungimii unei cãi. Dacã $i = M$, unde M este aceastã limitã superioarã a lungimii cãii, şi nu se ajunge la celula T, atunci calea cu cerinøa $L \leq M$ nu este gãsitã.

Faza de umplere începe prin etichetarea cu 1 a tuturor celulelor libere adiacente cu celula sursã S. Apoi, se eticheteazã cu 2 toate celulele libere adiacente cu cele etichetate cu 1. Acest proces continuã şi se terminã când apare una din cele trei condiøii amintite. Procesul de umplere este ilustrat în Figura 5.20(a).

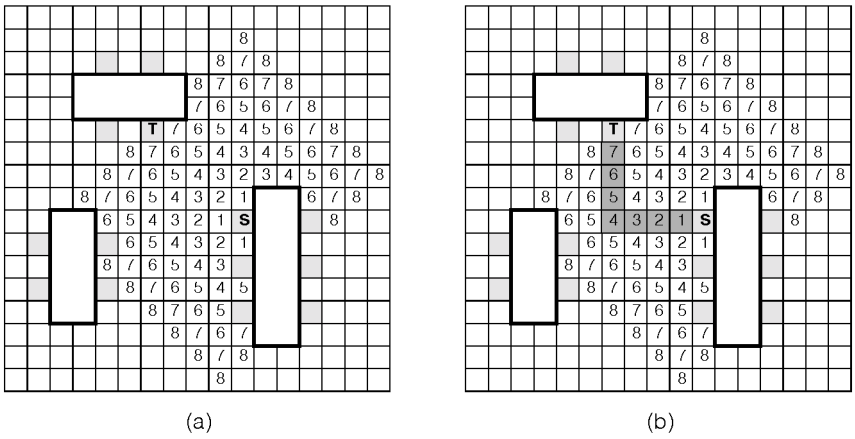

Figura 5.20. Algoritmul Lee. (a) Faza de umplere. (b) Faza de cãutare a sursei.

A doua fazã a algoritmului este numitã faza de cãutare a sursei. Aceastã procedurã este inversa procedurii de umplere. În aceastã fazã este gãsitã calea cea mai scurtã, dupã cum urmeazã. Dacã în pasul i s-a ajuns la celula T, va exista cel puøin o celulã adiacentã cu aceasta care conøine eticheta i - 1. De asemenea, o celulã cu eticheta i - 2 va fi adiacentã cu una etichetatã cu i - 1. În exemplul din Figura 5.20(a), deoarece la celula T s-a ajuns în pasul 8, va exista o celulã adiacentã etichetatã cu 7. De asemenea, va exista cel puøin o celulã cu eticheta 6 adiacentã cu 7. Prin trasarea celulelor numerotate în ordine descrescãtoare de la T la S, se poate gãsi calea cea mai scurtã doritã. Celulele haşurate din Figura 5.20(b) indicã aceastã cale pentru exemplul prezentat. În aceastã fazã, pot exista douã sau mai multe celule cu eticheta i - 1 adiacente cu celula cu eticheta i. În mod teoretic, oricare din acestea poate fi selectatã. În practicã, regula recomandatã este de a nu se schimba direcøia numai dacã aceastã schimbare este necesarã. Dupã gãsirea cãii dorite, celulele utilizate pentru conectarea celulelor S şi T sunt considerate ocupate pentru interconexiunile urmãtoare.

A treia fazã este numitã ştergerea etichetelor. În aceastã fazã etichetele tuturor celulelor, cu excepøia celor utilizate pentru calea gãsitã, sunt şterse pentru interconexiunile urmãtoare. Cãutarea celulelor etichetate se realizeazã în modul în care are loc propagarea undelor.

În algoritmul Lee, dacã L este lungimea cãii care trebuie gãsitã, timpul de procesare pentru faza de umplere este proporøional cu L^2, în timp ce timpul de procesare pentru faza de cãutare a sursei este proporøional cu L [146]. Astfel, complexitatea în timp a algoritmului este de $O(L^2)$ pentru fiecare cale. În plus, pentru un numãr de $N \times N$ celule, algoritmul necesitã o memorie de $O(N^2)$. De asemenea, este necesarã o memorie temporarã suplimentarã pentru memorarea poziøiei celulelor în faza de propagare a undelor.

Existã diferite extensii ale algoritmului Lee pentru reducerea cerinøelor de memorie şi a timpului de execuøie. Unele dintre acestea sunt prezentate în continuare.

Pentru reducerea cerinøelor de memorie, se observã în Figura 5.21 cã pentru fiecare celulã etichetatã cu k, toate celulele adiacente sunt etichetate fie cu k - 1, fie cu

k + 1. Astfel, în timpul fazei a doua este suficient dacã se pot distinge celulele predecesoare de celulele succesoare. Se utilizează diferite variante de etichetare bazate pe această idee. Două dintre ele sunt prezentate în continuare.

1. În prima variantă etichetarea începe, ca și la algoritmul obișnuit, cu celulele adiacente cu *S* etichetate cu 1, apoi cele adiacente cu 1 etichetate cu 2, și cele adiacente cu 2 etichetate cu 3. Apoi celulele adiacente cu 3 sunt etichetate din nou cu 1, și procesul continuă. Pentru etichetare se utilizează deci secvența 1,2,3, 1,2,3 …. Sunt necesari numai trei biți pentru fiecare celulă, deoarece există cinci stări în care se poate afla o celulă, și anume etichetată cu 1, 2 sau 3, liberă, sau blocată.
2. A doua variantă a fost propusă de Akers și constă din utilizarea secvenței 1,1, 2,2, 1,1, 2,2 …. Această variantă este cea mai economică, deoarece fiecare celulă se poate afla într-una din patru stări: etichetată cu 1, etichetată cu 2, liberă, sau blocată.

Timpul de execuție al algoritmului Lee este proporțional cu numărul de celule căutate în faza de umplere. Astfel, pentru a reduce numărul de celule care sunt etichetate, se pot utiliza următoarele tehnici.

1. *Selectarea punctului de început:* În cazul algoritmului Lee standard, oricare din cele două puncte care trebuie conectate se poate alege ca punct de început. Numărul de celule etichetate este mai redus dacă punctul ales ca punct de început este cel care se află cel mai departe de centrul grilei. Deoarece punctul de început este mai apropiat de marginea grilei, suprafața de propagare a undelor este mai redusă.
2. *Fan-out dublu:* În cazul acestei tehnici, în timpul fazei de umplere, undele sunt propagate atât de la celula sursă, cât și de la cea destinație. Etichetarea continuă până când se ajunge la un punct de contact între cele două unde. Cu această tehnică numărul de celule etichetate se reduce aproximativ la jumătate.
3. *Încadrare*: În această tehnică în jurul perechii de terminale care trebuie conectate se trasează o margine imaginară. Dacă nu se găsește nici o cale în interiorul zonei încadrate, marginea este extinsă și căutarea continuă. Această metodă crește viteza de căutare în mod considerabil.

Algoritmul Lee prezentat anterior conectează doi pini terminali utilizând calea cea mai scurtă de pe grilă. În cazul conexiunilor multipin, conectarea optimă a pinilor care aparțin aceleiași conexiuni se realizează cu ajutorul unui *arbore Steiner*. De multe ori obiectivul nu este de a găsi soluția minimă, ci o soluție care satisface anumite restricții ale lungimii conexiunilor. O soluție sub-optimă pentru cazul conexiunilor multipin poate fi obținută utilizând algoritmul Lee. Această metodă este prezentată în continuare.

În cazul algoritmului Lee clasic cele două puncte care trebuie interconectate au fost etichetate cu *S* (sursa) și *T* (destinația). Pentru cazul conexiunilor multipunct, unul din terminale este considerat sursă, iar restul ca și destinații. O undă este propagată de la sursă până când se ajunge la oricare din destinații. Apoi este găsită calea care conectează sursa cu această destinație. În continuare, toate celulele din calea găsită sunt etichetate ca celule sursă, iar terminalele rămase neconectate sunt etichetate ca destinații. Procesul este repetat până când se ajunge la unul din destinațiile rămase. Se determină apoi calea cea mai scurtă de la destinația găsită la una din surse. Procesul este continuat până când sunt conectate toate terminalele.

Nu se garantează că interconexiunea obținută prin acest proces este de lungime minimă. O interconexiune mai scurtă între două puncte poate fi determinată utilizând următoarea tehnică simplă. Prin eliminarea oricărei muchii dintr-un arbore se obțin doi sub-arbori. Calea cea mai scurtă între sub-arbori poate fi determinată aplicând din nou algoritmul Lee, toate celulele dintr-un sub-arbore având rol de celule

sursã, iar cele din al doilea sub-arbore având rol de celule destinaøie. În cazul în care calea cea mai scurtã obøinutã are lungime mai micã decât lungimea segmentului eliminat, prin inserarea acestei noi cãi se obøine o interconexiune de lungime mai redusã. Aplicând aceastã tehnicã pentru segmentul dintre sub-arborele *A–E* şi sub-arborele *B–C–D*, se obøine o interconexiune mai scurtã.

Rutarea labirint prezentatã este foarte generalã şi poate fi modificatã pentru rutarea circuitelor complexe cu obstacole. Aceastã tehnicã a fost utilizatã cu succes în douã programe de rutare, *Mighty* şi *Beaver*. Programul *Mighty* este bazat pe o strategie de rutare incrementalã, fiind elaborat de Shen şi Sangiovanni-Vincentelli. Programul are posibilitatea de a modifica conexiunile deja rutate. Funcøia de cost penalizeazã conexiunile lungi şi cele care necesitã orificii de trecere. Un alt program care utilizeazã rutarea labirint este *Beaver*, elaborat de Cohoon şi Patrick. Se utilizeazã o coadã de prioritãøi pentru a determina ordonarea conexiunilor. În plus, se utilizeazã controlul pistelor pentru a se preveni conflictele de rutare [146].

5.6.1.2 Rutarea prin metoda cãutãrii liniilor

Unul din principalele dezavantaje ale algoritmului Lee şi a variantelor sale este dimensiunea ridicatã a memoriei necesare pentru reprezentarea suprafeøei de rutare sub formã de grilã. Algoritmii de cãutare a liniilor eliminã acest dezavantaj.

Ideea principalã a acestor algoritmi este urmãtoarea. Presupunem douã puncte *S* şi *T* care trebuie conectate. Iniøial presupunem cã nu existã obstacole pe suprafaøa de rutare. Dacã se trasează o linie verticalã care trece prin *S* şi o linie orizontalã care trece prin *T*, cele douã linii se vor intersecta şi vor indica o cale Manhattan între *S* şi *T*. Dacã existã obstacole, trebuie efectuate operaøii suplimentare pentru a gãsi o cale între *S* şi *T*.

Spre deosebire de algoritmul Lee, care efectuează o cãutare în lãøime, algorimii de cãutare a liniilor efectuează o cãutare în adâncime. De aceea, acești algoritmi nu garantează gãsirea cãii celei mai scurte, şi pot necesita mai multe reveniri. În practicã algoritmii de cãutare a liniilor au o ratã de completare a rutãrii similarã cu algoritmul Lee, cu deosebirea cã atât necesarul de memorie, cât şi timpul de execuøie este considerabil mai redus. Aceasta deoarece spaøiul de rutare nu este memorat sub forma unei matrici, ci sub forma unor segmente de linii.

Primul algoritm de cãutare a liniilor a fost propus de Mikami şi Tabuchi. Acest algoritm este descris pe scurt în continuare. Fie *S* şi *T* o pereche de terminale ale unei conexiuni, terminale aflate la o intersecøie a unei grile imaginare. Primul pas constã în generarea a patru linii (douã orizontale şi douã verticale) care trec prin *S* şi *T*. Aceste linii sunt extinse pânã când întâlnesc anumite obstacole (de exemplu, o celulã plasatã) sau marginea suprafeøei. Dacã o linie generatã de la terminalul *S* intersectează o linie generatã de la terminalul *T*, atunci s-a gãsit o cale fãrã nici o schimbare de direcøie sau cu o schimbare de direcøie. Dacã cele patru linii generate nu se intersectează, atunci ele sunt identificate ca şi *linii de încercare* de nivel zero şi sunt pãstrate într-o memorie temporarã. În fiecare pas *i* se executã urmãtoarele operaøii.

1. Liniile de încercare de nivel *i* sunt selectate una câte una. De-a lungul fiecãrei linii de încercare se trasează toate punctele de intersecøie ale grilei (numite *puncte de bazã*). Pornind de la aceste puncte de bazã se generează noi linii de încercare perpendiculare pe linia de încercare *i*. Liniile generate sunt identificate ca şi linii de încercare de nivel *i* + 1.
2. Dacã o linie de încercare de nivel *i* + 1 intersectează o linie de încercare (de orice nivel) care porneşte de la celãlalt terminal, calea cerutã este gãsitã prin revenire de la punctul de intersecøie la ambele puncte *S* şi *T*. În caz contrar toate liniile de încercare de nivel *i* + 1 sunt memorate şi procedura se repetã de la pasul 1.

Algoritmul garantează găsirea unei căi dacă aceasta există, cu condiøia să se examineze toate liniile de încercare până la nivelul maxim de adâncime.

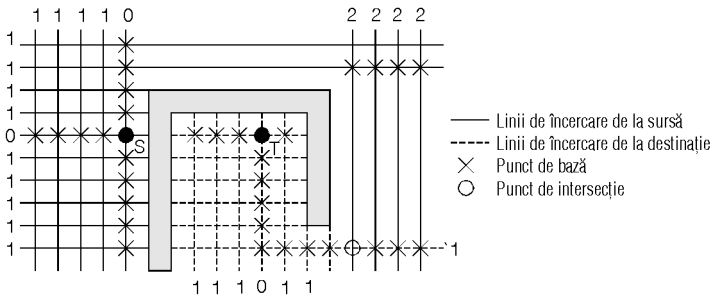

Figura 5.21. Rutarea prin algoritmul de căutare a liniilor Mikami-Tabuchi.

Figura 5.21 ilustrează un exemplu de determinare a unei căi prin aplicarea acestui algoritm. În acest exemplu linia de încercare de nivel 1 care pornește de la punctul *T* intersectează o linie de încercare de nivel 2 care pornește de la punctul *S*.

