3. PARTIȚIONAREA PENTRU CIRCUITELE CU RESURSE LIMITATE DE RUTARE

3.1 Introducere

Partiøionarea este tehnica de divizare a unui circuit sau sistem într-o colecøie de pãrøi de dimensiune mai micã (componente). Pe de o parte, partiøionarea este o etapã de proiectare pentru divizarea unui sistem în mai multe pãrøi care pot fi implementate prin componente separate, iar pe de altã parte partiøionarea reprezintã o metodã algoritmicã pentru rezolvarea problemelor complexe de optimizare care apar în sinteza logicã sau în proiectarea fizicã a circuitelor VLSI. În literaturã a fost publicat un numãr mare de lucrãri care trateazã problema de partiøionare a grafurilor sau a circuitelor [14], [23], [31], [33], [39], [40], [41], [49], [50], [51], [69], [70], [76], [85], [86], [89], [90], [94], [95], [97], [98], [105], [106], [116], [129], [130], [138], [139], [146], [155], [156], [168], [175], [176], [180], [181], [183], [186], [189].

Existã aplicaøii ale partiøionãrii la toate nivelele de abstractizare, de exemplu la nivel funcøional şi la nivel structural [98]. În primele etape ale procesului de proiectare, trebuie luate decizii de partiøionare a sistemului, bazate adeseori pe cunoştinøe incomplete. În particular, trebuie sã se decidã dacã o componentã va fi implementatã prin hardware sau prin software, pentru a se obøine un raport optim dimensiune/performanøã. Deoarece partiøionarea complet automatã este esenøialã pentru iteraøii rapide în ciclul de proiectare, existã eforturi considerabile pentru a facilita şi a îmbunãtãøi deciziile dificile la nivel funcøional.

Componentele care rezultã din partiøionarea sistemului sunt implementate de proiectanøi sau sunt sintetizate dintr-o descriere de nivel înalt prin utilizarea unor utilitare de sintezã care genereazã o implementare structuralã. În cazul în care componentele hardware au o complexitate prea mare datoritã restricøiilor de spaøiu sau a numãrului de terminale, acestea sunt partiøionate din nou la nivel structural pe baza unor obiecte ca module sau celule. Subiectul partiøionãrii la nivel structural este studiat extensiv în literaturã.

Pe mãsura creşterii complexitãøii sistemelor digitale, este necesarã partiøionarea acestora pentru simplificarea procesului de proiectare şi sintezã. Aceastã descompunere a problemei de sintezã este reflectatã în organizarea ierarhicã a plãcilor, modulelor multi-cip, circuitelor integrate şi a macro celulelor. La nivelele inferioare ale ierarhiei de proiectare, în general întârzierile semnalelor scad; de exemplu, comunicaøia în cadrul unui circuit integrat este mai rapidã decât comunicaøia între douã circuite. De aceea, metrica tradiøionalã pentru partiøionare este numãrul de conexiuni sau semnale existente între partiøii. Esenøa partiøionãrii o reprezintã minimizarea acestui numãr.

Orice decizie luatã în primele etape a procesului de proiectare va influenøa deciziile ulterioare. De aceea, soluøiile problemelor de plasare, rutare globalã şi rutare detaliatã depind de calitatea algoritmului de partiøionare. Dupã cum au arãtat autori ca

Wei şi Cheng [175], Hagen şi Kahng [86], sau Johannes [98], partiøionarea este importantã pentru numeroase probleme fundamentale CAD, dintre care se amintesc urmãtoarele:

- *Încapsularea circuitelor*: Logica este partiøionatã în blocuri, øinând cont de limitãrile numãrului pinilor de I/E şi de restricøiile suprafeøei unui bloc; aceastã partiøionare se efectueazã la fiecare îmbunãtãøire a tehnologiei, atunci când circuitele existente trebuie reîncapsulate în blocuri de capacitate mai mare.
- *Plasare*: Partiøionarea este utilizatã în acest caz pentru a obøine o listã de conexiuni grupatã care este utilizatã apoi pentru plasarea constructivã a modulelor.
- *Sinteza de nivel înalt*: Predicøia cu acurateøe a suprafeøei de amplasare a modulelor şi a conectivitãøii este importantã pentru sinteza de nivel înalt; modelele de predicøie se bazeazã pe analiza structurii de partiøionare a listelor de conexiuni împreunã cu modele pentru algoritmii de plasare şi rutare.
- *Simulare hardware şi test*: O partiøionare de calitate va minimiza numãrul semnalelor inter-blocuri care trebuie multiplexate de un simulator hardware; similar, reducerea numãrului de intrãri la un bloc va reduce adesea şi numãrul vectorilor necesari pentru simularea logicii.
- *Prototipizare rapidã bazatã pe circuite FPGA*: Emularea sistemelor şi prototipizarea rapidã bazatã pe reøele de circuite FPGA este din ce în ce mai rãspânditã. Pentru a obøine cel mai scurt ciclu de proiectare, partiøionarea automatã a sistemului este absolut necesarã. Deoarece numãrul circuitelor FPGA, tipul lor şi interconectarea acestora sunt date, partiøionarea constã în a gãsi o mapare a obiectelor sistemului la circuitele FPGA, cu satisfacerea unor restricøii ca numãrul blocurilor logice ale circuitelor, numãrul de pini, sau întârzierile unor cãi critice. Partiøionarea pentru circuite FPGA multiple este una din aplicaøiile cele mai des studiate în articolele recente [23], [90], [95], [179], [181].
- *Amplasarea componentelor*: Aceastã problemã este în strânsã legãturã cu partiøionarea. Dacã este datã partiøionarea sistemului în blocuri, sarcina amplasãrii este de a determina poziøiile relative ale blocurilor, dimensiunile lor şi poziøia pinilor, pentru a se optimiza spaøiul ocupat în cadrul circuitului, respectându-se restricøiile de întârziere a semnalelor. Amplasarea optimã depinde în mare mãsurã de calitatea partiøionãrii. Problema amplasãrii este dificilã deoarece este posibil ca unele pãrøi ale circuitului sã nu fie complet specificate sau implementate, ceea ce face ca estimarea cu acurateøe a parametrilor unui bloc sã fie esenøialã.
- *Partiøionarea algoritmilor DSP*: Partiøionarea algoritmilor procesoarelor de semnal pentru sistemele multi-procesor este dificilã din cauza numeroaselor restricøii. Programarea procesoarelor de semnal este realizatã prin maparea unei specificaøii funcøionale pe o reøea datã de procesoare. Este esenøial ca aceastã operaøie sã fie realizatã în mod automat pentru a se putea investiga rapid diferite arhitecturi şi soluøii.
- *Co-proiectare hardware/software*: Sistemele microelectronice constau de obicei din pãrøi hardware specifice aplicaøiei şi pãrøi software. Componentele hardware asigurã performanøe mai ridicate, având însã şi costuri mai ridicate, în timp ce componentele software sunt mai flexibile şi mai puøin costisitoare. Un proiectant de sistem va realiza o partiøionare în componente hardware şi software care satisface toate restricøiile de performanøã, menøinând în acelaşi timp costul componentelor hardware la minimul necesar.

3.2 Definirea problemei de partiționare

Problema de partiøionare a circuitelor se poate formula ca o problemã de partiøionare a grafurilor. Un model matematic standard al circuitelor asociazã un graf $G = (V, E)$ cu lista de conexiuni a circuitului, unde vârfurile din V reprezintã elemente de circuit (module), iar muchiile din E reprezintã interconexiuni. Vârfurile și muchiile grafului G pot fi ponderate pentru a reflecta suprafaøa ocupatã de modul sau importanøa unei conexiuni. Dacã se considerã circuitul din Figura 3.1(a), modelul sub forma unui graf al acestui circuit este indicat în Figura 3.1(b). Se observã cã toate interconexiunile sunt conexiuni cu doi pini. De exemplu, conexiunea pinilor de intrare ale porøilor 5 și 6 este modelatã ca o muchie între nodurile 5 și 6 în Figura 3.1(b).