Un alt algoritm de căutare a liniilor a fost propus de Hightower. Acest algoritm este similar cu algoritmul Mikami-Tabuchi. Diferenøa constă în faptul că în loc să se genereze toate segmentele de linii perpendiculare pe o linie de încercare, algoritmul Hightower consideră numai acele linii care se pot extinde dincolo de obstacolul care a blocat liniile de încercare precedente [146].

Atunci când suprafaøa de rutare nu este congestionată, este de aşteptat ca timpul de execuøie al acestor algoritmi să fie redus. În particular, este de aşteptat ca timpul de execuøie al algoritmului Hightower să fie proporøional cu numărul schimbărilor de direcøie. O estimare o timpului de execuøie într-un labirint complex este de $O(N^4)$, unde N este numărul de celule ale grilei imaginare. Astfel, necesarul de memorie al algoritmilor de căutare a liniilor este redus considerabil, dar timpul de execuøie nu se îmbunătăøeşte prea mult. De asemenea, pot fi necesare şi reveniri (în urma aplicării unor secvenøe necorespunzătoare de linii de încercare).

5.6.2 Rutarea prin canale

Rutarea prin canale este un caz special al problemei de rutare în care interconexiunile sunt realizate într-o regiune dreptunghiulară fără obstrucøii. Acest tip de rutare se utilizează pentru reøele de porøi şi celule standard. Rutarea prin canale este eficientă şi simplă, garantând rutarea completă dacă dimensiunea canalelor este ajustabilă.

5.6.2.1 Definirea problemei de rutare prin canale

Un canal de rutare este definit printr-o regiune dreptunghiulară cu două rânduri de terminale de-a lungul marginilor de sus şi de jos. Fiecărui terminal i se asociază un număr între 0 şi N. Aceste numere reprezintă etichetele punctelor unei grile verticale, puncte aflate pe marginile de sus şi de jos ale regiunii dreptunghiulare, după cum se indică în Figura 5.22. Terminalele având aceeaşi etichetă i ($1 \le i \le N$) trebuie conectate prin interconexiunea i. Zerourile indică faptul că punctul corespunzător nu trebuie conectat. Lista de conexiuni este reprezentată de obicei prin doi vectori TOP şi BOT. TOP(k) şi BOT(k) reprezintă punctele grilei de pe marginile de sus, respectiv de jos ale canalului în coloana k.

În general sunt disponibile două sau trei straturi pentru rutare. În continuare se va presupune cazul rutării cu două straturi (numită şi rutare H-V). Segmentele de con-

exiuni orizontale numite *trunchiuri* sunt dispuse de-a lungul pistelor, iar segmentele de conexiuni verticale numite *ramuri* conectează trunchiurile la terminale, după cum se indică în Figura 5.22.

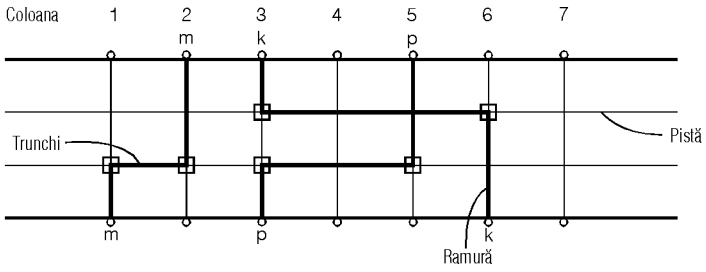

Figura 5.22. Canale, terminale, trunchiuri şi ramuri.

Sarcina programului de rutare prin canale este de a specifica pentru fiecare conexiune un set de segmente de interconectare orizontale şi verticale, ale căror puncte de sfârşit se află pe terminale sau pe pistele canalului. În cazul celulelor standard, obiectivul este de a se utiliza un număr minim de piste pentru efectuarea rutării complete. De aceea, numărul de piste necesare trebuie determinat de programul de rutare. În cazul reøelelor de porøi obiectivul este de a se finaliza rutarea utilizând un număr specificat de piste.

Dacă două segmente orizontale aparøinând unor conexiuni diferite nu se suprapun, ele pot fi asignate aceleiaşi piste. În cazul în care acestea se suprapun, ele trebuie asignate unor piste diferite. În Figura 5.22, segmentul orizontal al conexiunii *m* nu se suprapune cu segmentul conexiunii *p*, dar segmentele orizontale ale conexiunilor *k* şi *p* se suprapun. Există deci constrângeri orizontale asupra conexiunilor.

De asemenea, două conexiuni nu trebuie să se suprapună la o coloană verticală. În Figura 5.22, terminalul corespunzător conexiunii *k* apare în partea de sus a coloanei 3, iar terminalul corespunzător conexiunii *p* apare în partea de jos. Trunchiul conexiunii *k* trebuie plasat deasupra trunchiului conexiunii *p*. Astfel, există şi constrângeri verticale asupra conexiunilor.

5.6.2.2 Grafuri de constrângeri

Fiecărei probleme de rutare prin canale i se pot asocia două grafuri de constrângeri, dintre care una modelează constrângerile orizontale, iar cea de-a doua modelează constrângerile verticale. În cazul ambelor grafuri, fiecare conexiune este reprezentată printr-un vârf.

Graful constrângerilor orizontale GCH(V, E) este un graf nedirecøionat în care un vârf $i \in V$ reprezintă conexiunea i şi muchia $(i, j) \in E$ dacă segmentele orizontale ale conexiunii i şi conexiunii j se suprapun.

Graful constrângerilor verticale GCV(V, E) este un graf direcøionat în care fiecare nod $i \in V$ corespunde conexiunii i, şi fiecare coloană verticală introduce o muchie $(i, j) \in E$ dacă şi numai dacă conexiunea i are un pin în partea de sus a canalului şi conexiunea j are un pin în partea de jos a canalului, în aceeaşi coloană. Astfel, dacă există un ciclu în graful *GCV*, cerinøele de rutare nu pot fi realizate fără divizarea unor conexiuni.

Considerăm lista de conexiuni din Figura 5.23, pentru care vom construi grafurile *GCH* şi *GCV*. Lista de conexiuni poate fi reprezentată prin doi vectori TOP şi BOT definiøi prin TOP = [0,1,6,1,2,3,5] şi BOT = [6,3,5,4,0,2,4]. Pentru a determina dacă două segmente orizontale ale conexiunilor se suprapun se defineşte setul $S(i)$ al co-

nexiunilor ale căror segmente orizontale intersectează coloana *i*. Numărul de elemente din fiecare set se numește densitate locală. Deoarece segmentele orizontale ale conexiunilor distincte nu trebuie să se suprapună, segmentele orizontale a două conexiuni din orice set *S*(*i*) nu trebuie plasate în aceeași pistă orizontală.

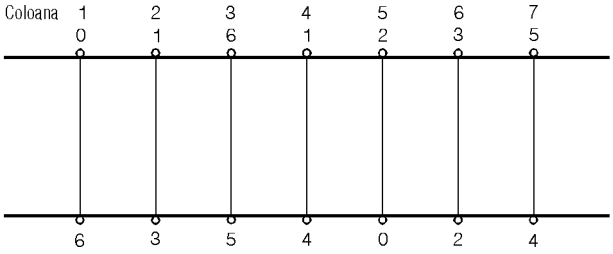

Figura 5.23. Listă de conexiuni.

Pentru exemplul considerat, valorile *S* (*i*) sunt:

S (1) = {6} *S* (2) = {1, 3, 6} *S* (3) = {1, 3, 5, 6}
S (4) = {1, 3, 4, 5} *S* (5) = {2, 3, 4, 5} *S* (6) = {2, 3, 4, 5}
S (7) = {4, 5}

Deoarece elementele din *S* (*i*) reprezintă trunchiuri ale acelor conexiuni care nu trebuie să se afle pe aceeași pistă orizontală, se pot elimina acele seturi care sunt deja subseturi ale altor seturi. De exemplu, *S* (1) = {6} și *S* (2) = {1, 3, 6} sunt subseturi ale *S* (3) = {1, 3, 5, 6}. Seturile rămase după eliminare se numesc *seturi maximale*. Pentru exemplul considerat, seturile maximale sunt:

S (3) = {1, 3, 5, 6} *S* (4) = {1, 3, 4, 5} *S* (5) = {2, 3, 4, 5}

Graful *GCH* este construit prin plasarea unei muchii între vârfurile *i* și *j* dacă ambele aparțin aceluiași set *S* (*k*), pentru un anumit *k*. De exemplu, pentru *S* (3) = {1, 3, 5, 6} se plasează muchii între vârfurile (1,3), (1,5), (1,6), (3,5), (3,6) și (5,6). Graful *GCH* este prezentat în Figura 5.24(a).

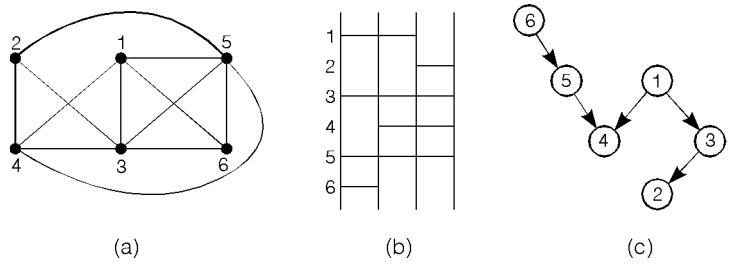

Figura 5.24. (a) Graf al constrângerilor orizontale. (b) Reprezentarea zonelor.
(c) Graf al constrângerilor verticale.

O reprezentare alternativă pentru *GCH* este reprezentarea prin zone, care este o reprezentare grafică a seturilor maximale *S* (*i*). Fiecare set *S* (*i*) este reprezentat printr-o coloană și elementele seturilor maximale *S* (*i*) sunt reprezentate prin segmente de linii după cum se arată în Figura 5.24(b). În Figura 5.24(b) există trei coloane corespunzătoare celor trei seturi maximale *S* (3), *S* (4) și *S* (5). Prima coloană are patru segmente de linii oriazontale la 1, 3, 5 și 6, corespunzătoare elementelor setului maximal *S* (3).

Construirea grafului *GCV* este mai simplã. Pentru fiecare coloanã *k* a canalului care nu conøine un zero în TOP(*k*) şi nici în BOT(*k*) se trasează o muchie direcøionatã de la vârful TOP(*k*) la vârful BOT(*k*). De exemplu, în lista de conexiuni datã, TOP(2) = 1 şi BOT(2) = 3. De aceea, *GCV* va avea o muchie de la vârful 1 la vârful 3. Similar, va exista o muchie de la vârful 6 la vârful 5. Întregul graf este arãtat în Figura 5.24(c).

5.6.2.3 Algoritmul marginii din stânga

Cele mai multe soluøii ale problemei de rutare prin canale se bazează pe algoritmul marginii din stânga, existând diferite extensii şi variaøii ale acestei tehnici. În această secøiune se va prezenta algoritmul de bază. Acest algoritm de rutare a fost propus de Hashimoto şi Stevens. Algoritmul încearcă să maximizeze plasarea segmentelor orizontale în fiecare pistă. Segmentele conexiunilor sunt sortate în ordine crescătoare a distanøei marginii din stânga a acestora faøă de marginea din stânga a canalului. Algoritmul de bază impune restricøia ca fiecare conexiune să fie formată dintr-un singur trunchi, şi ca trunchiurile (segmentele orizontale) să fie rutate pe un strat, iar ramurile (segmentele verticale) să fie rutate pe celălalt strat. În absenøa constrângerilor verticale, algoritmul generează o soluøie cu un număr minim de piste dat de $\max_i |S(i)|$, care este de asemenea limita inferioară a numărului de piste.

Procedura de asignare a trunchiurilor la segmente este următoarea. După sortarea conexiunilor în modul descris mai sus, algoritmul selectează trunchiul corespunzător primei conexiuni şi îl plasează în prima pistă de jos, eliminând apoi conexiunea din listă. Algoritmul parcurge apoi lista rămasă pentru a găsi prima conexiune care nu se suprapune cu conexiunea plasată. Dacă o asemenea conexiune este găsită, ea este plasată în aceeaşi pistă. Procesul este repetat până când nu se mai pot plasa conexiuni în prima pistă. Algoritmul continuă apoi cu plasarea conexiunilor în pista 2 şi apoi în celelalte piste, până când vor fi plasate toate conexiunile din listă. Algoritmul este cunoscut şi cu denumirea de *algoritmul marginii din stânga fără restricøii*, fiind descris în Figura 5.25.

Algorithm *Rutare_prin_canale_fără_restricţii*;
begin
Pas 1. Se sortează conexiunile după poziţiile lor din stânga;
Pas 2. Se selectează conexiunea cu poziţia cea mai din stânga;
Se plasează această conexiune pe pista cea mai de jos disponibilă;
Se elimină conexiunea din listă;
Pas 3. Se continuă parcurgerea listei şi se selectează din listă conexiuni care nu se suprapun cu conexiunile asignate acestei piste;
Se plasează conexiunile în pista curentă şi se elimină din listă;
Pas 4. **if** ($lista \neq \phi$) **then goto** Pas 2;
Pas 5. e**xit**;
end.

Figura 5.25. Algoritmul marginii din stânga fără restricøii.

În absenøa constrângerilor verticale soluøia descrisă anterior este acceptabilă. De obicei există însă constrângeri verticale, şi ignorarea lor ar crea scurtcircuite între conexiuni.

Un algoritm care øine cont de restricøiile verticale este *algoritmul marginii din stânga cu restricøii* elaborat de Perskey, Deutch şi Schweikert. Ca şi în cazul precedent, segmentele orizontale sunt plasate în piste începând din coløul din stânga jos al regiunii de rutare. Algoritmul va plasa un segment orizontal al unei conexiuni numai dacă această conexiune nu are descendenøi în graful constrângerilor verticale. Algoritmul este ilustrat în Figura 5.26.