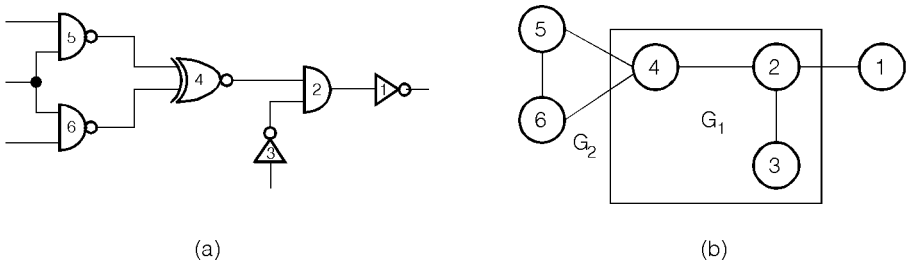

Figura 3.1. (a) Exemplu de circuit. (b) Graful corespunzător circuitului.

Există două formulări de bazã ale problemei de bipartiøionare a circuitelor. Acestea sunt următoarele [86]:

- *Tãietura minimã*: Fiind dat graful $G = (V, E)$, se partiøionează V în subseturile disjuncte U și W astfel încât $e(U, W)$, adicã numărul muchiilor în $\{(u, w) \in E \mid u \in U \text{ și } w \in W\}$, este minimizat.
- *Bisecøia de lãøime minimã*: Fiind dat graful $G = (V, E)$, se partiøionează V în subseturile disjuncte U și W, cu $|U| = |W|$, astfel încât $e(U, W)$ este minimizat.

Deoarece bisecøia de lãøime minimã împarte modulele astfel încât numărul acestora este egal în cele două partiøii, acesta reprezintã un obiectiv mai des utilizat pentru circuitele reale.

Problema mai generalã de partiøionare este cea în care se formează k subseturi disjuncte. Aceasta se numește partiøionare cu k cãi și este definită astfel [146]:

- Fiind dat un graf $G = (V, E)$, unde fiecare vârf $v \in V$ are o dimensiune $s(v)$, iar fiecare muchie $e \in E$ are o pondere $w(e)$, se divide setul V în k subseturi V_1, V_2, …, V_k, astfel încât sã se optimizeze o funcøie obiectiv, øinând cont de anumite restricøii.

  Restricøiile sunt analizate în secøiunea 3.3.

Deoarece listele de conexiuni au adesea mai mult de doi pini, o listă de conexiuni poate fi reprezentatã mai general printr-un *hipergraf* $H = (V, E')$, unde hipermuchiile din E' sunt subseturile lui V conøinute de fiecare conexiune. O mare parte din literaturã tratează partiøionarea grafurilor în loc de partiøionarea hipergrafurilor, deoarece formularea este mai simplă, și un număr mare de algoritmi sunt aplicabili numai pentru grafuri.

3.3 Restricții

În cadrul modelului sub formã de graf, dimensiunea *s*(*v*) a unui nod *v* reprezintã suprafaøa elementului de circuit corespunzãtor. Dacã presupunem cã circuitul este partiøionat în *k* subcircuite, partiøionarea împarte graful *G* = (*V*, *E*) în *k* subgrafuri G_i = (v_i, E_i), *i* = 1,2,…, *k*. În Figura 3.1(a), dacã circuitul se partiøioneazã în douã subcircuite, cu porøile 2, 3, 4 într-o partiøie øi porøile 1, 5, 6 în cealaltã, cele douã subgrafuri sunt indicate în Figura 3.1(b). Subgraful G_1 constã din nodurile 2, 3, 4 øi muchiile (2, 4) øi (2, 3), iar subgraful G_2 constã din nodurile 1, 5, 6 øi muchia (5, 6). Muchiile (5, 4), (1, 2) øi (4, 6) sunt 'tãiate' de partiøie. Numele *set de tãieturã* este utilizat pentru a descrie setul acestor muchii. Setul de tãieturã al unei partiøii este indicat prin ψ øi este egal cu setul muchiilor tãiate de partiøie.

Fiind dat un set de noduri *V* interconectate prin muchiile e_1, …, e_n, existã numeroase restricøii care pot fi impuse unei probleme de partiøionare. De exemplu, se poate specifica gãsirea unei partiøii (V_1, …, V_k) care satisface una sau mai multe din urmãtoarele restricøii:

1. Se specificã o dimensiune *s*(*v*) pentru fiecare $v \in V$ øi o limitã superioarã de dimensiune *S*, cerându-se ca dimensiunea fiecãrui nod V_i sã fie cel mult *S*.
2. Sunt specificate valorile întregi pozitive n_1, …, n_k, øi se cere ca numãrul de elemente din V_i sã fie n_i, deci $|V_i| = n_i$, pentru fiecare i = 1, …, *k*.
3. Se specificã o limitã *P* a numãrului de pini, øi se cere ca numãrul de pini din fiecare V_i sã fie cel mult *P*, deci $P_i \le P$, pentru fiecare i = 1, …, *k*.
4. Trebuie minimizatã valoarea tãieturii totale a tuturor muchiilor, øi anume $\sum_{i=1}^{n} c(e_i)$. Cantitatea $\sum_{i=1}^{n} c(e_i)$ este numitã øi dimensiunea tãieturii partiøiei. Pentru o bipartiøie, dimensiunea tãieturii este egalã cu numãrul de conexiuni tãiate de partiøie.
5. Pentru fiecare muchie e_i este specificatã o pondere $w(e_i)$, cerându-se ca valoarea tãieturii totale ponderate a tuturor muchiilor, deci $\sum_{i=1}^{n} w(e_i)c(e_i)$, sã fie minimizatã. Cantitatea $\sum_{i=1}^{n} w(e_i)c(e_i)$ este numitã øi dimensiunea ponderatã a tãieturii partiøiei.
6. Se dã o submuløime *TEST* a muchiilor care sunt desemnate ca *testabile*, øi se cere ca fiecare muchie din *TEST* sã fie externã sau tãiatã de partiøie, deci *c*(*e*) > 0 pentru fiecare $e \in TEST$.
7. Un set *CRIT* al muchiilor care sunt desemnate ca fiind *critice* trebuie sã fie intern sau netãiat de partiøie, deci *c*(*e*) = 0 pentru fiecare $e \in CRIT$.

În cazul problemei generale de partiøionare cu *k* cãi, restricøia de dimensiune este exprimatã prin plasarea unei limite superioare asupra dimensiunii fiecãrui subcircuit. Dimensiunea subcircuitului *i* este datã prin $\sum_{v \in V_i} s(v)$. Dacã limita superioarã a dimensiunii acestui subcircuit este S_i, avem:

$$\sum_{v \in V_i} s(v) \le S_i \tag{3.1}$$

Este de dorit ca circuitul sã fie împãrøit în partiøii de dimensiuni aproximativ egale. Aceasta se poate exprima prin modificarea ecuaøiei 3.1 dupã cum urmeazã:

$$|V_i| = \sum_{v \in V} s(v) \leq \left\lceil \frac{1}{k} \sum_{v \in V} s(v) \right\rceil = \frac{1}{k}|V| \tag{3.2}$$

unde $|V_i|$ şi $|V|$ reprezintã dimensiunile seturilor V_i, respectiv V. Dacã toate elementele de circuit au aceeaşi dimensiune, ecuaøia 3.2 se reduce la:

$$n_i \leq \frac{n}{k} \tag{3.3}$$

unde n_i şi n reprezintã numãrul elementelor din V_i, respectiv V.

Dacã subcircuitele sunt implementate în capsule separate, sunt necesare legãturi externe între acestea. În particular, conexiunile care aparøin setului de tãieturã vor fi implementate ca şi legãturi externe. Legãturile externe nu sunt de dorit, deoarece introduc întârzieri suplimentare. De aceea este necesarã minimizarea acestor legãturi externe. Ponderea $w(e)$ a unei muchii e a grafului circuitului reprezintã costul implementãrii conexiunii respective ca o legãturã externã. Astfel, funcøia de cost care trebuie minimizatã în timpul partiøionãrii este:

$$Cost = \sum_{\psi} w(e) \tag{3.4}$$

Presupunem cã partiøiile sunt numerotate 1, 2, …, k. Fie $p(u)$ numãrul partiøiei nodului u. Condiøia $e \in \psi$ poate fi scrisã ca $e = (u, v)$, şi $p(u) \neq p(v)$. Astfel, ecuaøia 3.4 poate fi rescrisã ca:

$$Cost = \sum_{\forall e=(u,v) \& p(u) \neq p(v)} w(e) \tag{3.5}$$

3.4 Prezentarea sintetică a metodelor de partiționare

Problema de partiøionare, dupã cum este formulatã în secøiunea 3.2, este o problemã intractabilã [146]. Chiar şi cazul cel mai simplu al problemei, şi anume bipartiøionarea cu dimensiuni egale ale nodurilor şi ponderi unitare ale muchiilor, este o problemã NP-completã. Pentru un circuit cu $2n$ noduri, numãrul partiøiilor echilibrate creşte exponenøial cu n. Chiar pentru valori moderate ale lui n, este impracticã enumerarea tuturor partiøiilor şi alegerea celei mai bune. Singurul mod de a rezolva asemenea probleme NP-complete este de a obøine soluøii aproximative. O asemenea soluøie trebuie sã satisfacã restricøiile cerute, dar costul obøinut nu este neapãrat cel minim.

Existã diferite tehnici euristice pentru a se genera soluøii aproximative ale problemei de partiøionare [98], [146]. Acestea se pot clasifica în algoritmi *deterministici* şi *stohastici* (probabilistici). Un algoritm deterministic generează aceeaşi soluøie la fiecare execuøie, deoarece deciziile pe care le ia sunt deterministice. O metodã stohasticã se bazează pe decizii aleatoare pentru generarea soluøiei, astfel cã o asemenea metodã generează soluøii diferite pentru aceleaşi intrãri.

Metodele de partiøionare pot fi clasificate de asemenea ca fiind *constructive* şi *iterative*. O metodã constructivã porneşte de la o anumitã componentã nucleu (sau mai multe asemenea componente), selectând apoi alte componente pentru a fi adãugate la soluøia parøialã, pânã la obøinerea unei soluøii complete. Odatã ce o componentã este selectatã pentru a aparøine unei partiøii, ea nu mai este mutatã în etapele urmãtoare ale procedurii de partiøionare. O metodã iterativã are ca scop îmbunãtãøirea calitãøii unei soluøii existente de partiøionare, de exemplu reducerea costului. De multe ori, o asemenea metodã se aplicã pentru îmbunãtãøirea soluøiei generate de o metodã

constructivã. De obicei metodele constructive sunt deterministice, în timp ce metodele iterative pot fi deterministice sau stohastice.

Metodele constructive de partiøionare se bazeazã în principal pe grupare, metode spectrale sau vectori proprii, partiøionare bazatã pe plasare, programare matematicã, sau calcule ale fluxului în reøele.

Gruparea este o tehnicã pentru a determina componentele puternic conectate ale unui graf. Pentru partiøionarea unor circuite conøinând milioane de module gruparea de jos în sus este combinatã adesea cu partiøionarea de sus în jos. O formulare care unificã ambele strategii a fost publicatã în [94]. Gruparea a fost aplicatã de asemenea pentru optimizarea performanøelor [136], [179], [184]. Structuri formate din setul tuturor nodurilor unui bloc combinaøional între o singurã ieşire şi intrãrile care conduc la aceastã ieşire, au fost aplicate la partiøionarea de jos în sus a circuitelor FPGA pentru cãi critice [23]. În acest caz, considerarea direcøiei semnalelor a determinat îmbunãtãøirea soluøiei de partiøionare. Compromisul între timpul de execuøie şi performanøã este investigat în [185]. În [109] se descrie o metodã de grupare în care informaøiile globale de conectivitate ale grafului sunt obøinute din proprietatea de grupare a metodei vectorilor proprii.

Tehnicile de programare matematicã sunt utilizate pentru optimizarea unei funcøii obiectiv sub restricøiile unor inegalitãøi. Pentru rezolvarea problemelor de partiøionare au fost aplicate programarea cuadraticã [139], programarea booleanã cuadraticã [155], programarea liniarã [116].

Metodele spectrale au fost propuse în ultimii ani [39], [40], [86], [109]. Pe baza matricii de adiacenøã a grafului, obiectivul tãieturii minime poate fi rescris ca un sistem de ecuaøii. Vectorul propriu al valorii proprii minime diferite de zero a matricii poate fi interpretat ca o plasare liniarã sau ordonare a nodurilor grafului. Aceastã ordonare poate fi divizatã pentru a obøine o partiøionare a nodurilor. În literaturã au fost publicate numeroase modificãri ale acestei metode de bazã, inclusiv utilizarea mai multor vectori proprii. În cazul partiøionãrii cu cãi multiple s-a demonstrat faptul cã odatã cu creşterea numãrului de vectori proprii, creşte calitatea partiøionãrii.

Partiøionarea bazatã pe plasare este strâns legatã de metodele spectrale. Aceste metode minimizeazã o funcøie obiectiv cuadraticã. Pentru plasare, s-a arãtat cã minimizarea unui obiectiv liniar conduce la rezultate îmbunãtãøite. În [139] aceastã observaøie a fost utilizatã pentru obøinerea unor partiøionãri de calitate mai bunã. Deoarece plasarea cu un obiectiv liniar este derivatã din plasarea bazatã pe vectori proprii, aceastã metodã poate fi clasificatã şi ca o metodã de îmbunãtãøire iterativã. Aceastã metodã bazatã pe plasare a fost de asemenea extinsã la partiøionarea cu cãi multiple cu aplicaøii la circuitele FPGA [138].

În cazul metodelor bazate pe fluxul în reøele, fluxul direcøionat al semnalelor poate fi utilizat pentru îmbunãtãøirea performanøelor sistemului. Au fost propuse diferite tipuri de formulãri ale fluxului în reøele [67], [94], [95], [116], [180], [181], [184]. Toate acestea au în comun faptul cã este generat un model al grafului din lista de conexiuni direcøionatã pentru a determina un flux maxim care este echivalent cu o tãieturã minimã.

Metodele constructive stohastice nu au fost utilizate frecvent pentru soluøionarea problemelor de partiøionare. Exemple mai recente sunt [90] şi [184]. În [90] este determinatã o odonare liniarã prin selectarea aleatoare a unor noduri de început. Prin utilizarea programãrii dinamice ordonarea este divizatã în grupe. În [184] este propusã o metodã probabilisticã pentru a reduce complexitatea computaøionalã a algoritmului bazat pe flux.

Au fost publicate numeroase metode deterministice iterative. Acestea interschimbã în mod iterativ noduri sau perechi de noduri pentru a minimiza numãrul de muchii tãiate. Din acest motiv, acestea sunt indicate în mod colectiv ca algoritmi de tãieturã minimã. Cele mai multe din acestea sunt de tip greedy [105], [146], [189]. Acešti algoritmi diferã în mod semnificativ în alegerea funcøiei obiectiv utilizate. Un obiectiv

îmbunãtãøit bazat pe calculul probabilistic al câştigurilor este introdus în [69]. Cele mai multe implementãri utilizează configuraøii iniøiale aleatoare multiple [175] pentru cãutarea adecvatã în spaøiul soluøiilor şi a obøine o anumitã mãsurã de "stabilitate", deci, de performanøe predictibile. În [50] a fost propusã o metodã de bipartiøionare cu performanøe stabile, care nu necesitã generarea unui mare numãr de configuraøii iniøiale. Se utilizează o tehnicã de partiøionare recursivã de sus în jos, şi se divide întregul circuit în grupuri mici, puternic conectate, care sunt apoi rearanjate în douã subseturi care respectã restricøia de dimensiune.

Îmbunãtãøirea iterativã poate fi combinatã cu gruparea pentru a reduce complexitatea calculelor. Deoarece metodele iterative deterministice sunt sensibile la modul de alegere a partiøiei iniøiale, în [117] a fost propusã o metodã bazatã pe gradient pentru a elimina acest dezavantaj. Au fost propuse şi metode de partiøionare de tãieturã minimã bazate pe programarea cuadraticã.

Metodele stohastice de îmbunãtãøire iterativã utilizate în mod obişnuit sunt cãlirea simulatã şi evoluøia simulatã [76], [189]. Cel mai important avantaj al acestora este cã pot evita minimele locale. O evaluare experimentalã a bipartiøionãrii în [186] aratã cã prin cãlirea simulatã se obøin anumite avantaje în privinøa calitãøii soluøiei, dar timpul de calcul consumat este foarte mare.