Algorithm *Rutare_prin_canale_cu_restricţii*;
begin
Pas 1. Se sortează conexiunile după poziţiile lor din stânga;
Pas 2. Se selectează următoarea conexiune *n* cu poziţia cea mai din stânga;
if (*n* nu are descendenţi în *GCV*) **then**
Se plasează *n* pe pista cea mai de jos disponibilă;
Se elimină *n* din lista sortată;
Se elimină *n* din *GCV*;
else goto Pas 2;
endif;
Pas 3. Se continuă parcurgerea listei şi se selectează din listă conexiuni care nu se suprapun cu conexiunile asignate acestei piste şi nu au descendenţi în GCV;
Se plasează conexiunile în pista curentă şi se elimină din listă;
Pas 4. **if** (*lista* ≠ *ϕ*) **then goto** Pas 2;
Pas 5. e**xit**;
end.

Figura 5.26. Algoritmul marginii din stânga cu restricţii.

5.6.2.4 Algoritmii Yoshimura şi Kuh

Dacă există o cale n_1-n_2-n_3-…-n_k în graful constrângerilor verticale, atunci nu pot fi plasate două conexiuni dintre {n_1, n_2, n_3, …, n_k} în aceeaşi pistă. Astfel, dacă cea mai lungă cale din punctul de vedere al numărului de noduri este *k*, sunt necesare cel puţin *k* piste orizontale pentru realizarea interconexiunilor.

În această secţiune se prezintă doi algoritmi propuşi de Yoshimura şi Kuh. Primul algoritm utilizează graful *GCV* şi reprezentarea prin zone a grafului *GCH*, încercând să minimizeze calea cea mai lungă din *GCV*. Aceasta se realizează prin fuzionarea nodurilor din *GCV* (care corespund conexiunilor), astfel încât după fuzionare lungimea căii celei mai lungi este minimizată. Această fuzionare este realizată cu scopul de a minimiza densitatea canalelor. Al doilea algoritm propus de Yoshimura şi Kuh realizează minimizarea căii celei mai lungi prin tehnici de combinare într-un graf bipartit.

Fie *i* şi *j* două conexiuni pentru care

1. nu există suprapunere orizontală în reprezentarea prin zone a grafului *GCV*, şi
2. nu există o cale direcţionată între nodul *i* şi nodul *j* în graful *GCV*.

Aceasta înseamnă că cele două conexiuni *i* şi *j* pot fi plasate în aceeaşi pistă orizontală. Operaţia de fuzionare a conexiunii *i* şi a conexiunii *j* are ca rezultat următoarele:

1. Se modifică graful *GCV* prin fuzionarea nodului *i* şi a nodului *j* într-un singur nod $i \cdot j$.
2. Se actualizează reprezentarea prin zone prin înlocuirea conexiunii *i* şi a conexiunii *j* prin conexiunea $i \cdot j$ care ocupă zonele consecutive inclusiv cele ale conexiunii *i* şi ale conexiunii *j*.

Dacă graful *GCV* original nu are cicluri, atunci nici graful *GCV* actualizat cu nodurile unite nu va avea cicluri [146]. Deoarece se presupune că nu există cicluri în graful *GCV* iniţial, se poate repeta operaţia de fuzionare a conexiunilor fără a genera nici un ciclu în graf. Algoritmul descris în Figura 5.27 realizează fuzionarea sistematică a conexiunilor conform reprezentării prin zone.

Algorithm *Fuzionare1*(z_s, z_t);
begin
Pas 1. $L = \{\}$; z_s = zona cea mai din stânga; z_t = zona cea mai din dreapta;
Pas 2. **for** $z = z_s$ **to** z_t **do**
Pas 3. $L = L + \{$conexiuni care se termină în zona $z\}$;
Pas 4. $R = \{$conexiuni care încep în zona $z + 1\}$;
Pas 5. Se fuzionează L şi R astfel încât să se minimizeze creşterea căii celei mai lungi în graful constrângerilor verticale;
Pas 6. $L = L - \{n_1, n_2, \ldots, n_j\}$, unde $\{n_1, n_2, \ldots, n_j\}$ sunt conexiunile fuzionate în pasul 5;
endfor;
end.

Figura 5.27. Primul algoritm pentru fuzionarea conexiunilor

Pasul principal al algoritmului din Figura 5.27 este Pasul 5, în care sunt selectate şi fuzionate două seturi de conexiuni. Procesul de fuzionare se execută astfel. Se modifică mai întâi graful *GCV* prin adăugarea a două noduri artificiale *s* (sursă) şi *t* (destinaţie), şi a unor arce de la *s* la noduri fără predecesori şi de la noduri fără succesori la *t*.

Fie $P = \{n_1, n_2, \ldots, n_p\}$ şi $Q = \{m_1, m_2, \ldots, m_q\}$ ($p > q$) cele două seturi de conexiuni candidate pentru fuzionare, unde elementele din *P* se află pe o cale verticală separată de cea a elementelor din *Q*. Fie $u(n)$, $n \in P \cup Q$, lungimea căii celei mai lungi de la *s* la *n*, şi $d(n)$, $n \in P \cup Q$, lungimea căii celei mai lungi de la *n* la *t*.

Atunci când sunt fuzionate două noduri $n \in P$ şi $m \in Q$, care se află pe două căi separate de la *s* la *t*, lungimea căii celei mai lungi va crește sau va rămâne aceeaşi. Creşterea exactă, notată cu $h(n, m)$, este dată de:

$$h(n, m) = h^{\max}(n, m) - \max \{u(n) + d(n), u(m) + d(m)\} \quad (5.11)$$

unde $h^{\max}(n, m) = \max \{u(n), u(m)\} + \max \{d(n), d(m)\}$ [146].

Scopul principal este minimizarea lungimii căii celei mai lungi după fuzionare. Timpul necesar pentru găsirea unei fuzionări cu o creştere minimă este însă ridicat. Yoshimura şi Kuh au propus un algoritm euristic pentru fuzionare. În acest algoritm, se alege mai întâi un nod $m \in Q$ aflat pe calea cea mai lungă. În plus, acest nod se află cel mai departe de *s*, cât şi de *t*. Apoi, se alege un nod $n \in P$ astfel încât creşterea căii celei mai lungi după fuzionare să fie minimă. Dacă există mai multe asemenea noduri, este selectat nodul *n* astfel încât $u(n) + d(n)$ să fie aproape maxim şi condiţia $\frac{u(m)}{d(m)} = \frac{u(n)}{d(n)}$ să fie aproape satisfăcută. Această euristică este implementată prin introducerea următoarelor funcţii:

1. Pentru fiecare $m \in Q$,

$$f(m) = C_\infty \times \{u(m) + d(m)\} + \max\{u(m), d(m)\}, \quad C_\infty >> 1$$

2. Pentru fiecare $n \in P$, şi fiecare $m \in Q$,

$$g(n,m) = C_\infty \times h(n,m) - \{\sqrt{u(m) \times u(n)} + \sqrt{d(m) \times d(n)}\}$$

O descriere formală a algoritmului de fuzionare corespunzător pasului 5 din Figura 5.27 este prezentată în Figura 5.28.

În cadrul algoritmului din Figura 5.28 este posibil ca fuzionarea conexiunilor să blocheze fuzionări ulterioare. Pentru a se evita pe cât posibil această situaţie şi pentru ca algoritmul să fie mai flexibil, Yoshimura şi Kuh au introdus un alt algoritm. În acest algoritm este construit un graf bipartit G_h, în care un nod reprezintă o conexiune şi o muchie între nodul *a* şi nodul *b* indică faptul că cele două conexiuni *a* şi *b* pot fi fuzi-

onate. O fuzionare este exprimatã printr-o combinare în graf, care este actualizat în mod dinamic.

```
Algorithm Fuzionare2(P, Q);
begin
        while Q ≠ 0 do
                Se determină m* din Q care maximizează f(m);
                Se determină n* din P care minimizează g(n, m*), şi care
                nu este nici predecesor nici succesor al m*;
                Se fuzionează n* şi m*;
                Se elimină n* şi m* din P, respectiv din Q;
        endwhile;
end.
```

Figura 5.28. Al doilea algoritm pentru fuzionarea conexiunilor.

5.6.2.5 Alte metode de rutare prin canale

Algoritmul marginii din stânga nu poate fi aplicat în cazul în care existã cicluri în graful *GCV*. Deutch a propus un algoritm pentru evitarea buclelor constrângerilor verticale şi pentru reducerea densitãøii canalului. Aceasta se realizează prin divizarea segmentelor orizontale ale unei conexiuni, cu efectul minimizãrii numãrului de piste orizontale. Divizarea orizontalã a unei conexiuni poate fi realizatã numai în poziøiile terminalelor, nefiind permise piste verticale adiøionale. Algoritmul Deutch divizează fiecare conexiune multipin în segmente orizontale individuale. Utilizând noul graf *GCV* creat, în continuare algoritmul se executã similar cu algoritmul marginii din stânga cu restricøii [146].

O euristicã de tip *greedy* pentru rutarea prin canale a fost propusã de Rivest şi Fiduccia. Acest algoritm rutează canalul coloanã cu coloanã începând din stânga. În fiecare coloanã algoritmul aplicã o secvenøã de euristici inteligente pentru maximizarea numãrului de piste disponibile în urmãtoarea coloanã. Nu se utilizează constrângeri orizontale sau verticale, deciziile fiind luate local, la nivelul coloanei. Algoritmul poate rezolva problema de rutare chiar în cazul existenøei ciclurilor în graful *GCV*. Rutarea este terminatã întotdeauna, uneori fiind necesare coloane adiøionale la sfârşitul canalului. Spre deosebire de aløi algoritmi de rutare prin canale, în cazul acestei tehnici o conexiune poate ocupa două piste diferite pânã când algoritmul va decide unirea acestora.

Considerând un exemplu al problemei de rutare prin canale, cu TOP = [5,4,3,2,1,6] şi BOT = [1,2,4,3,5,6], aceastã problemã poate fi specificatã de asemenea ca TOP = [1,2,3,4,5,6] şi BOT = [5,4,2,3,1,6], unde etichetele terminalelor din TOP au fost reordonate pentru a fi în ordine crescãtoare, fiind efectuate modificãrile corespunzãtoare etichetelor din BOT. Conexiunile din BOT sunt o permutare a secvenøei din TOP. Două permutãri p_i şi p_{i+1} se numesc adiacente dacã problema de rutare obøinutã prin asignarea permutãrii p_i pãrøii de jos a canalului şi a permutãrii p_{i+1} pãrøii de sus a canalului, poate fi rutatã într-o pistã. Exemple de permutãri adiacente sunt ilustrate în Figura 5.29.

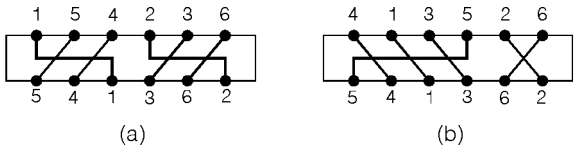

Figura 5.29. Exemple de permutãri adiacente.

Soluøia problemei de rutare prin canale poate fi reprezentatã ca o serie de permutãri $\{p_i\}$, i = 1, 2, …, w, astfel încât p_1 este permutarea datã (BOT) şi p_w = (1, 2, …, n) (TOP), iar p_i este adiacent cu p_{i+1} pentru $0 \le i \le w$. Problema de rutare se reduce atunci la gãsirea unei serii de permutãri adiacente intermediare $\{p_i\}$ astfel încât numãrul de permutãri w este minimizat.

O metodã de rutare bazatã pe permutãri este *rutarea prin interschimbare*, în care douã conexiuni care au terminale adiacente într-o ordine necorespunzãtoare sunt interschimbate. Pentru aceste conexiuni se poate utiliza rutarea X, dupã cum se indicã în Figura 5.30. O serie de permutãri adiacente pot fi realizate utilizând doar rutarea X. Aceasta corespunde factorizãrii permutãrii ca un produs de transpoziøii. Rutarea se execută din partea de jos în cea de sus. Dacã (a_1, a_2, …, a_n) este permutarea de jos, se comparã a_i şi a_{i+1} pentru i = 1,3,5,…,n, şi terminalele sunt interschimbate dacã $a_i > a_{i+1}$. În pasul următor procesul este repetat pentru i = 2,4,6,…,n. Acești paşi sunt repetaøi până când toate terminalele sunt în ordinea corectã. Deoarece douã conexiuni se încrucişeazã o singurã datã dacã terminalele lor nu sunt în ordinea corectã, rutarea obøinutã prin această metodã este o soluøie cu încrucişare minimã.

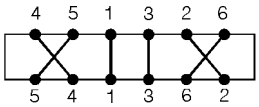

Figura 5.30. Rutarea *X* utilizatã în cadrul metodei de rutare prin interschimbare.

Un algoritm de rutare prin canale bazat pe *sortare* a fost propus de Chaudry şi Robinson [47]. Această metodã presupune cã interconexiunile pot fi rutate nu numai orizontal şi vertical, ci şi la 45° şi 135°. Algoritmul utilizează sortarea bubble. Într-un pas al acestei metode de sortare se interschimbã o pereche de numere o singurã datã, în cazul în care acestea nu sunt în ordinea corectã. Astfel, ca şi în cazul rutării prin interschimbare, rutarea obøinutã prin această metodã este o soluøie cu încrucişare minimã. Deoarece într-un pas al sortării bubble cel puøin unul din numere îşi ocupã poziøia finalã, sunt necesari cel mult n paşi pentru sortarea a n numere. Numãrul de piste va fi deci maxim n, unde n este numãrul de conexiuni.

Pentru rutarea prin canale cu straturi multiple au fost propuse diferite tehnici. Asemenea tehnici au fost propuse de Braun *et al.* [24], şi acestea au fost implementate într-un program de rutare numit Chameleon. Aceste tehnici øin cont de diferite constrângeri tehnologice, de exemplu lãøimea pistelor şi spaøierea dintre ele pot fi specificate independent pentru fiecare strat. Chameleon constã din douã etape, un program de partiøionare şi unul de rutare detaliatã. Rolul programului de partiøionare este de a diviza problema în subprobleme pentru douã şi trei straturi astfel încât suprafaøa globalã a canalului sã fie minimizatã.

O altã abordare a problemei de rutare prin canale cu trei straturi, bazatã pe ideea transformării unei soluøii cu douã straturi într-una cu trei straturi a fost descrisã de Cong *et al.* [61]. Această transformare constã din mai multe etape care pot fi formulate ca probleme de planificare, rutare labirint, respectiv problema cãii celei mai scurte. Deoarece aceste probleme au o complexitate polinomialã, rutarea cu trei straturi poate fi rezolvatã într-un mod optim. Cele mai multe din aceste tehnici pot fi extinse şi pentru patru straturi.