Ca exemple de metode deterministice vor fi prezentate euristicile Kernighan-Lin şi Fiduccia-Mattheyses în secøiunile 3.4.1, respectiv 3.4.3. Variante ale euristicii Kernighan-Lin sunt prezentate în secøiunea 3.4.2. Euristica stohasticã de cãlire simulatã este prezentatã în secøiunea 3.4.4. În secøiunea 3.4.5 este descrisã metoda de partiøionare care utilizează tãietura proporøionalã, o metricã propusã de Wei şi Cheng, care s-a dovedit o funcøie obiectiv de succes pentru numeroase aplicaøii. O metodã de partiøionare cu performanøe stabile, care nu depind de alegerea partiøiei iniøiale, este descrisã în secøiunea 3.4.6. Metodele spectrale de partiøionare, care utilizează valori proprii şi vectori proprii ale matricilor obøinute din graful circuitului, sunt prezentate în secøiunea 3.4.7. În secøiunea 3.4.8 se descrie o metodã pentru modelarea unei liste de conexiuni printr-o reøea de flux, şi o euristicã de bipartiøionare echilibratã bazatã pe utilizarea repetatã a tehnicii fluxului maxim şi tãieturii minime. Un algoritm de multipartiøionare cu îmbunãtãøire iterativã, care poate utiliza diferite funcøii obiectiv cu aplicaøii în proiectarea circuitelor VLSI, este prezentat în secøiunea 3.4.9. În sfârşit, partiøionarea prin metode probabilistice este prezentatã în secøiunea 3.4.10.

3.4.1 Algoritmul Kernighan-Lin

Acest algoritm este unul din cele mai utilizate pentru rezolvarea problemei de bipartiøionare. Algoritmul Kernighan-Lin (KL) a fost elaborat iniøial pentru bisecøionarea grafurilor, şi a fost extins apoi pentru rezolvarea problemei de bisecøionare a circuitelor [189].

Problema de bipartiøionare este caracterizatã printr-o matrice de conectivitate C, care este o matrice pãtratã cu un numãr de linii egal cu numãrul de noduri ale grafului circuitului. Elementul c_{ij} reprezintã suma ponderilor muchiilor care conectează elementele i şi j. În cazul în care muchiile au ponderi unitare, c_{ij} indicã numãrul de muchii care conectează i şi j. Rezultatul algoritmului de partiøionare este o pereche de seturi (blocuri) A şi B astfel încât $|A| = |B| = n$, $A \cap B = \emptyset$, iar setul de tãieturã are o dimensiune cât mai micã. Această dimensiune este mãsuratã prin T,

$$T = \sum_{a \in A, b \in B} c_{ab} \tag{3.6}$$

Algoritmul porneşte de la o partiøie iniøialã astfel încât $|A| = |B| = n$ şi $A \cap B = \emptyset$. Fie $P^* = \{A^*, B^*\}$ partiøia optimã, iar $P = \{A, B\}$ partiøia curentã. Pentru a se obøine P^* din P, trebuie sã se interschimbe un subset $X \subseteq A$ cu un subset $Y \subseteq B$, astfel încât:

(1) $|X| = |Y|$
(2) $X = A \cap B^*$
(3) $Y = A^* \cap B$

Aceastã interschimbare este ilustratã în Figura 3.2.

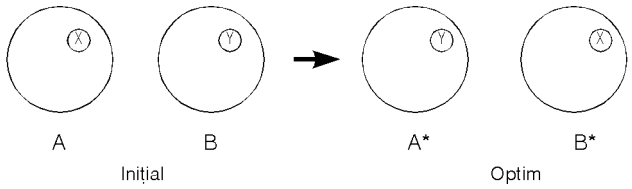

Figura 3.2. Partiøia iniøialã øi cea optimã.

Problema identificãrii subseturilor *X* øi *Y* este la fel de dificilã cu cea a determinãrii partiøiei $P^* = \{A^*, B^*\}$. Kernighan øi Lin au propus o metodã euristicã pentru a aproxima *X* øi *Y*. Se va analiza mai întâi efectul interschimbãrii unui singur nod din blocul *A* cu un alt nod din blocul *B*.

Considerãm un nod oarecare *a* din blocul *A*. Contribuøia nodului *a* la setul de tãieturã se numeøte costul extern al lui *a*, sau E_a, øi este numãrul muchiilor dintre nodul $a \in A$ øi care se terminã în *B*:

$$E_a = \sum_{v \in B} c_{av} \tag{3.7}$$

Similar, se poate defini costul intern I_a al nodului $a \in A$ ca fiind:

$$I_a = \sum_{v \in A} c_{av} \tag{3.8}$$

Dacã nodul *a* se mutã din blocul *A* în blocul *B*, dimensiunea setului de tãieturã va creøte cu o valoare I_a øi va scãdea cu o valoare E_a. Câøtigul în urma mutãrii este deci $E_a - I_a$, valoare care se numeøte valoare *D* a nodului *a*:

$$D_a = E_a - I_a \tag{3.9}$$

Efectul interschimbãrii a douã noduri între blocurile *A* øi *B* este caracterizat de lema 3.4.1.1 [146].

Lema 3.4.1.1. Dacã douã elemente $a \in A$ øi $b \in B$ sunt interschimbate, reducerea costului este datã de:

$$g_{ab} = D_a + D_b - 2c_{ab} \tag{3.10}$$

Interschimbarea a douã noduri afecteazã valorile *D* ale tuturor nodurilor care sunt conectate la oricare din nodurile interschimbate. Lema 3.4.1.2 indicã modul de actualizare a valorilor *D* ale nodurilor rãmase dupã ce douã noduri *a* øi *b* au fost interschimbate.

Lema 3.4.1.2. Dacã douã elemente $a \in A$ øi $b \in B$ sunt interschimbate, noile valori *D*, indicate prin *D'*, sunt date de:

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\} \tag{3.11}$$

$$D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\} \tag{3.12}$$

Presupunem cã existã o partiøie iniøialã {*A*, *B*} de câte *n* elemente fiecare. Kernighan øi Lin au utilizat Lema 1 øi Lema 2 øi au elaborat o procedurã de tip greedy

pentru a identifica două subseturi $X \subseteq A$ şi $Y \subseteq B$, de cardinalităøi egale, astfel încât în urma interschimbării acestora costul partiøiei este îmbunătăøit. X şi Y pot fi vide, indicând în acest caz faptul că partiøia curentă nu mai poate fi îmbunătăøită.

În cursul procedurii, se calculează câştigurile interschimbării oricăror două module $a \in A$ şi $b \in B$. Este selectată perechea (a_1, b_1) care conduce la câştigul maxim g_1, iar elementele a_1 şi b_1 sunt blocate pentru a nu fi luate în considerare la următoarele interschimbări. Valorile D ale celulelor libere rămase sunt actualizate, iar câştigurile sunt recalculate. Apoi este selectată şi blocată a o doua pereche (a_2, b_2) cu câştigul maxim g_2. De notat că g_2 este câştigul interschimbării a_2 cu b_2 după ce a_1 a fost deja interschimbar cu b_1. Astfel, câştigul interschimbării perechii (a_1, b_1) urmat de interschimbarea (a_2, b_2) este $G_2 = g_1 + g_2$. Procesul continuă selectând (a_1, b_1), (a_2, b_2),…,(a_i, b_i) … (a_n, b_n), câştigurile corespunzătoare fiind $g_1, g_2, \ldots, g_i, \ldots, g_n$. Evident, $G = \sum_{i=1}^{n} g_i = 0$, deoarece aceasta presupune interschimbarea tuturor elementelor din A cu cele din B, rezultând partiøia iniøială. În general, câştigul interschimbării primelor k perechi (a_1, b_1), (a_2, b_2),…,(a_k, b_k), $1 \le k \le n$ este $G_k = \sum_{i=1}^{k} g_i$. Dacă nu există un k astfel ca $G_k > 0$ atunci partiøia curentă nu poate fi îmbunătăøită; în caz contrar se alege valoarea k care maximizează G_k şi se efectuează interschimbarea $\{a_1, a_2, \ldots, a_k\}$ cu $\{b_1, b_2, \ldots, b_k\}$ în mod permanent.

Procedura de îmbunătăøire a unei partiøii, indicată mai sus, constituie un singur pas al algoritmului Kernighan-Lin. Partiøia obøinută după pasul i constituie partiøia iniøială pentru pasul $i + 1$. Iteraøiile se termină atunci când $G_k \le 0$, deci nu se mai pot obøine îmbunătăøiri suplimentare prin interschimbarea perechilor. Algoritmul este descris în Figura 3.3.

Algorithm *KL*;
begin
Pas 1. V = setul a $2n$ elemente;
*$\{A, B\}$ este partiția inițială astfel încât
$|A| = |B|$, $A \cap B = \varnothing$ şi $A \cup B = V$;
Pas 2. Se calculează D_v pentru fiecare $v \in V$;
lista $\leftarrow \phi$; $i \leftarrow 1$;
$A' = A$; $B' = B$;
Pas 3. Se alege $a_i \in A'$, $b_i \in B'$ care maximizează $g_i = D_{ai} + D_{bi} - 2c_{aibi}$;
Se adaugă perechea (a_i, b_i) la *lista*;
$A' = A' - \{a_i\}$; $B' = B' - \{b_i\}$;
Pas 4. **if** A' şi B' sunt ambele vide **then goto** Pas 5
else
Se recalculează valorile D pentru $A' \cup B'$;
$i \leftarrow i + 1$; **goto** Pas 3;
endif;
Pas 5. Se determină k pentru a maximiza suma parțială $G = \sum_{i=1}^{n} g_i$;
if $G > 0$ **then**
Se mută $X = \{a_1, \ldots, a_k\}$ în B şi $Y = \{b_1, \ldots, b_k\}$ în A;
goto Pas 2
else STOP
endif
end.

Figura 3.3. Algoritmul Kernighan-Lin pentru bipartiøionare.

Complexitatea algoritmului Kernighan-Lin este $O(pn^2 \log n)$, unde p este numărul de iteraøii ale procedurii de îmbunătăøire. Experimente asupra unor circuite de

dimensiuni mari au indicat faptul cã p nu creşte cu n. Astfel, complexitatea algoritmului este de $O(n^2 \log n)$ [146].

Complexitatea etapei de selecøie a perechilor poate fi îmbunãtãøitã prin parcurgerea listei nesortate de valori D şi selectarea a şi b care maximizează D_a şi D_b. Deoarece aceasta se poate realiza într-un timp liniar, complexitatea algoritmului se reduce la $O(n^2)$. Această metodă este potrivită pentru matrici rare unde probabilitatea ca $c_{ab} > 0$ este redusă. Aceasta reprezintă însă o aproximare a procedurii de selecøie greedy, şi poate genera o soluøie diferită comparativ cu selecøia greedy.

3.4.2 Variante ale algoritmului Kernighan-Lin

Algoritmul Kernighan-Lin poate fi extins pentru a rezolva şi alte cazuri ale problemei de partiøionare. O parte din acestea sunt prezentate în continuare.

Blocuri cu dimensiuni inegale. Pentru partiøionarea unui graf $G = (V, E)$ cu $2n$ vârfuri în două subgrafuri cu dimensiuni inegale n_1 şi n_2, $n_1 + n_2 = 2n$, se poate utiliza procedura următoare:

1. Se divide setul V în două subseturi A şi B, una conøinând $MIN(n_1, n_2)$ vârfuri, iar cealaltă conøinând $MAX(n_1, n_2)$ vârfuri. Această diviziune se poate efectua în mod arbitrar.
2. Se aplicã algoritmul din Figura 3.3 începând de la Pasul 2, dar se restricøionează numărul maxim de vârfuri care pot fi interschimbate într-un pas la $MIN(n_1, n_2)$.

O altă soluøie posibilă a acestei probleme este următoarea. Presupunând că $n_1 < n_2$, se divide V astfel încât există cel puøin n_1 vârfuri în blocul A şi cel mult n_2 vârfuri în blocul B, utilizând procedura de mai jos.

1. Se divide setul V în blocurile A cu n_1 vârfuri şi B cu n_2 vârfuri.
2. Se adaugă $n_2 - n_1$ vârfuri fictive blocului A. Aceste vârfuri nu au conexiuni cu graful original.
3. Se aplicã algoritmul din Figura 3.3 începând de la Pasul 2.
4. Se elimină vârfurile fictive.

Elemente cu dimensiuni inegale. Pentru a genera o bipartiøie a unui graf ale cărui vârfuri au dimensiuni inegale, se poate proceda în modul următor:

1. Se presupune cã elementul cu dimensiune minimă are dimensiunea unitară.
2. Se înlocuieşte fiecare element de dimensiune s cu s vârfuri complet conectate cu muchii de pondere infinită.
3. Se aplicã algoritmul din Figura 3.3.

Partiøionare cu k cãi. Se presupune cã graful are $k \cdot n$ vârfuri, $k > 2$, şi este necesară generarea unei partiøii cu k cãi, fiecare cu n elemente.

1. Se începe cu o partiøie aleatoare de k seturi cu n vârfuri fiecare.
2. Se aplicã procedura de bipartiøionare pentru fiecare pereche de partiøii.

Optimalitatea perechilor de partiøii este doar o condiøie necesară a optimalitãøii în cazul problemei de partiøionare cu k cãi. Uneori va fi necesară o interschimbare complexă de trei sau mai multe elemente din trei sau mai multe subseturi pentru a se obøine soluøia optimă globală. Deoarece există $\binom{k}{2}$ perechi care trebuie luate în considerare, complexitatea pentru un pas al procedurii executate pentru toate perechile

este $\binom{k}{2} n^2 = O(k^2 n^2)$. În general, vor fi necesari mai muløi paši, deoarece atunci când o anumitã pereche de partiøii este optimizatã, optimalitatea acestor partiøii faøã de altele se poate modifica.

3.4.3 Euristica Fiduccia-Mattheyses

Algoritmul Kernighan-Lin partiøioneazã un circuit modelat ca un graf în douã blocuri *A* øi *B* astfel încât costul muchiilor tãiate de partiøie sã fie minimizat. În cazul conexiunilor cu douã puncte, numãrul muchiilor tãiate de partiøie este egal cu numãrul conexiunilor tãiate. În cazul conexiunilor multipunct, situaøia este însã diferitã. Figura 3.4 ilustreazã un circuit øi reprezentarea sa sub forma unui graf. Dacã se partiøioneazã graful în douã blocuri $A = \{1, 2, 3\}$ øi $B = \{4, 5, 6\}$, numãrul muchiilor tãiate este egal cu patru, în timp ce sunt necesare doar trei conexiuni între celulele blocului *A* øi celulele blocului *B*. De aceea, reducerea numãrului de conexiuni tãiate este un obiectiv mai realist decât reducerea numãrului de muchii tãiate.

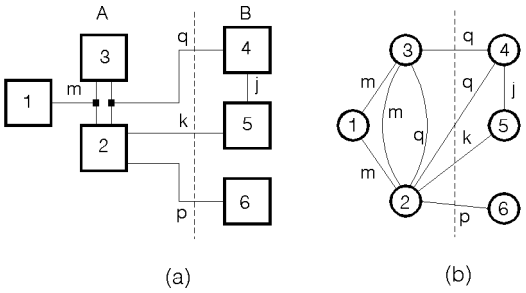

Figura 3.4. (a) Ilustrarea tãieturii conexiunilor. (b) Ilustrarea tãieturii muchiilor.

Fiduccia øi Mattheyses au elaborat o euristicã iterativã care ia în considerare atât conexiunile multipin, cât øi dimensiunile elementelor de circuit. Contribuøia acestora constã în permiterea mutãrii unui singur nod la un moment dat între cele douã subseturi ale partiøiei, o analizã øi o implementare atentã a efectului mutãrii unui singur nod asupra valorilor *D* ale altor noduri, øi utilizarea unei structuri de date eficiente pentru a se evita cãutarea inutilã a nodului care va fi mutat în etapa urmãtoare [189].