O euristicã pentru rutarea prin canale care asignează interconexiunile pistã cu pistã într-un mod greedy a fost propusã de Ho *et al.* [92]. Structura de date şi strategia utilizatã sunt simple şi pot fi generalizate pentru obøinerea unei clase de euristici pen-

tru rutarea prin canale. Acest algoritm are posibilitatea de revenire prin care cresc şansele de efectuare a rutării complete cu un număr minim de piste.

5.7 Rutarea circuitelor FPGA

Rutarea circuitelor FPGA este mai dificilă decât rutarea clasică, deoarece poziţia segmentelor utilizate pentru interconexiuni este fixă, iar interconectarea segmentelor este posibilă numai în locuri predeterminate. La anumite arhitecturi FPGA cantitatea resurselor de rutare este redusă, ceea ce impune limitări asupra numărului alternativelor de rutare pentru o conexiune. Atunci când două sau mai multe conexiuni trec printr-un canal de rutare comun, există o competiţie pentru resursele de rutare din acel canal. În cazul circuitelor FPGA cu o conectivitate limitată, rezolvarea conflictelor de rutare este esenţială pentru obţinerea unei rutări complete. De asemenea, s-a observat că performanţele circuitelor FPGA sunt limitate adesea de întârzierile de rutare, mai mult decât de întârzierile blocurilor şi ale porţilor logice.

În secţiunile următoare se prezintă unele metode de rutare dezvoltate în mod special pentru circuitele FPGA. În secţiunea 5.7.1 se prezintă problema de rutare a circuitelor FPGA. În secţiunea 5.7.2 se descrie rutarea prin expandarea grafului grosier. În secţiunea 5.7.3 se descrie rutarea pe baza grafurilor cu ponderi multiple. Se prezintă modelarea circuitului FPGA printr-un graf şi se trece în revistă metoda 1-Steiner iterată şi metoda Kou, Markowsky şi Berman. Se descrie apoi metoda care permite optimizarea simultană a obiectivelor multiple.

5.7.1 Problema de rutare a circuitelor FPGA

Metodele obişnuite de rutare globală sau detaliată nu sunt adecvate pentru circuitele FPGA. Rutarea labirint este ineficientă, deoarece această metodă este secvenţială şi astfel, atunci când se rutează o conexiune, nu poate lua în considerare efectele pe care le are această rutare asupra altor conexiuni. Rutarea prin canale nu este adecvată, deoarece problema de rutare detaliată a circuitelor FPGA nu poate fi divizată în general în canale independente. Din aceste motive, sunt necesare metode specifice pentru rutarea circuitelor FPGA.

Algoritmii de rutare pentru circuitele FPGA trebuie să ia în considerare nu numai rutarea cu succes a tuturor conexiunilor, dar trebuie să ţină cont şi de numărul segmentelor de interconectare alocate pentru fiecare conexiune. Prima din aceste cerinţe se referă la *rutabilitate*, iar a doua se referă la *performanţele de vitezã* ale circuitelor implementate.

Problema de rutare a circuitelor FPGA poate fi definită astfel: Fiind dată o arhitectură FPGA împreună cu o configuraţie de asignare a pinilor blocurilor logice şi o colecţie de conexiuni între pini, să se ruteze toate conexiunile fără a se depăşi resursele de rutare disponibile ale circuitului.

În cazul circuitelor VLSI, interconexiunile pot fi plasate în orice poziţie disponibilă din cadrul regiunilor de rutare. De aceea, consideraţiile principale în acest caz sunt de natură *geometricã* (minimizarea lungimii conexiunilor, evitarea coliziunilor cu alte conexiuni etc). În cazul circuitelor FPGA, arhitectura de rutare a acestora oferă un set limitat şi fix de canale de rutare, şi utilizarea unei piste dintr-un canal pentru o conexiune exclude utilizarea acesteia pentru rutarea altor conexiuni. De aceea, problema de rutare a circuitelor FPGA este de natură *combinatorialã* (satisfacerea conectivităţii disponibile, menţinerea fezabilităţii etc).

5.7.2 Rutarea prin expandarea grafului

Această metodă de rutare detaliată a fost elaborată la Universitatea din Toronto, fiind implementată în cadrul programului de rutare *CGE* (*Coarse Graph Expansion*) [27]. Metoda poate lua în considerare efectele pe care le are rutarea unei conexiuni asupra celorlalte, având şi posibilitatea de optimizare a întârzierilor de rutare pentru conexiunile critice. *CGE* presupune că în prealabil a fost efectuată rutarea globală cu scopul de a echilibra densitatea canalelor de rutare. Rezultatele arată că programul *CGE* poate ruta circuite de dimensiuni relativ mari utilizând un număr de piste apropiat de numărul minim determinat de programul de rutare globală.

În scopul aplicării unor tehnici bazate pe grafuri pentru rutare, circuitul FPGA trebuie modelat în prealabil printr-un graf, topologia acestui graf reflectând arhitectura completă a circuitului. Programul de rutare *CGE* poate fi utilizat pentru diferite arhitecturi de circuite constând dintr-o reţea bidimensională de celule logice interconectate prin canale de rutare verticale şi orizontale. Modelul este prezentat în Figura 5.31, fiind format din trei tipuri de componente: celule logice (L), blocuri de conectare (C) şi blocuri de comutare (S). O celulă logică are un număr de pini care se pot conecta la cele patru blocuri C adiacente. Celulele de I/E apar ca celule logice care se află la periferia circuitului.

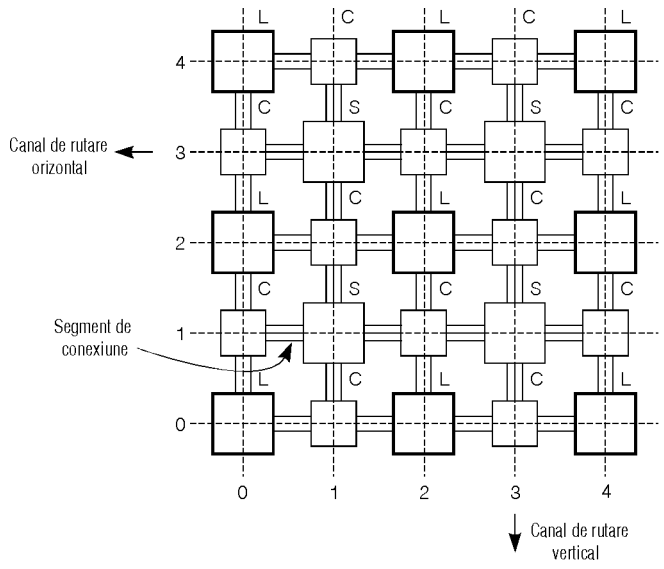

Figura 5.31. Modelul circuitului FPGA.

Blocurile S sunt utilizate pentru conectarea segmentelor de rutare dintr-un canal la cele dintr-un alt canal. În funcţie de topologie, un segment dintr-o laturã a unui bloc S poate fi conectat la toate sau o parte a segmentelor din celelalte laturi ale blocului S. O conexiune poate trece printr-un bloc S prin intermediul unui comutator, sau printr-o legãturã directã. Structura generalã a unui bloc S este ilustratã în Figura 5.32(a). Există două reprezentări pentru comutatoare, fie printr-o linie punctată pentru cele care conectează capetele a două segmente, fie printr-un $\times$ pentru cele care trec direct prin blocul S. Pentru exemplul din Figura 5.32(a), comutatoarele blocului S permit conectarea pistelor orizontale notate cu 1, 2 şi 3 la pistele verticale cu aceleaşi numere. Deşi exemplul din figură este specific, modelul circuitului tratează blocul S ca un bloc de comutare general care poate fi configurat într-un mod oarecare.

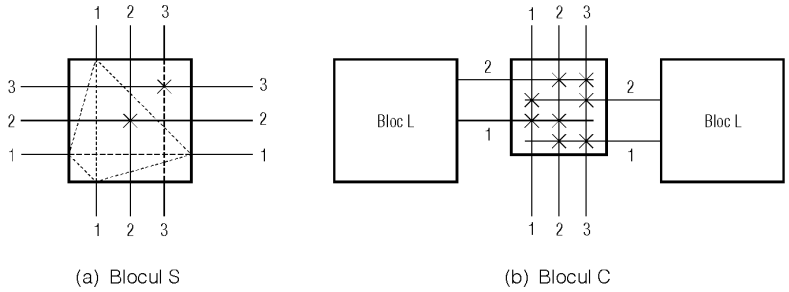

Figura 5.32. (a) Exemplu de bloc *S*. (b) Exemplu de bloc *C*.

Există doi parametri ai arhitecturii FPGA care determină configurația comutatoarelor dintr-un bloc S. Primul este segmentarea canalelor. Această segmentare poate fi specificată prin orice număr de grupuri de piste care au o anumită lungime de segmentare sau o distribuție probabilistică a lungimilor. Al doilea parametru, F_S, reprezintă flexibilitatea blocului și definește numărul celorlalte segmente de interconectare la care se poate conecta un segment care se termină la un bloc S. Pentru exemplul din Figura 5.32(a), segmentul din stânga sus a blocului S se poate conecta la alte trei segmente, astfel că F_S este 3.

Figura 5.32(b) ilustrează un bloc C. Blocurile C sunt utilizate pentru conectarea pinilor celulelor logice la canalele de rutare, prin comutatoare programabile. În funcție de topologia blocului C, pinii celulelor logice pot fi conectați la toate sau o parte din segmentele de interconectare care trec prin acest bloc. Flexibilitatea unui bloc C, F_C, este definită ca numărul segmentelor de interconectare la care se poate conecta fiecare pin al blocurilor logice. Pentru exemplul din figură, fiecare pin se poate conecta la 2 piste verticale, astfel încât F_C este 2. Conexiunile de-a lungul unui canal de rutare pot trece de asemenea direct printr-un bloc C, dar într-o arhitectură tipică nu va fi implicat nici un comutator programabil pentru asemenea conexiuni.

Rutarea globală utilizată este o adaptare a algoritmului de rutare *LocusRoute* pentru celule standard. Algoritmul împarte conexiunile multi-punct în conexiuni cu două puncte, pe care le rutează utilizând căi cu distanță minimă. Scopul principal al algoritmului este distribuirea conexiunilor între canale astfel încât densitățile canalelor să fie echilibrate.

În cadrul rutării globale se definește o cale de rutare grosieră pentru fiecare conexiune prin asignarea unei secvențe de segmente de canale conexiunii respective. Figura 5.33(a) ilustrează o reprezentare a unei căi de rutare tipice. Se indică o secvență de segmente care pot fi alese pentru conectarea unui pin din locația 2,2 a grilei cu un alt pin din locația 4,4. Calea globală reprezintă un graf grosier $G(V, A)$, unde celula logică din locația 2,2 este numită rădăcină a grafului, iar celula din locația 4,4 este numită frunză. Vârfurile V și muchiile A ale grafului sunt identificate prin grila din Figura 5.31.

După rutarea globală, pentru fiecare conexiune cu două puncte programul de rutare detaliată trebuie să aleagă segmente de interconectare specifice pentru implementarea segmentelor de canale asignate în timpul rutării globale. Aceasta necesită informații complete despre arhitectura de rutare a circuitului FPGA, astfel încât *CGE* utilizează detalii despre blocurile logice, blocurile C și blocurile S.

Algoritmul de rutare *CGE* constă din două faze. În prima fază se memorează un număr de alternative pentru fiecare cale detaliată a grafurilor grosiere, iar apoi în faza a doua se efectuează alegeri specifice pentru fiecare conexiune. Se utilizează iterații multiple ale celor două faze în scopul reducerii necesarului de memorie și a timpului de execuție.

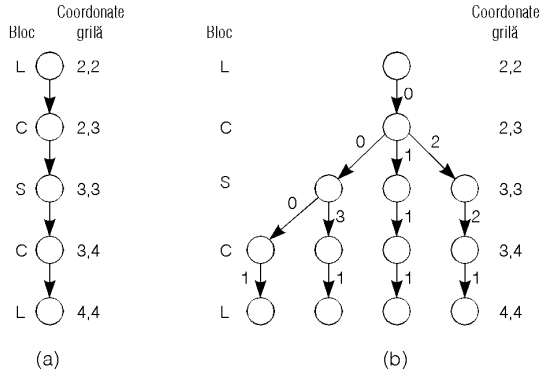

Figura 5.33. (a) Un graf grosier tipic. (b) Graful sãu expandat.

În *faza 1*, *CGE* expandează fiecare graf grosier şi memorează un subset al modurilor posibile în care conexiunea poate fi implementată. Pentru fiecare graf $G(V, A)$, faza de expandare produce un graf expandat, numit $D(N, E)$. Muchiile acestui graf sunt etichetate cu un număr care se referă la un segment specific de interconectare din circuitul FPGA.

În cadrul algoritmului de expandare, procedurile care definesc topologia de conectare a blocurilor C şi S sunt tratate ca funcøii de tip cutie neagrã. Funcøia pentru un bloc C este notată cu $f_c([d_1, d_2, l], d_3)$, iar funcøia pentru un bloc S este notată cu $f_s([d_1, d_2, l], d_3)$. Parametrii din parantezele drepte definesc o muchie care conectează vârful d_1 cu vârful d_2 printr-un segment de interconectare etichetat cu l. O asemenea muchie este notată cu e, $e = (d_1, d_2, l)$. Parametrul d_3 este vârful succesor al lui d_2 în graful G. Dacă pentru conectarea vârfului d_1 cu d_2 se utilizează segmentul etichetat cu l, prin apelul unei asemenea funcøii se va returna un set de muchii reprezentând segmentele care pot fi utilizate pentru conectarea d_2 cu d_3. Această metodă asigură independenøa faøă de o anumită arhitectură de rutare. Rezultatul unei expansiuni este ilustrat în Figura 5.33(b), care indică un posibil graf expandat pentru graful din Figura 5.33(a).

Într-o formă algoritmică, procesul de expandare a fiecărui graf grosier se execută în modul descris în Figura 5.34.

Se creează graful D şi i se atribuie aceeaşi rădăcină ca şi cea a grafului G. Succesorul imediat al rădăcinii grafului D va fi acelaşi ca şi cel al rădăcinii grafului G.
while se traversează D în lăøime, enumerând căile pornind de la fiecare vârf adăugat:
  Se expandează un vârf C din D prin apelul funcøiei $Z = f_c(e_C, n)$. e_C este muchia din D care se conectează la C de la predecesorul acestuia. n este vârful succesor dorit al C din G, iar Z este setul de muchii returnat de $f_c()$. Apelul $f_c()$ adaugă Z la D.
  Se expandează un vârf S din D prin apelul funcøiei $Z = f_s(e_S, n)$. e_S este muchia din D care se conectează la S de la predecesorul acestuia. n este vârful succesor dorit al S din G, iar Z este setul de muchii returnat de $f_s()$. Apelul $f_s()$ adaugă Z la D.
endwhile

Figura 5.34. Faza 1 a algoritmului de rutare *CGE*.

După expandare, fiecare graf $D(N, E)$ poate conøine un număr de căi alternative. În *faza 2*, *CGE* plasează toate căile de la toate grafurile expandate într-o singură listă de căi. Pe baza unei funcøii de cost, algoritmul selectează apoi căi din această listă. Fiecare din acestea defineşte calea detaliată a conexiunii corespunzătoare. Operaøiile efectuate în această fază sunt descrise în Figura 5.35.

```
Se depun toate căile din grafurile expandate în lista de căi
while lista de căi nu este vidă
        if (există căi în lista căilor care sunt esențiale)
                Se selectează calea esențială cu costul c_f cel mai redus
        else if (există căi în lista căilor care corespund conexiunilor critice)
                Se selectează calea critică cu costul c_t cel mai redus
        else
                Se selectează calea cu costul c_f cel mai redus
        Se marchează graful corespunzător căii selectate ca fiind rutat
        Se caută toate căile care sunt în conflict cu calea selectată şi se elimină
                din lista de căi. Dacă o conexiune îşi pierde toate căile sale
                alternative, se re-expandează graful acesteia. Dacă astfel nu
                rezultă noi căi, conexiunea este considerată ca fiind nerutabilă.
        Se actualizează costurile tuturor căilor afectate
endwhile
```

Figura 5.35. Faza 2 a algoritmului de rutare *CGE*.

Fiecare muchie a grafurilor expandate are un cost format din două părți: $c_f(e)$ reflectă competiția între diferitele conexiuni pentru aceleași segmente de interconectare, iar $c_t(e)$ indică întârzierea de rutare asociată cu segmentul de interconectare. *CGE* selectează căile pe baza costului c_t numai dacă o cale corespunde unei conexiuni critice. În caz contrar, căile sunt selectate pe baza costului c_f.

Costul c_f are un rol dublu. În primul rând, acest cost are rolul de a selecta o cale care va avea un efect negativ redus asupra conexiunilor rămase, din punct de vedere al rutabilității. În al doilea rând, acest cost este utilizat pentru identificarea unei căi *esențiale* pentru o conexiune. O cale este numită esențială dacă ea reprezintă singura opțiune rămasă pentru o conexiune, deoarece selecțiile precedente au consumat toate celelalte alternative.

Algoritmul de rutare *CGE* a fost extins pentru arhitecturile care conțin canale de rutare segmentate, cu lungimi diferite. Noul algoritm, *SEGA* (*Segment Allocator*) [28], ține cont de lungimea segmentelor de interconectare în timpul rutării, obținând rezultate semnificativ mai bune față de algoritmul *CGE* din punct de vedere al vitezei circuitelor implementate. În principiu, *SEGA* utilizează aceeași strategie ca și *CGE*, diferența principală constând în funcția de cost. Această funcție are ca scop minimizarea pierderilor datorate alocării unor segmente lungi pentru conexiunile scurte și minimizarea numărului de segmente utilizate în timpul selecției unei căi exacte pentru o conexiune.

5.7.3 Rutarea pe baza grafurilor cu ponderi multiple

Alexander și Robins [2] au propus o abordare unitară pentru rutarea circuitelor FPGA, care permite optimizarea simultană a unor obiective multiple. Această abordare se bazează pe utilizarea unor grafuri multi-ponderate, având următoarele avantaje principale:

- este independentă de arhitectură;
- este eficientă din punct de vedere computațional și relativ ușor de codificat;
- permite optimizarea simultană a unor obiective multiple (de exemplu, lungimea interconexiunilor, congestia etc.), sub controlul proiectantului;
- are un fundament teoretic solid;
- este foarte generală și poate fi aplicată pentru alte probleme CAD, ca și pentru probleme de optimizare combinatorială.

Aceastã metodã combinã tehnici de rutare atât geometrice, cât şi combinatoriale, utilizând avantajele ambelor. În cazul tehnicilor geometrice, restricøiile sunt în principal geometrice, obiectivul tipic fiind minimizarea lungimii totale a conexiunilor. În cazul tehnicilor bazate pe grafuri, restricøiile sunt în principal topologice sau combinatoriale, obiectivul tipic fiind gãsirea unei soluøii fezabile pe baza restricøiilor de rutare date.

Pentru cele douã categorii se pot alege diferite metode existente, rezultând o metodã hibridã. Pentru rutarea geometricã, în [2] s-a ales metoda de rutare 1-Steiner iteratã a lui Kahng şi Robins [99], iar pentru rutarea bazatã pe grafuri s-a ales metoda Kou, Markowsky şi Berman de aproximare a arborilor Steiner.

Pentru aplicarea unor tehnici bazate pe grafuri problemei de rutare a circuitelor FPGA, circuitul trebuie modelat mai întâi ca un graf, astfel încât topologia grafului sã reflecte arhitectura circuitului. Cãile din acest graf corespund unor cãi de rutare fezabile din circuit. Fiecãrei muchii din graf i se asociazã un numãr de ponderi distincte, care reprezintã diferitele criterii de optimizare. Aceastã tehnicã este foarte flexibilã prin faptul cã se pot adãuga cu uşurinøã noi criterii în cadrul modelului prin introducerea unor seturi adiøionale de ponderi.

Atunci când se rutează o conexiune, nodurile şi muchiile reprezentând pinii şi segmentele conexiunii sunt adãugate la graf. Atunci când rutarea unei conexiuni este terminatã, aceleaşi noduri şi muchii sunt eliminate din graf, indicând faptul cã ele nu mai sunt utilizabile.

În secøiunile urmãtoare se descriu pe scurt cele douã metode care sunt combinate în cadrul algoritmului hibrid prezentat.

5.7.3.1 Metoda 1-Steiner iterată

Pentru un set P de n puncte în plan, o muchie (conexiune) între douã puncte $x \in P$ şi $y \in P$ este notatã cu (x, y). Costul unei muchii este distanøa rectiliniarã (Manhattan) dintre capetele acesteia. Un *arbore de acoperire* peste P este un set T de n - 1 muchii cu capetele în P astfel încât graful indus este conectat. Costul unui arbore T, notat cu $\overline{T}$, este suma costurilor muchiilor sale. Un *arbore de acoperire minim* (*minimum spanning tree - MST*) este un arbore de acoperire cu costul minim. Un *arbore Steiner* este un arbore de acoperire peste setul de puncte original P şi un set adiøional (posibil vid) de puncte S (puncte *Steiner*). Problema arborelui Steiner rectiliniar minim poate fi definitã astfel:

Problema arborelui Steiner rectiliniar minim (MRST): Fiind dat un set P de n puncte din planul Manhattan, sã se gãseascã un set S de *puncte Steiner* astfel încât arborele de acoperire minim peste $P \cup S$ sã aibã un cost minim.

Figura 5.36 ilustrează un arbore de acoperire minim şi un arbore Steiner rectiliniar minim pentru un set de puncte fix.

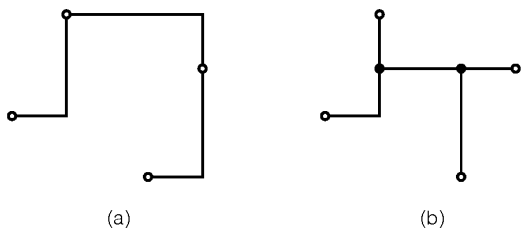

Figura 5.36. Un arbore de acoperire minim (a) şi un arbore Steiner rectiliniar minim (b) pentru o conexiune fixã.

În cadrul cercetărilor legate de problema *MRST* au fost obøinute câteva rezultate fundamentale. Hanan a arătat cã existã întotdeauna un arbore Steiner rectiliniar minim cu punctele Steiner alese dintre punctele de intersecøie ale tuturor liniilor orizontale şi verticale trecând prin toate punctele din P (Figura 5.37). Un alt rezultat major obøinut de Garey şi Johnson a arătat cã în ciuda acestei restricøii asupra spaøiului soluøiilor, problema *MRST* este NP-completã [2].

Un al treilea rezultat important stabileşte cã arborele de acoperire minim rectiliniar reprezintã o aproximare corespunzãtoare a arborelui Steiner rectiliniar minim, cu un raport de performanøã cel mai defavorabil $\overline{MST} / \overline{MRST} \leq \frac{3}{2}$. Aceasta implicã faptul cã orice strategie bazatã pe arborele *MST* care îmbunătãøeşte o topologie iniøialã *MST* va avea de asemenea un raport de performanøã de cel mult $\frac{3}{2}$, ceea ce a determinat elaborarea unui numãr mare de euristici bazate pe metode clasice de construire a unui arbore *MST*.

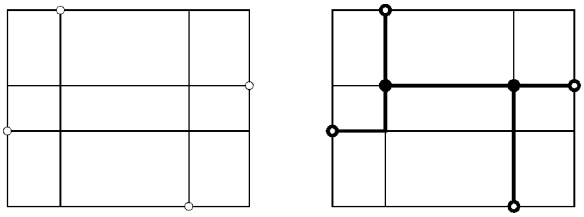

Figura 5.37. Teorema lui Hanan: existã întotdeauna un arbore Steiner rectiliniar minim cu punctele Steiner alese dintre punctele de intersecøie ale tuturor liniilor orizontale şi verticale care trec prin toate punctele.

În literaturã s-a arătat cã toate construcøiile *MRST* bazate pe un arbore *MST* au un raport de performanøã cel mai defavorabil de $\frac{3}{2}$. Acest rezultat a determinat cãutarea unor metode alternative pentru aproximarea *MRST*, rezultatul cel mai bun obøinându-se prin algoritmul 1-Steiner iterat [99]. Performanøele acestui algoritm sunt strict mai bune decât raportul de $\frac{3}{2}$ faøã de cazul optim, obøinându-se o îmbunătãøire de aproximativ 11% faøã de costul arborelui *MST*.

Pentru două seturi de puncte *P* şi *S*, se defineşte reducerea costului *MST* al setului *S* faøã de setul *P* ca fiind:

$$\Delta\overline{MST}(P,S) = \overline{MST}(P) - \overline{MST}(P \cup S) \tag{5.12}$$

Se notează cu *H*(*P*) setul punctelor Steiner candidate aflate la intersecøia tuturor liniilor verticale şi orizontale care trec prin punctele *P*. Pentru un set de puncte *P*, un *punct 1-Steiner* $x \in H(P)$ maximizează $\Delta\overline{MST}(P,\{x\}) > 0$. Metoda 1-Steiner iteratã gãseşte în mod repetat puncte 1-Steiner şi le include în S. Costul *MST* peste $P \cup S$ se va reduce cu fiecare punct adãugat, şi construcøia se terminã când nu existã *x* cu $\Delta\overline{MST}(P,\{x\}) > 0$. Deşi un arbore Steiner poate conøine cel mult *n* - 2 puncte Steiner, algoritmul poate adãuga mai mult de *n* - 2 puncte Steiner; de aceea, în fiecare pas se eliminã punctele Steiner suplimentare având gradul 2 sau mai mic în *MST*. Figura 5.38 ilustrează execuøia algoritmului pentru un exemplu cu 4 puncte.

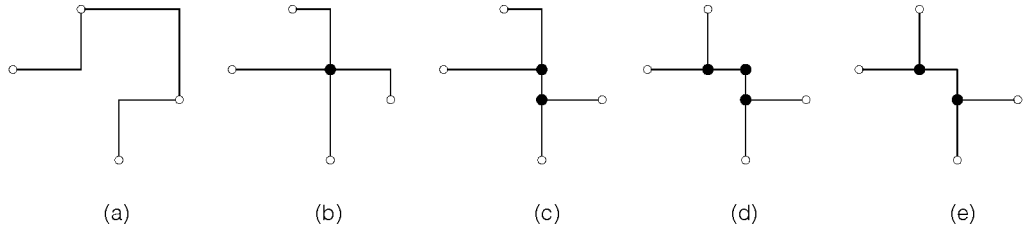

Figura 5.38. Execuţia algoritmului 1-Steiner iterat pentru un exemplu cu 4 puncte. În pasul (d) se formează un arbore Steiner de gradul 2 şi este deci eliminat din topologie (e).

Algoritmul 1-Steiner iterat este descris în Figura 5.39.

Algorithm *1-Steiner*;
/* Intrare: Un set P de n puncte */
/* Ieşire: Un arbore Steiner rectiliniar peste P */
begin
$S = \phi$;
while ($T = \{x \in H(P) \mid \Delta\overline{MST}(P \cup S, \{x\}) > 0\} \neq \phi$) **do**
Se caută $x \in T$ cu $\Delta\overline{MST}(P \cup S, \{x\})$ maxim;
$S = S \cup \{x\}$;
Se elimină din S punctele cu grad ≤ 2 în $MST(P \cup S)$;
endwhile;
return ($MST(P \cup S)$);
end.

Figura 5.39. Algoritmul 1-Steiner iterat.