Euristica Fiduccia-Mattheyses este o tehnicã utilizatã pentru a gãsi soluøia la urmãtoarea problemã de bipartiøionare: Fiind dat un circuit constând din *C* celule conectate printr-un set de *N* conexiuni, problema este de a partiøiona circuitul *C* în douã blocuri *A* øi *B* astfel încât numãrul conexiunilor care au celule în ambele blocuri este minimizat øi este satisfãcut factorul de echilibru *r* [76]. Se prezintã în continuare principalele diferenøe øi similaritãøi între euristicile Kernighan-Lin øi Fiduccia-Mattheyses [146].

1. Spre deosebire de euristica Kernighan-Lin în care este selectatã pentru interschimbare o pereche de celule, câte una din fiecare bloc, în fiecare pas, în cazul euristicii Fiduccia-Mattheyses este selectatã la un moment dat o singurã celulã, din oricare bloc, pentru a fi mutatã în blocul complementar.
2. Euristica Kernighan-Lin partiøioneazã un graf în douã blocuri astfel încât costul muchiilor tãiate este minim, în timp ce euristica Fiduccia-Mattheyses are ca scop reducerea costului conexiunilor tãiate de partiøie.
3. Euristica Fiduccia-Mattheyses este similarã cu cea Kernighan-Lin în ceea ce priveøte selecøia celulelor. În locul câøtigului datorat interschimbãrii a douã celule este calculat câøtigul datorat mutãrii unei singure celule dintr-un bloc în

altul. Dupã selecøia unei celule, aceasta este blocatã pentru restul pasului respectiv. Numãrul total de celule care sunt mutate este dat de secvenøa cea mai bunã de mutãri c_1, c_2,..., c_k. În cazul euristicii Kernighan-Lin, într-un pas sunt interschimbate primele cele mai bune k perechi.

4. Datoritã faptului cã este mutatã o singurã celulã, poate apare un dezechilibru între cele douã blocuri. De aceea, euristica Fiduccia-Mattheyses generează partiøii echilibrate din punct de vedere al dimensiunii. Factorul de echilibru r este specificat de utilizator øi este definit astfel: $r = \frac{|A|}{|A|+|B|}$, unde $|A|$ øi $|B|$ sunt dimensiunile blocurilor partiøionate A øi B.
5. Anumite celule pot fi blocate iniøial într-una din partiøii.
6. Complexitatea în timp a euristicii Fiduccia-Mattheyses este liniarã. În practicã este necesar un numãr mic de paši, rezultând un algoritm rapid.

Se prezintã în continuare unele definiøii øi termeni utilizaøi pentru descrierea algoritmului.

Fie $p(j)$ numãrul pinilor celulei j, øi fie $s(j)$ dimensiunea celulei j, pentru j = 1, 2,..., C. Dacã V este setul celor C celule, atunci $|V| = \sum_{i=1}^{C} s(i)$.

Starea de tãieturã a unei conexiuni: O conexiune este tãiatã dacã are celule în ambele blocuri, øi este netãiatã în caz contrar. Se utilizează o variabilã pentru a indica starea unei conexiuni.

Setul de tãieturã al partiøiei: Acest set al unei partiøii este cardinalitatea setului tuturor conexiunilor cu starea tãiatã.

Câøtigul celulei: Câøtigul $g(i)$ al unei celule i este numãrul de conexiuni cu care s-ar reduce setul de tãieturã dacã celula i ar fi mutatã.

Criteriu de echilibru: Pentru a se evita migrarea tuturor celulelor într-un bloc, este menøinut un criteriu de echilibru. O partiøie (A, B) este echilibratã dacã:

$$r \times |V| - s_{max} \le |A| \le r \times |V| + s_{max} \qquad (3.13)$$

unde $|A| + |B| = |V|$, iar $s_{max} = Max\,[s(i)]$, $i \in A \cup B = V$.

Celulã de bazã: Celula selectatã pentru mutarea dintr-un bloc în altul este numitã *celulã de bazã*. Aceasta este celula cu câøtigul maxim øi cea a cãrei mutare nu va viola criteriul de echilibru.

Distribuøia unei conexiuni: Distribuøia unei conexiuni n este o pereche ($A(n)$, $B(n)$), unde (A, B) este o partiøie arbitrarã, $A(n)$ este numãrul de celule ale conexiunii n care sunt în A, iar $B(n)$ este numãrul de celule ale conexiunii n care sunt în B.

Conexiune criticã: O conexiune este criticã dacã are o celulã care, dacã este mutatã, va schimba starea sa de tãieturã. Aceasta se întâmplã dacã øi numai dacã $A(n)$ este fie 0 sau 1, sau $B(n)$ este fie 0 sau 1.

Algoritmul este prezentat în Figura 3.5.

Înaintea fiecãrei mutãri, trebuie gãsitã celula cu valoarea D maximã dintr-una din cele douã subseturi ale partiøiei curente. Pentru a evita cãutarea unei asemenea celule, algoritmul FM utilizează o tabelã pentru fiecare subset al partiøiei, a cãrei intrare k este o listã dublu înlãnøuitã de celule din subsetul a cãrei valoare D este k, øi un pointer la intrarea cu valoarea D maximã pentru fiecare din cele douã tabele. În aceste tabele sunt pãstrate numai celulele care nu au fost mutate în pasul curent. Astfel, utilizând aceste tabele, o celulã cu valoarea D maximã din oricare subset al partiøiei poate fi gãsitã într-un timp constant. Pentru a se permite actualizarea celor douã tabele, fiecare celulã are un pointer la locaøia sa din tabela corespunzãtoare. Dacã câøtigul D_i al celulei i s-a modificat din cauza mutãrii unei alte celule, atunci se utili-

zează pointerul celulei *i* pentru a elimina celula *i* din lista sa curentă din tabelă, şi pentru adăugarea în lista care corespunde noului câştig D_i al celulei. Această actualizare necesită numai un timp constant datorită utilizării listelor dublu înlănøuite. Utilizând această structură de date, Fiduccia şi Mattheyses au arătat că timpul de execuøie al unui pas al algoritmului este liniar cu numărul total de pini al circuitului [189].

Algorithm *FM*;
begin
Pas 1. Se calculează câştigul tuturor celulelor;
Pas 2. $i = 1$;
Este selectată celula de bază şi este notată cu c_i;
if nu există celulă de bază **then exit**; **endif**;
O celulă de bază este cea care
(i) are câştigul maxim;
(ii) satisface criteriul de echilibru;
if *egalitate* **then** se utilizează Criteriul de dimensiune sau Numărul conexiunilor interne;
endif;
Pas 3. Se blochează celula c_i;
Se actualizează câştigul celulelor conexiunilor critice afectate;
Pas 4. **if** *celule_libere* $\neq \phi$ **then**
$i = i + 1$;
Este selectată următoarea celulă de bază c_i;
endif;
if $c_i \neq \phi$ **then goto** Pas 3 **endif**;
Pas 5. Se selectează secvenţa cea mai bună de mutări $c_1, c_2, \ldots, c_k$ $(1 \le k \le i)$
astfel încât $G = \sum_{j=1}^{k} g_j$ este maxim;
if $G \le 0$ **then exit**; **endif**;
Pas 6. Toate cele k mutări devin permanente;
Se deblochează toate celulele;
goto Pas 1
end.

Figura 3.5. Algoritmul de bipartiøionare Fiduccia-Mattheyses.

3.4.4 Partiționarea prin metoda călirii simulate

Călirea simulată este o tehnică iterativă utilizată pe scară largă pentru rezolvarea diferitelor probleme combinatoriale de optimizare. A fost utilizată pentru majoritatea problemelor CAD, inclusiv pentru partiøionare. Este o euristică adaptivă care aparøine clasei algoritmilor stohastici. Această euristică a fost introdusă pentru prima dată de Kirkpatrick, Gelatt şi Vecchi [102].

Euristica de călire simulată se inspiră din procesul de răcire controlată a metalelor topite pentru a se obøine o structură cristalină corespunzătoare. Dacă se compară optimizarea cu procesul de călire, se poate realiza o analogie între obøinerea optimului global şi obøinerea structurii cristaline dorite.