5.7.3.2 Metoda Kou, Markowsky şi Berman

Problema arborelui Steiner are şi o versiune pentru grafuri. Fiind dat un graf $G = (V, E)$, unde V este setul de noduri, iar $E \subseteq V \times V$ este un set de muchii ponderate, trebuie acoperit un subset al nodurilor, utilizând nodurile rămase ca puncte Steiner. Fiecare muchie e_{ij} din graf are o pondere w_{ij}, iar muchiile lipsă din graf se presupun că au o pondere infinită. Scopul este minimizarea costului total al arborelui de acoperire. Figura 5.40 prezintă un graf şi un arbore Steiner minim care acoperă subsetul de noduri care sunt puse în evidenţă. Problema arborelui Steiner minim pentru grafuri (*Graph Steiner Minimum Tree - GSMT*) poate fi definită astfel:

Problema arborelui Steiner minim pentru grafuri (GSMT): Fiind dat un graf ponderat $G = (V, E)$ şi un set de terminale $N \subseteq V$ care trebuie conectate, să se găsească un arbore $T = (V', E')$ cu $N \subseteq V' \subseteq V$ şi $E' \subseteq E$ astfel încât $\sum_{e_{ij} \in E'} w_{ij}$ să fie minimizat.

Problema *GSMT* nu este mai ušor de soluţionat decât problema geometrică *MRST* corespunzătoare, deoarece ultima este un caz special al primei probleme. Rezultă că problema *GSMT* este NP-completă. Algoritmul Kou, Markowsky şi Berman (KMB) soluţionează problema *GSMT* într-un timp polinomial, având o limită de maxim $2 \cdot (1 - \frac{1}{L})$ faţă de soluţia optimă, unde L este numărul minim de frunze din oricare soluţie optimă.

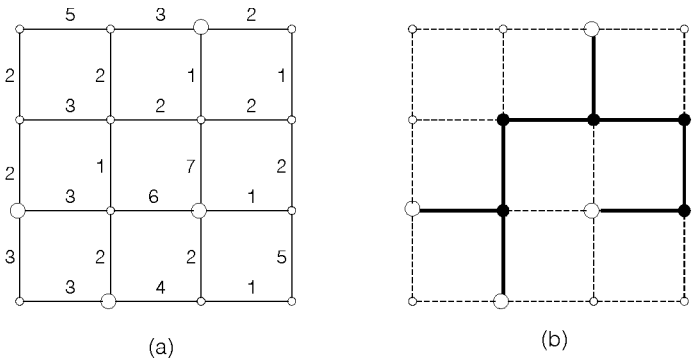

Figura 5.40. (a) Un exemplu de graf. (b) Un arbore Steiner minim care acoperă nodurile evidenøiate.

Algoritmul *KMB* poate fi descris astfel. În primul rând, se construieşte graful complet *G'* peste *N*, ponderea fiecărei muchii e_{ij} fiind egalã cu $dist_G(N_i, N_j)$, deci costul cãii celei mai scurte din *G* între N_i şi N_j. În al doilea rând, se determinã *MST*(*G'*), arborele de acoperire minim al *G'*, şi se expandeazã fiecare muchie e_{ij} din *MST*(*G'*) în calea cea mai scurtã corespunzãtoare, notatã cu *cale*(N_i, N_j), obøinându-se un subgraf *G''* care acoperã *N*. În sfârşit, se determinã arborele de acoperire minim *MST*(*G''*), şi se eliminã muchii din *MST*(*G''*) pânã când toate frunzele aparøin lui *N*. Arborele rezultat este o aproximare a *GMST*, fiind notat cu *KMB*, costul acestui arbore fiind $\overline{KMB}$. O descriere formalã a algoritmului *KMB* este prezentatã în Figura 5.41.

Algorithm *KMB*;
/* Intrare: Un graf $G = (V, E)$ cu ponderile muchiilor w_{ij}, şi o conexiune $N \subseteq V$ */
/* Ieşire: Un arbore cu cost redus $T' = (V', E')$ care acoperă N */
begin
$G' = (N, N \times N)$, cu ponderile muchiilor $w'_{ij} = dist_G(N_i, N_j)$;
Se determină $T = (N, E'') = MST(G')$;
Fie graful $G'' = \bigcup_{e_{ij} \in E''} cale_G(N_i, N_j)$;
Se determină $T' = MST(G'')$;
Se elimină din T' toate nodurile frunză care nu fac parte din N;
return T';
end.

Figura 5.41. Algoritmul *KMB* pentru problema *GSMT*.

De notat cã existã algoritmi de construire a arborelui Steiner pentru grafuri care au limite de performanøã mai bune decât cele ale metodei KMB. De exemplu, euristica Steiner a lui Zelikovsky are un raport de performanøã pentru cazul cel mai defavorabil de $\frac{11}{6}$. Complexitatea acestei metode este însã excesivã pentru cazurile practice.

5.7.3.3 Algoritm de rutare hibrid

Combinarea euristicii geometrice 1-Steiner cu algoritmul *KMB* bazat pe grafuri permite modelarea arhitecturii circuitului FPGA într-un mod natural (ca un graf), şi utilizarea în acelaşi timp a informaøiilor geometrice despre circuit în scopul obøinerii unor soluøii îmbunãtãøite de rutare. Metoda hibridã rezultatã este numitã algoritm 1-Steiner iterat pentru grafuri (*Graph Iterated 1-Steiner - GI1S*). Se prezintã varianta

algoritmului pentru cazul în care muchiile grafului au câte o singură pondere. În secţiunea 5.7.3.4 se prezintă modul în care metodologia de rutare poate fi extinsă pentru grafuri cu ponderi multiple.

Metoda *GI1S* este în principiu o adaptare a metodei 1-Steiner iterate pentru grafuri. Însă, atunci când trebuie acoperit un subset *N* al nodurilor dintr-un graf, noţiunea arborelui de acoperire minim nu mai este bine definită. În esenţă, un arbore de acoperire pentru *N* este acum un arbore Steiner, care nu mai poate fi construit într-un mod eficient, deoarece problema este NP-completă. Această dilemă poate fi rezolvată prin înlocuirea construcţiei *MST* cu construcţia *KMB*. Deci, în locul utilizării unei subrutine *MST* pentru a determina reducerea costului datorată unui punct Steiner candidat, se utilizează algoritmul *KMB* în acest scop. Astfel, fiind dat un graf $G = (V, E)$, conexiunea $N \subseteq V$, şi un set de puncte Steiner candidate, se poate defini reducerea costului ca fiind:

$$\Delta\overline{KMB}_G(N,S) = \overline{KMB}_G(N) - \overline{KMB}_G(N \cup S) \tag{5.13}$$

Algoritmul *GI1S* începe prin construirea arborelui *KMB*. Apoi, în fiecare iteraţie se determină noduri Steiner candidate care reduc costul total *KMB*, noduri care sunt incluse în setul de noduri Steiner *S*. Costul arborelui *KMB* peste $N \cup S$ se va reduce cu fiecare nod adăugat, construcţia fiind terminată atunci când nu mai există $x \in V$ cu $\Delta\overline{KMB}(N \cup S, \{x\}) > 0$. Metoda este descrisă în Figura 5.42.

Algorithm *GI1S*;
/* Intrare: Un graf ponderat $G = (V, E)$ şi o conexiune $N \subseteq V$ */
/* Ieşire: Un arbore cu cost redus $T' = (V', E')$ care acoperă N */
begin
$S = \phi$;
while ($T = \{x \in V - N \mid \Delta\overline{KMB}_G(N \cup S, \{x\}) > 0\} \neq \phi$) **do**
Se caută $x \in T$ cu $\mid \Delta\overline{KMB}_G(N \cup S, \{x\})$ maxim;
$S = S \cup \{x\}$;
Se elimină din S punctele Steiner cu grad ≤ 2 în $KMB(N \cup S)$;
endwhile;
return ($KMB_G(N \cup S)$);
end.

Figura 5.42. Algoritmul 1-Steiner iterat pentru grafuri care utilizează euristica *KMB*.

Deoarece un arbore Steiner optimal pentru o conexiune cu 3 pini poate avea cel mult un punct Steiner, algoritmul *GI1S*, care selectează cel mai bun candidat dintre punctele Steiner, garantează găsirea soluţiei optime pentru orice conexiune cu 3 pini. Complexitatea în timp a algoritmului pentru rutarea unei singure conexiuni cu *n* pini este limitată superior de $O(n \cdot |G| \cdot [n^2 \log n + |G|])$. $|G|$ este numărul de muchii ale grafului de rutare, $|G| = O(B \cdot S)$, unde *B* este numărul total al blocurilor de comutare ale circuitului, iar *S* este numărul de piste din fiecare canal [2].

5.7.3.4 Optimizarea simultană a obiectivelor multiple

Euristica *GI1S* se poate generaliza pentru grafuri cu ponderi multiple, unde fiecare criteriu de optimizare are un set separat de ponderi ale muchiilor grafului. Optimizarea simultană se realizează prin transformarea acestor ponderi multiple într-o singură medie ponderată, care se utilizează apoi pentru executarea *GI1S* în modul normal. Ponderile constau din *k* numere reale $d_1, d_2, \ldots, d_k$, unde $\sum_{i=1}^{k} d_i = 1$. Această

metodã permite un control de cãtre proiectant al diferitelor obiective. În continuare se prezintã bazele teoretice ale grafurilor cu ponderi multiple.

Fie $V = \{v_1, v_2, \ldots, v_n\}$ un set de $|V| = n$ vârfuri, şi $E \subseteq V \times V$ un set de $|E| = m$ muchii. Un graf *k*-ponderat $G = (V, E)$ este definit ca un graf ponderat cu o funcøie pondere $w: E \rightarrow \Re^k$. Deci, fiecãrei muchii $e_{ij} \in E$ i se asociazã un vector de *k* ponderi cu valori reale $\vec{w}_{ij} = (w_{ij1}, w_{ij2}, \ldots, w_{ijk})$.

Fie $\vec{d} = (d_1, d_2, \ldots, d_k)$ un vector de *k* parametri de compromis, unde $0 \le d_i \le 1$ pentru $0 \le i \le k$, şi $\sum_{i=1}^{k} d_i = 1$. Pornind de la graful *k*-ponderat $G = (V, E)$ şi de la parametrii de compromis $\vec{d}$ se construieşte un nou graf ponderat de compromis $\hat{G}(\vec{d}) = (V, E)$ cu funcøia pondere $w'_{ij} = \vec{d} \cdot \vec{w}_{ij} = \sum_{m=1}^{k} d_m \cdot w_{ij}$. Graful $\hat{G}$ este un graf ponderat obişnuit având aceeaşi topologie ca şi G, dar ale cãror ponderi reprezintã mediile ponderate ale multi-ponderilor lui G, faøã de parametrii de compromis $\vec{d}$.

Fie $\vec{u} = (1, 1, \ldots, 1)$ vectorul unitar, şi fie $\vec{v}_i = (0, 0, \ldots, 0, v_i, 0, 0, \ldots, 0)$ vectorul obøinut din vectorul $\vec{v}$ prin utilizarea v_i în poziøia *i*, restul poziøiilor fiind setate la zero. Deci, $\vec{u}_i$ indicã vectorul constând din zerouri în toate poziøiile cu excepøia poziøiei *i*, care va conøine 1. Un graf *k*-ponderat poate induce *k* grafuri distincte, fiecare cu aceeaşi topologie, dar cu ponderile muchiilor restrânse la una singurã din cele *k* componente ale funcøiei pondere $\vec{w}$; aceste *k* grafuri induse sunt notate cu $G_i = \hat{G}(\vec{u}_i)$.

Se defineşte arborele minim de acoperire (*MST*) pentru un graf *G* multi-ponderat în raport cu parametrii de compromis $\vec{d}$ ca arborele *MST* normal peste graful de compromis $\hat{G}(\vec{d})$, şi se noteazã cu $MST(\hat{G}(\vec{d}))$. Similar, se poate construi arborele *MST* pentru fiecare din cele *k* grafuri induse G_i, acestea fiind notate cu $MST(G_i)$.

Fiind dat un graf *G* *k*-ponderat şi un vector de parametri $\vec{d}$, $\overline{MST}(\hat{G}(\vec{d}))$ (costul grafului de compromis) satisface urmãtoarea inegalitate pentru grafurile metrice [2]:

$$\sum_{i=1}^{k} d_i \cdot \overline{MST}(G_i) \le \overline{MST}(G(d)) \le (n-1) \cdot \sum_{i=1}^{k} d_i \cdot \overline{MST}(G_i) \qquad (5.14)$$

5.8 Algoritm propus pentru rutarea circuitelor FPGA Atmel 6000

În aceastã secøiune se descrie un algoritm de rutare elaborat pentru circuitele FPGA *Atmel* din seria 6000, care a fost implementat pentru circuitul *Atmel* 6002. Algoritmul de rutare propus executã simultan rutarea globalã şi cea detaliatã. Avantajul acestei abordãri este cã estimarea preliminarã a rutãrii globale poate fi corectatã în mod corespunzãtor şi imediat. Algoritmul poate considera efectele deciziilor de rutare luate pentru o conexiune asupra celorlalte conexiuni, rezolvând astfel conflictele de rutare. Se pot efectua douã tipuri de optimizãri: o optimizare din punct de vedere al numãrului de celule utilizate (al *suprafeøei* ocupate de acestea), sau o optimizare din punct de vedere al *vitezei*.

5.8.1 Arhitectura de rutare a circuitelor FPGA Atmel 6000

Arhitectura *Atmel* este formatã dintr-o reøea simetricã de celule identice. Reøeaua este continuã de la o margine a circuitului la alta, cu excepøia repetoarelor de magistralã care sunt amplasate la o distanøã de opt celule [7]. Figura 5.43 prezintã reøeaua de magistrale.