Orice problemă de optimizare combinatorială poate fi discutată în termenii unui *spaøiu al stărilor*. O *stare* este o configuraøie a obiectelor combinatoriale implicate. Din numărul mare de configuraøii, numai unele din acestea corespund optimului global.

O metodă de îmbunătăøire iterativă porneşte de la o anumită stare dată, şi examinează o vecinătate locală a stării pentru a căuta soluøii mai bune. O vecinătate locală a unei stări S este setul tuturor stărilor care pot fi obøinute din S prin unele modificări

ale stãrii *S*. De exemplu, dacã *S* reprezintã o bipartiøie a unui graf, setul tuturor partiøiilor care se pot obøine prin interschimbarea a douã noduri din cadrul partiøiei reprezintã o vecinãtate localã.

Metodele iterative de partiøionare prezentate anterior sunt de tip greedy, deci sunt permise numai perturbaøiile care îmbunãtãøesc soluøia curentã. Astfel aceste metode pot obøine soluøii optime locale. Euristica de cãlire simulatã încearcã remedierea acestei situaøii, acceptând şi perturbaøii care conduc la obøinerea unei soluøii inferioare celei curente, existând astfel posibilitatea de a evita obøinerea unei soluøii optime locale.

Algoritmul de cãlire simulatã este prezentat în Figura 3.6.

Partea principalã a algoritmului este procedura *Metropolis*, care simulează procesul de cãlire la o temperaturã datã *T*. Aceastã procedurã constã în generarea unui lanø Markov de stãri, ale cãror energii ar avea o distribuøie Boltzmann dacã lanøul ar avea o lungime infinitã. Deoarece lanøul este de lungime finitã, distribuøia rezultatã va fi doar apropiatã de distribuøia Boltzmann. Pe lângã temperaturã, procedura Metropolis mai are ca intrãri soluøia curentã *S* care va fi îmbunãtãøitã, şi valoarea *M*, care este durata de timp în care este aplicatã cãlirea la temperatura *T*. Procedura *SA* apelează procedura *Metropolis* la diferite temperaturi descrescãtoare. Temperatura este redusã lent de la valoarea T_0 în progresie geometricã, în funcøie de parametrul α. Durata de timp a procesului de cãlire la o anumitã temperaturã este mãritã gradat pe mãsurã ce temperatura este micşoratã. Aceasta se realizează utilizând parametrul $\beta > 0$.

```
Algorithm SA (S0, T0, α, β, M, TimpMax);
        /* S0 este soluția inițială */
        /* T0 este temperatura inițială */
        /* α este rata de răcire */
        /* β este o constantă */
        /* TimpMax este timpul total permis pentru procesul de călire */
        /* M reprezintă timpul până la următoarea actualizare a parametrilor */
begin
        T = T0;
        S = S0;
        Timp = 0;
        repeat
                Metropolis (S, T, M);
                Timp = Timp + M;
                T = α × T;
                M = β × M
        until (Timp ≥ TimpMax)
end.
```

Figura 3.6. Algoritmul de cãlire simulatã.

Procedura *Metropolis* este prezentatã în Figura 3.7. Aceasta utilizează procedura *Vecin* pentru a genera o vecinãtate localã *NewS* a unei soluøii date *S*. Funcøia *Cost* returnează costul unei soluøii date *S*. În cazul în care costul noii soluøii *NewS* este mai redus decât costul soluøiei curente *S*, noua soluøie este acceptabilã, şi se va seta *S* = *NewS*. Dacã noua soluøie are un cost mai mare comparativ cu soluøia originalã *S*, *Metropolis* va accepta noua soluøie într-un mod probabilistic. Se generează un numãr aleator între 0 şi 1; dacã acest numãr este mai mic decât $e^{-\Delta h/T}$, unde Δh este diferenøa între costuri, iar *T* este temperatura, soluøia inferioarã calitativ va fi acceptatã. Acest criteriu pentru acceptarea noii soluøii este numit *criteriu Metropolis*. Procedura *Metropolis* generează şi examinează *M* soluøii.

```
Algorithm Metropolis (S, T, M);
begin
        repeat
                NewS = Vecin (S);
                Δh = (Cost(NewS) - Cost(S));
                if ((Δh  < 0) or (random < e^(-Δh /T))) then S = New endif;
                M = M - 1
        until (M = 0)
end.
```

Figura 3.7. Procedura Metropolis.

Probabilitatea ca o soluție inferioară să fie acceptată de procedura *Metropolis* este dată de P ($random < e^{-\Delta h/T}$). Se presupune că generarea numerelor aleatoare urmărește o distribuție uniformă. În acest caz, probabilitatea se reduce la $e^{-\Delta h/T}$. Deoarece s-a presupus că soluția *NewS* este inferioară comparativ cu S, $\Delta h > 0$. La temperaturi foarte înalte, de exemplu $T \to \infty$, probabilitatea de sus se apropie de 1. Din contră, atunci când $T \to 0$, probabilitatea $e^{-\Delta h/T}$ se apropie de 0.

Pentru a se utiliza călirea simulată la rezolvarea problemei de bipartiționare, trebuie să se formuleze mai întâi o funcție de cost care să reflecte atât criteriul de echilibru, cât și ponderea setului de tăietură. Pentru o partiție dată (A, B) a circuitului, se definește

$$Dezechilibru\ (A, B) = \sum_{v \in A} s(v) - \sum_{v \in B} s(v) \tag{3.14}$$

$$Pondere\ set\ de\ tăietură\ (A, B) = \sum_{n \in \psi} w_n \tag{3.15}$$

unde $s(v)$ este dimensiunea vârfului v, w_n este ponderea conexiunii n, iar ψ este setul conexiunilor cu terminale atât în A cât și în B.

$$Cost\ (A, B) = W_c * Pondere\ set\ de\ tăietură\ (A, B) + W_s * Dezechilibru\ (A, B) \tag{3.16}$$

W_c și W_s sunt constante în domeniul [0, 1] care indică importanța relativă a minimizării setului de tăietură, respectiv a echilibrului. De observat că, spre deosebire de algoritmul Kernighan-Lin, restricția de echilibru este parte a funcției de cost. Dacă algoritmul se oprește la o minimă locală, partiția obținută poate fi dezechilibrată.

Cea mai simplă funcție de vecinătate se obține prin interschimbarea unei perechi de elemente, câte una din fiecare partiție. Se poate selecta de asemenea un subset de elemente din fiecare partiție pentru a fi interschimbate. O altă posibilitate este de a se selecta acele elemente a căror contribuție la costul extern este ridicată, sau cele care sunt conectate intern cu cel mai mic număr de vârfuri.

Studii teoretice au arătat că prin metoda călirii simulate se poate obține soluția optimă dacă sunt satisfăcute anumite condiții [189]. În practică, aceste condiții implică un număr infinit de iterații. Se utilizează diferite euristici pentru a reduce timpul de execuție al algoritmului. Totuși, algoritmul de călire simulată rămâne lent din punct de vedere computațional, și mai puțin eficient comparativ cu anumite euristici specifice diferitelor probleme.

3.4.5 Partiționarea prin tăietura proporțională

Fiind dat un circuit $N = (V, E)$, unde V este setul de noduri, iar E este setul de muchii, fie c_{ij} capacitatea unei muchii care conectează nodul i cu nodul j. (A, A') indică o tăietură care separă un set de noduri A de $A' = V - A$. Capacitatea acestei tăie-

turi este egalã cu $C_{AA'} = \sum_{i \in A} \sum_{j \in A'} c_{ij}$. Proporøia acestei tãieturi este definitã prin raportul

$$R_{AA'} = \frac{C_{AA'}}{|A| \times |A'|} \tag{3.17}$$

unde $|A|$ øi $|A'|$ reprezintã cardinalitatea subseturilor A øi A'. Aceastã metricã din ecuaøia 3.17 a fost propusã de Wei øi Cheng [175] øi s-a dovedit o funcøie obiectiv de succes pentru numeroase aplicaøii. Numãrãtorul reprezintã criteriul de tãieturã minimã, în timp ce numitorul favorizeazã o partiøie echilibratã, deoarece $|A| \times |A'|$ este maxim atunci când $|A| = |A'|$. Tãietura proporøionalã este tãietura care generează proporøia minimã $R_{AA'}$ dintre toate tãieturile circuitului, deci $\min_A (C_{AA'} / |A| \times |A'|)$ ($A \subset V$ øi $A \neq \emptyset$, $A' \neq \emptyset$).