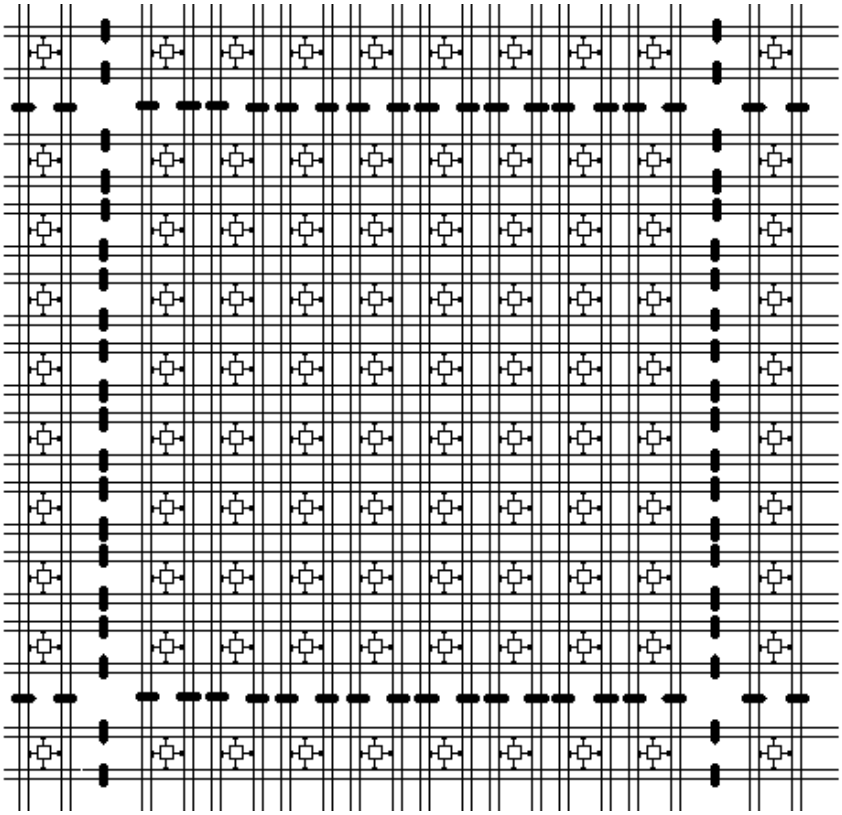

Figura 5.43. Reøeaua de magistrale a circuitelor FPGA *Atmel* 6000.

În plus faøã de implementarea funcøiilor logice combinaøionale øi secvenøiale, celulele pot fi utilizate øi pentru interconectarea blocurilor logice pe distanøe scurte, o caracteristicã importantã care este utilizatã atât de algoritmii de plasare descriøi în capitolul 4, cât øi de algoritmul de rutare. Magistralele permit o comunicaøie rapidã øi eficientã pe distanøe medii øi lungi. Existã douã tipuri de magistrale: magistrale locale øi magistrale expres (Figura 5.44).

Magistralele locale reprezintã legãtura dintre celule øi reøeaua de interconexiuni. Existã douã magistrale locale pentru fiecare coloanã de celule, nord-sud 1 øi 2 (NS1, NS2), øi douã magistrale locale pentru fiecare linie de celule, est-vest 1 øi 2 (EW1, EW2). Într-un sector de opt celule fiecare magistralã localã este conectatã la fiecare celulã din coloana sau linia respectivã, asigurând astfel accesul fiecãrei celule la douã magistrale nord-sud øi douã magistrale est-vest. În plus, fiecare celulã are posibilitatea de a ruta un semnal între magistralele NS1 øi EW1, sau între magistralele NS2 øi EW2 (întoarcere de 90°).

Magistralele expres nu sunt conectate direct la celule, asigurând astfel o vitezã mai ridicatã. Fiecãrei magistrale locale îi corespunde o magistralã expres, astfel încât existã douã magistrale expres pentru fiecare coloanã øi douã magistrale expres pentru fiecare linie de celule.

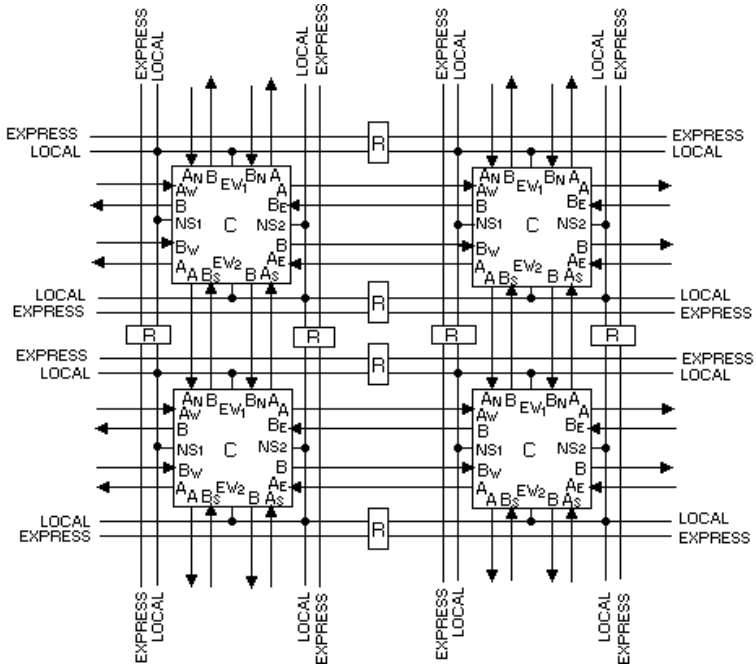

Figura 5.44. Magistrale locale şi magistrale expres.

Fiecare magistralã localã şi expres este divizatã în segmente de cãtre unitãøi de conectare, numite *repetoare*, care sunt spaøiate la un interval de opt celule. Repetoarele sunt aliniate în linii şi coloane, partiøionând astfel reøeaua în sectoare de 8×8 celule. Fiecare repetor este asociat cu o pereche de magistrale localã/expres, şi de fiecare parte a repetorului existã conexiuni la un segment de magistralã localã şi la un segment de magistralã expres. Fiecare repetor poate fi programat pentru a realiza una din cele 21 de funcøii de interconectare. Aceste funcøii sunt simetrice atât faøã de cele douã laturi ale repetorului, cât şi faøã de cele douã tipuri de magistrale. Dintre funcøiile pe care le pot realiza repetoarele se pot aminti urmãtoarele:

- Izolarea segmentelor de magistralã între ele
- Conectarea a douã segmente ale unei magistrale locale
- Conectarea a douã segmente ale unei magistrale expres
- Implementarea unui transfer între o magistralã localã şi una expres

În toate aceste cazuri, fiecare conexiune asigurã regenerarea semnalelor, şi deci este unidirecøionalã. Pentru conexiuni bidirecøionale, funcøia de bazã a repetoarelor NS2 şi EW2 este extinsã cu o conexiune programabilã specialã, care permite comunicaøia bidirecøionalã între segmentele magistralelor locale. Aceastã opøiune este utilizatã în mod special pentru a implementa magistrale lungi cu trei stãri.

5.8.2 Modelarea circuitului printr-un graf

Pentru utilizarea unor tehnici bazate pe grafuri pentru problema de rutare a circuitelor FPGA, circuitul trebuie modelat mai întâi ca un graf, astfel încât topologia grafului sã reflecte întreaga arhitecturã a circuitului. Cãile din acest graf corespund unor cãi de rutare fezabile din circuit, şi invers. Figura 5.45(a) prezintã o porøiune a arhitecturii circuitelor FPGA *Atmel*, iar Figura 5.45(b) ilustreazã graful de rutare corespunzãtor.

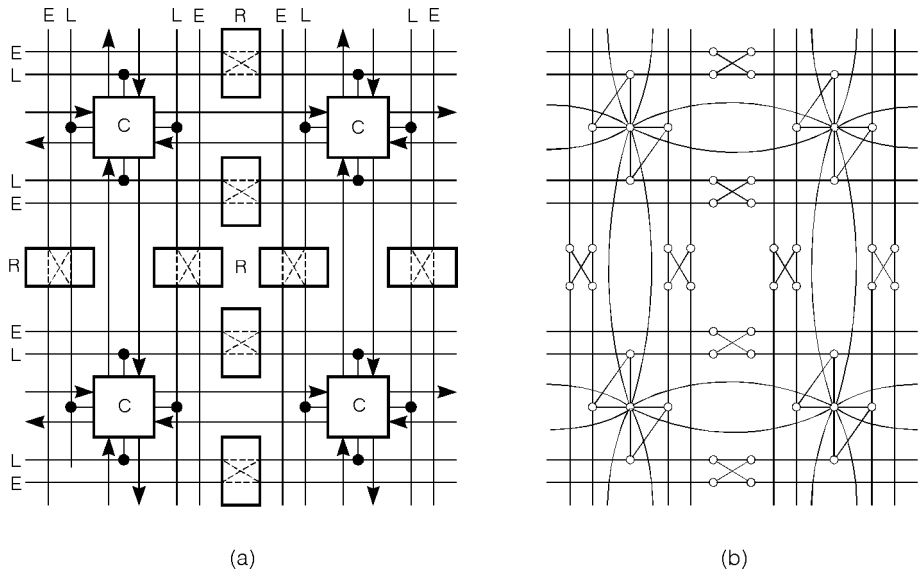

Figura 5.45. (a) Arhitectura circuitelor FPGA *Atmel*. (b) Graful de rutare corespunzãtor.

Fie $G = (V, E)$ graful de rutare al circuitului, unde fiecare muchie $e_{ij} \in E$ are o pondere w_{ij}, care corespunde lungimii segmentului de rutare al circuitului FPGA. O conexiune $N = \{n_0, n_1, \ldots, n_k\} \subseteq V$ este un set de pini care trebuie conectaøi împreunã din punct de vedere electric, n_0 fiind sursa semnalului, iar ceilaløi pini reprezintã destinaøii. O soluøie de rutare pentru o conexiune este un arbore $T \subseteq G$ care acoperã N, iar costul arborelui T este suma ponderilor muchiilor sale.

5.8.3 Descrierea algoritmului de rutare [15]

Problema de rutare este soluøionatã de obicei în douã etape. În prima etapã se executã *rutarea globalã*, prin care se asigneazã fiecare conexiune unui subset al regiunilor de rutare, creându-se astfel un nou set de probleme restricøionate de rutare. În a doua etapã, cea de *rutare detaliatã*, sunt selectate segmente de rutare şi comutatoare programabile specifice pentru fiecare conexiune, în cadrul restricøiilor setate în cadrul rutãrii globale.

Aceastã abordare are avantajul cã în fiecare etapã se poate soluøiona în mod eficient o parte a problemei de rutare. Cu toate acestea, împãrøirea problemei de rutare în douã subprobleme conduce la rezultate globale care nu sunt optime, chiar dacã ambele probleme sunt soluøionate în mod optim. De aceea, ori de câte ori este posibil, o asemenea împãrøire trebuie evitatã.

Algoritmul de rutare implementat executã simultan rutarea globalã şi cea detaliatã. Un avantaj al acestei metode este cã estimarea preliminarã a rutãrii globale poate fi imediat corectatã. De asemenea, algoritmul poate lua în considerare efectele secundare pe care le au deciziile de rutare luate pentru o conexiune asupra celorlalte conexiuni, rezolvând astfel conflictele de rutare. Aceastã problemã este descrisã în continuare.

În cazul rutãrii circuitelor FPGA, o problemã importantã este cã alegerea unei resurse de rutare pentru o conexiune poate bloca o altã conexiune. Figura 5.46 ilustreazã trei vederi ale unei secøiuni dintr-un canal de rutare a unui circuit FPGA (canalele verticale nu sunt prezentate în figurã). Pentru fiecare din acestea, figura indicã opøiunile de rutare disponibile pentru trei conexiuni diferite, A, B şi C. Un segment de

interconectare este indicat printr-o linie întreruptă, o rutare posibilă printr-o linie continuă, iar o conexiune programabilă printr-un ×.

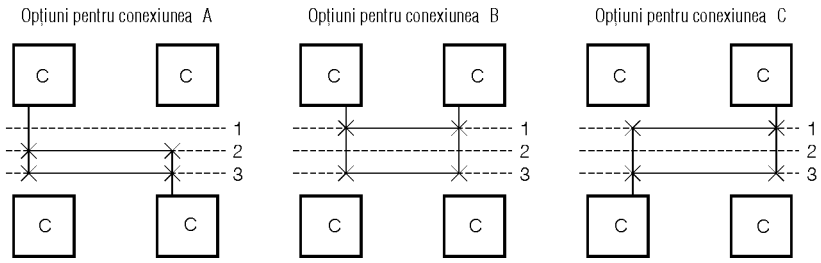

Figura 5.46. Conflicte de rutare.

Presupunem că programul de rutare efectuează mai întâi conexiunea A. Dacă se alege segmentul de interconectare 3, una din conexiunile B sau C nu se va putea ruta, deoarece ambele depind de singura opțiune rămasă, și anume segmentul 1. Alegerea corectă pentru conexiunea A este segmentul 2, caz în care atât B cât și C vor putea fi rutate. Acest exemplu simplu ilustrează faptul că deciziile de rutare luate pentru o conexiune pot bloca o altă conexiune. Asemenea conflicte pentru resursele de rutare reprezintă motivul principal pentru care rutarea circuitelor FPGA poate fi mai dificilă decât rutarea clasică.

Algoritmul de rutare ține cont de următoarele aspecte, care sunt specifice pentru arhitectura FPGA *Atmel*:

1. Pentru rutarea semnalelor pe distanțe scurte, algoritmul utilizează celule libere în locul magistralelor. Magistralele sunt rezervate pentru rutarea semnalelor pe distanțe mai mari (mai mult de cinci celule), pentru semnale cu trei stări, sau pentru semnale cu un fan-out ridicat.
2. Algoritmul utilizează magistrale expres ori de câte ori este posibil. Aceste magistrale nu sunt conectate direct la celule, de aceea ele au o încărcare capacitivă mai redusă și sunt mai rapide decât magistralele locale. De asemenea, prin utilizarea unei magistrale expres în locul uneia locale, magistrala locală este eliberată pentru alte conexiuni. Totuși, înlocuirea unei magistrale locale printr-o magistrală expres nu este posibilă în anumite cazuri:

- la conectarea directă la o celulă
- la utilizarea unui semnal bidirecțional
- la efectuarea unei întoarceri cu 90°

3. Pentru creșterea performanțelor, algoritmul limitează numărul segmentelor magistralelor locale prin care trece un semnal și utilizează repetoarele numai dacă este necesar. Se efectuează ramificarea semnalului de pe o magistrală expres la magistrala locală la fiecare repetor, ceea ce are un efect benefic atunci când valoarea fanout-ului este mai mare decât opt, sau dacă semnalul trece prin mai mult de un repetor. Figura 5.47 ilustrează un exemplu de ramificație.

Semnalul X este rutat printr-o magistrală expres și se ramifică în dreptul repetoarelor la segmentele magistralelor locale care sunt conectate la celulele A și B. Dacă semnalul X ar fi fost rutat prin magistrala locală la celulele A1, A2 și A3, și printr-un repetor la celulele B1, B2 și B3, încărcarea celulelor A ar influența viteza semnalului care ajunge la celulele B.

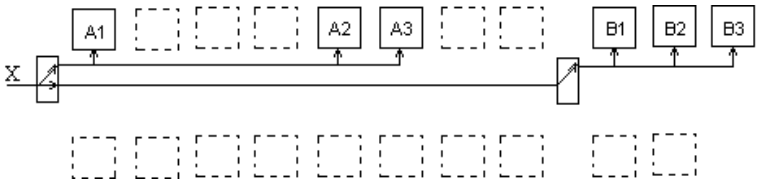

Figura 5.47. Un exemplu de ramificaøie.

Fiecãrui segment de interconectare al circuitului i se asociazã douã costuri:

- Costul distanøei (C_d), care reflectã întârzierile de rutare asociate cu segmentul de interconectare;
- Costul competiøiei (C_c), care contorizează legãturile care se aflã în competiøie pentru acelaşi segment.

Fiecãrei cãi din lista de conexiuni i se asociazã de asemenea douã costuri:

- Suma costului distanøelor (S_d) pentru segmentele cãii;
- Suma costului competiøiei (S_c) pentru segmentele cãii.

Algoritmul nu poate considera toate posibilitãøile de interconectare din cadrul circuitului într-o singurã etapã, în scopul reducerii necesarului de memorie şi al timpului de execuøie. Acesta este motivul pentru care se utilizează o metodã iterativã. În prima etapã se consideră numai acele cãi posibile pentru o conexiune care corespund costului C_d minim. Dacã aceste cãi sunt în conflict cu conexiunile deja rutate, algoritmul continuã cãutarea pornind de la costul cãii eşuate.

Algoritmul de rutare este descris în Figura 5.48 [15].

```
Algorithm Rutare_Atmel;
begin
        Se construieşte graful de conectivitate pe baza arhitecturii circuitului Atmel;
        Se divid conexiunile multi-punct în conexiuni cu două puncte, construid
                graful de rutare;
        Se determină distanţa minimă a căii de rutare pentru fiecare conexiune
                care nu poate fi împărţită în legături directe;
        Se depun toate alternativele de rutare într-o listă a conexiunilor;
        while (lista conexiunilor nu este vidă) do
                Se sortează lista conexiunilor după numărul alternativelor de rutare;
                Se selectează conexiunea din capul listei;
                Se sortează căile posibile după Sc;
                if (lista căilor nu este vidă) do
                        Se rutează conexiunea selectată utilizând calea cu cost minim;
                        Se marchează conexiunea în graful de conectivitate;
                        Se determină toate căile care sunt în conflict cu calea
                                selectată şi se elimină din listă;
                else do
                        Se rerutează conexiunea selectată;
                        if (conexiunea nu poate fi rutată)
                                Se marchează circuitul ca fiind nerutabil;
                        endif;
                endif;
        endwhile;
end.
```

Figura 5.48. Algoritmul de rutare pentru circuitele FPGA *Atmel* 6000.

În prima etapă, se construiește graful de conectivitate pe baza structurii circuitului FPGA, iar apoi se construiește graful de rutare pe baza rezultatelor obținute după maparea tehnologică și plasare. Prin legături directe se desemnează acele legături care nu implică utilizarea unei magistrale.

În următoarea etapă, algoritmul încearcă divizarea conexiunilor în legături directe în cazul în care acestea nu trebuie să treacă prin mai mult de cinci celule. Pentru fiecare conexiune, algoritmul determină distanța minimă a căii de rutare, și memorează toate alternativele de rutare într-o listă a conexiunilor.

Algoritmul poate efectua două tipuri de optimizări:

1. din punct de vedere al spațiului ocupat
2. din punct de vedere al vitezei

Pentru optimizarea din punctul de vedere al *spațiului*, algoritmul sortează mai întâi conexiunile după numărul alternativelor posibile de rutare (numărul căilor din graf, utilizând costul S_c), astfel încât conexiunile care au un număr mai redus de alternative vor fi mai prioritare. După selectarea unei conexiuni prin această procedură de sortare, algoritmul utilizează o funcție de cost pentru evaluarea costului fiecărei căi disponibile, și alege calea cu costul S_d minim.

Pentru optimizarea din punctul de vedere al *vitezei*, algoritmul sortează mai întâi conexiunile după lungimea lor (utilizând costul S_d). Astfel, va determina ca liniile lungi să aleagă magistralele expres, care sunt mai rapide. Dintre toate conexiunile posibile, se alege cea cu costul S_c minim.

În etapa de rutare, algoritmul selectează alternativele de rutare cu costul S_c sau S_d minim. În etapa de rerutare, se încearcă găsirea altor căi posibile, pe baza grafului de conectivitate actualizat. În această etapă, în graful de conectivitate se marchează punctele de conectivitate utilizate de conexiunile deja rutate. Celula care generează semnalul conexiunii este de asemenea marcat în graful de conectivitate. Astfel, în etapa de rerutare alternativele de rutare care pornesc de la un punct de conectivitate care este utilizat pentru alte conexiuni vor fi luate în considerare.

Funcția de cost utilizată în cazul optimizării pentru rutabilitate are rolul de a selecta o cale de rutare care va avea un efect negativ redus asupra conexiunilor rămase, din punct de vedere al rutabilității. Această funcție împiedică selectarea căilor care conțin segmente de interconectare pentru care există un număr mare de cereri. Acest aspect a fost ilustrat în Figura 5.46, unde conexiunea A trebuie rutată utilizând segmentul de interconectare numărul 2, deoarece există o cerere mai mare pentru segmentul numărul 3.

Pentru a determina dacă există o cerere mare la o muchie *e* din graful de rutare al unei conexiuni, se pot contoriza aparițiile muchiei *e* în grafurile altor conexiuni. Însă, probabilitatea de utilizare a unei muchii depinde de numărul alternativelor posibile la utilizarea acestei muchii. Deci, costul unei muchii *e* pentru care există alte *j* alternative (muchii în paralel cu *e*) va fi definit astfel:

$$cost_r(e) = \sum_j \frac{1}{alt(e_j)} \tag{5.15}$$

unde $alt(e_j)$ este numărul muchiilor paralele cu e_j. Astfel, cu cât *e* apare în mai multe grafuri, cu atât costul său va fi mai ridicat. Aceasta indică faptul că cererea la *e* este ridicată și previne utilizarea acestei muchii atunci când există alte posibilități. O muchie care apare numai în propriul său graf va avea un cost egal cu 0.

Acest algoritm se poate utiliza și pentru rutarea altor arhitecturi, dacă se izolează etapa de construire a grafului de conectivitate, care este dependentă de structura circuitului. Algoritmul a fost implementat în limbajul C, fiind inclus în cadrul sistemului CAD realizat pentru proiectarea cu circuitele FPGA *Atmel*.

5.9 Concluzii

În acest capitol a fost prezentatã problema de rutare a circuitelor VLSI şi FPGA, şi a fost propus un algoritm de rutare pentru circuitele FPGA cu resurse limitate de rutare, în particular pentru circuitul FPGA *Atmel*.

Rutarea se descompune de obicei în douã etape: *rutarea globalã* şi *rutarea detaliatã*. În cadrul rutării globale se elaborează un plan de rutare astfel încât fiecare conexiune sã fie asignatã unor regiuni particulare de rutare. Rutarea detaliatã se aplicã apoi pentru fiecare regiune de rutare, şi fiecãrei conexiuni i se asignează piste particulare de rutare. Au fost prezentate funcøiile de cost şi restricøiile pentru rutarea globalã şi cea detaliatã, pentru diferite tipuri de circuite.

Problema de *rutare globalã* este formulatã în mod diferit pentru diferite tipuri de circuite. În cazul reøelelor de porøi, regiunile de rutare constau din canale orizontale şi verticale cu capacitate fixã. Deoarece reøeaua de porøi are o dimensiune fixã şi un spaøiu de rutare fix, obiectivul rutării globale este nu numai elaborarea unui plan de rutare, ci şi testarea fezabilitãøii rutării detaliate.

În cazul celulelor standard, regiunile de rutare sunt canale orizontale a cãror capacitate nu este fixatã dinainte. Numãrul pistelor de rutare poate fi deci extins pentru a se asigura rutabilitatea. Rutarea globalã constã în asignarea conexiunilor pentru aceste canale astfel încât sã se minimizeze congestia acestora şi lungimea totalã a conexiunilor.

Pentru circuitele cu macro-celule, celulele au forme şi dimensiuni diferite, ceea ce conduce la regiuni de rutare neregulate. Identificarea acestor regiuni este o etapã importantã a rutării globale. Ca şi în cazul celulelor standard, regiunile de rutare nu au capacitãøi predefinite.

Existã diferite metode de rutare globalã: metode secvenøiale, metode aleatoare, metoda programării liniare şi metoda descompunerii ierarhice. Dintre acestea, metoda programării liniare combinatã cu cea a descompunerii ierarhice a fost raportatã ca având rezultatele cele mai bune. Au fost prezentate diferite metode de rutare globalã: metoda parcurgerii labirintului, o metodã de rutare globalã orientatã pe performanøe bazatã arbori de rutare cu razã limitatã, metoda de cãlire simulatã, metoda programării întregi. Dintre metodele prezentate, cea bazatã pe arbori de rutare cu razã limitatã permite obøinerea celor mai bune rezultate.

Au fost prezentate douã metode de *rutare detaliatã* generalã: metoda parcurgerii labirintului şi metoda cãutării liniilor. Algoritmii de parcurgere a labirintului garantează gãsirea cãii celei mai scurte dacã o asemenea cale existã. Din aceastã categorie a fost descris algoritmul Lee, care necesitã însã un timp de execuøie ridicat şi un necesar ridicat de memorie. Algoritmii de cãutare a liniilor eliminã aceste dezavantaje ale algoritmului Lee. Acešti algoritmi garantează gãsirea unei cãi dacã o asemenea cale existã, nu neapãrat cea mai scurtã.

Dintre metodele de rutare prin canale, a fost descris algoritmul marginii din stânga şi doi algoritmi elaboraøi de Yoshimura şi Kuh. Primul algoritm utilizează fuzionarea nodurilor astfel încât dupã fuzionare lungimea cãii celei mai lungi este minimizatã. Al doilea algoritm minimizează lungimea cãii celei mai lungi prin tehnici de potrivire într-un graf bipartit. Au fost prezentate de asemenea alte metode de rutare prin canale: o euristicã de tip greedy, o metodã de rutare bazatã pe permutări şi o metodã bazatã pe sortare.

A fost definitã problema de rutare pentru circuitele FPGA, punându-se în evidenøã modul în care aceastã problemã este diferitã de problema de rutare a circuitelor VLSI. Conflictele pentru resursele de rutare fixe reprezintã motivul principal pentru care rutarea circuitelor FPGA poate fi mai dificilã decât rutarea clasicã. Metodele obişnuite de rutare globalã sau detaliatã nu sunt adecvate pentru circuitele FPGA. De aceea, sunt necesare metode specifice pentru rutarea acestor circuite. În literaturã a

fost publicat un numãr redus de programe de rutare pentru circuitele FPGA. Au fost prezentate douã din acestea: rutarea prin expandarea grafului şi rutarea bazatã pe grafuri cu ponderi multiple.

Algoritmul de rutare prin expandarea grafului constã din douã faze. În prima fazã se memoreazã un numãr de alternative pentru fiecare cale detaliatã a grafurilor grosiere, iar apoi în faza a doua se efectueazã alegeri specifice pentru fiecare conexiune. În cazul rutării pe baza grafurilor cu ponderi multiple, fiecãrei muchii din graf i se asociazã un numãr de ponderi distincte, care reprezintã diferitele criterii de optimizare. Algoritmul descris care utilizeazã aceastã metodã este un algoritm hibrid, care combinã euristica geometricã 1-Steiner cu algoritmul Kou, Markowsky, Berman de aproximare a arborilor Steiner. Astfel se permite modelarea arhitecturii circuitului FPGA ca un graf şi utilizarea în acelaşi timp a informaţiilor geometrice despre circuit în scopul obţinerii unor soluţii îmbunãtãţite de rutare.

În acest capitol s-a descris un algoritm de rutare care a fost conceput şi implementat pentru circuitele FPGA *Atmel*. Algoritmul executã simultan rutarea globalã şi cea detaliatã. Un avantaj al acestei metode este cã estimarea preliminarã a rutării globale poate fi imediat corectatã. De asemenea, algoritmul poate lua în considerare efectele secundare pe care le au deciziile de rutare luate pentru o conexiune asupra celorlalte conexiuni, rezolvând astfel conflictele de rutare. Algoritmul poate efectua douã tipuri de optimizări: din punct de vedere al spaţiului ocupat şi din punct de vedere al vitezei. Pentru optimizarea din punctul de vedere al spaţiului, algoritmul sortează conexiunile dupã numărul alternativelor posibile de rutare. Pentru optimizarea din punctul de vedere al vitezei, conexiunile sunt sortate dupã lungimea lor.

Contribuţia principalã a acestui capitol este elaborarea şi implementarea unui algoritm de rutare pentru circuitele FPGA *Atmel*. Algoritmul are urmãtoarele avantaje:

- Executã simultan rutarea globalã şi cea detaliatã, nefiind necesarã împãrţirea problemei de rutare în douã subprobleme, împãrţire care conduce de obicei la rezultate globale care nu sunt optime. De asemenea, prin aceastã abordare estimarea preliminarã a rutării globale poate fi corectatã imediat.
- Algoritmul poate lua în considerare efectele secundare pe care le au deciziile de rutare luate pentru o conexiune asupra celorlalte conexiuni, rezolvând astfel conflictele de rutare.
- Algoritmul poate efectua douã tipuri de optimizări: din punct de vedere al spaţiului ocupat şi din punct de vedere al vitezei.
- Algoritmul ţine cont de aspectele specifice ale arhitecturii circuitelor FPGA *Atmel*, în scopul creşterii eficienţei.