Metoda de partiøionare prin tãietura proporøionalã are tendinøa de a identifica grupãrile naturale din circuit [50], [146], [175]. Aceastã proprietate poate fi interpretatã printr-un model de graf aleator. Figura 3.8 ilustrează un graf aleator cu distribuøie uniformã cu n noduri. Probabilitatea de a exista o muchie care sã conecteze fiecare pereche de noduri este egalã cu aceeaøi valoare f.

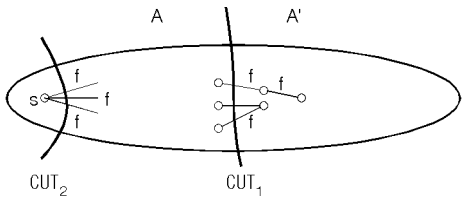

Figura 3.8. Un graf aleator cu n noduri øi probabilitate f.

Considerãm o tãieturã CUT_1 în Figura 3.8, care partiøioneazã circuitul în douã subseturi A øi A' cu dimensiuni comparabile de $\alpha \times n$ noduri, respectiv $(1 - \alpha) \times n$ noduri, unde $0 < \alpha < 1$. Capacitatea probabilã $C_{AA'}$ a tãieturii CUT_1 este egalã cu probabilitatea f multiplicatã cu numãrul posibil de muchii între A øi A' :

$$P(C_{AA'}) = f \times |A| \times |A'| = f \times \alpha \times (1 - \alpha) \times n^2 \tag{3.18}$$

Pe de altã parte, dacã o altã tãieturã CUT_2 în Figura 3.8 separã un singur nod s de restul nodurilor, numãrul probabil de muchii tãiate de CUT_2 este:

$$P(C_{\{s\}\{s\}'}) = f \times (n - 1) \tag{3.19}$$

Pe mãsurã ce n se apropie de infinit, valoarea din ecuaøia 3.18 devine mult mai mare decât cea din ecuaøia 3.19. Aceasta este o explicaøie a motivului pentru care metoda de flux maxim øi tãieturã minimã are tendinøa sã genereze subseturi de dimensiuni foarte diferite, pentru care valoarea de tãieturã este minimã [175]. De aceea, se propune raportul $(C_{AA'} / |A| \times |A'|)$ pentru a reduce acest efect.

Ca o consecinøã, valoarea probabilã a acestui raport este o constantã pentru diferite tãieturi:

$$P(R_{AA'}) = P\left(\frac{C_{AA'}}{|A| \times |A'|}\right) = \frac{f \times |A| \times |A'|}{|A| \times |A'|} = f \tag{3.20}$$

Deci, dacã muchiile grafului sunt distribuite uniform, toate tãieturile au aceeaøi valoare a raportului de tãieturã. Cu alte cuvinte, în grafurile aleatoare cu distribuøie uniformã nu are importanøã alegerea tãieturilor. Însã, într-un circuit general, diferite

tãieturi generează rapoarte diferite. Tãietuorile care trec prin grupuri slab conectate le corespund raporturi mai mici.

Ca şi alte probleme de partiøionare, gãsirea tãieturii proporøionale într-un circuit aparøine clasei problemelor NP-complete. De aceea, este necesarã o euristicã rapidã pentru a putea fi utilizatã în cazul circuitelor VLSI complexe. Wei şi Cheng [175] au elaborat o asemenea euristicã, bazatã pe algoritmul Fiduccia-Mattheyses [76], datoritã eficienøei acestuia. Algoritmul pentru tãietura proporøionalã constã din trei faze principale: 1) iniøializare, 2) deplasare iterativã, şi 3) interschimbarea grupurilor. Aceste faze sunt prezentate în continuare.

Iniøializare. În această fazã, restricøia privind dimensiunea subseturilor este eliminatã. Algoritmul stabileşte în mod dinamic propriile subseturi care sunt apropiate de grupãrile din circuit. Este selectat în mod aleator un modul *s* ca nucleu, iar apoi un alt modul *t* aflat la capãtul cãii celei mai lungi, prin cãutare în lãøime începând cu modulul *s*. Grupul va conøine la început fie modulul *s*, fie modulul *t*, şi de fiecare datã se adaugã la acest grup cel mai bun candidat, pânã când se epiuzează toate modulele (exceptând al doilea nucleu). Deoarece la fiecare pas de adãugare rezultã o nouã valoare a raportului, se memorează fiecare valoare chiar dacã ea este mai mare decât precedenta, astfel încât algoritmul poate evita minimele locale. Tãietura care determinã raportul minim formează o grupare iniøialã din circuit. Dacã circuitul are un set de *M* module, procedura de iniøializare poate fi descrisã astfel:

1. Se alege în mod aleator un modul *s*. Se determinã al doilea modul *t* la capãtul cãii celei mai lungi prin cãutare în lãøime începând de la modulul *s*. Fie $X = \{s\}$ şi $Y = M - \{s, t\}$.
2. Se alege un modul *i* din *Y* a cãrui mutare în *X* va genera cel mai bun raport faøã de toate celelalte module concurente. Se mutã modulul *i* din *Y* în *X*; se actualizează $X = X \cup \{i\}$ şi $Y = Y - \{i\}$.
3. Se repetã pasul 2 pânã când $Y = \varnothing$.
4. Se repetã paşii 2 şi 3 cu $X = \{t\}$ şi $Y = M - \{s, t\}$ pânã când $Y = \varnothing$.
5. Tãietura cu raportul minim care s-a gãsit în timpul procedurii formează partiøia iniøialã.

Deplasare iterativã. Se presupune cã modulul nucleu *s* este fixat în partea stângã, iar modulul *t* în partea dreaptã. Se defineşte o operaøie de deplasare la dreapta (stânga) ca fiind deplasarea modulelor cu valoarea cea mai bunã a raportului de tãieturã de la modulul *s* (*t*) cãtre modulul *t* (*s*).

După generarea unei partiøii iniøiale, se repetã operaøiile de deplasare în direcøia opusã pentru a se obøine o îmbunãtãøire suplimentarã. Procedura este aplicatã în mod iterativ ultimei partiøii. Dacã direcøia partiøionãrii iniøiale este de la *s* la *t*, procesul de deplasare începe cu urmãtoarele operaøii.

1. Se repetã operaøiile de deplasare la dreapta pânã la epuizarea tuturor modulelor.
2. Se alege valoarea minimã a raportului obøinut în pasul 1. Dacã noua valoare a raportului este mai micã faøã de pasul 1, tãietura care produce acest raport formează o nouã partiøie de început; în caz contrar, se returnează partiøia precedentã şi procesul se terminã.
3. Se repetã paşii 1 şi 2 cu operaøiile de deplasare la stânga.
4. Se repetã paşii 1 - 3.

Interschimbarea grupurilor. După terminarea deplasării iterative, se obøine o partiøionare minimã localã în sensul cã mutarea unui singur modul din subsetul sãu

curent în celãlalt set nu poate reduce valoarea raportului de tãieturã. Pentru a se obøine îmbunãtãøiri suplimentare, se utilizeazã o tehnicã de interschimbare a grupurilor.

Se defineşte câştigul raportului $r(i)$ a unui modul i modificarea raportului dacã modulului i (cu excepøia celor douã nuclee s şi t) ar fi mutat din subsetul sãu curent în celãlalt subset. Procesul de interschimbare a grupurilor este urmãtorul.

1. Se calculeazã câştigul raportului $r(i)$ pentru fiecare modul i, şi se seteazã toate modulele în starea “neblocatã”.
2. Se selecteazã un modul neblocat i cu câştigul raportului maxim.
3. Se mutã modulului i în celãlalt subset, şi se blocheazã.
4. Se actualizeazã câştigurile raporturilor pentru restul modulelor afectate şi neblocate.
5. Se repetã paşii 2 - 4 pânã când toate modulele vor fi blocate.
6. Dacã cel mai mare câştig al raportului care s-a acumulat în timpul acestui proces este pozitiv, se interschimbã grupul de module corespunzãtoare câştigului maxim, şi se continuã cu pasul 1; în caz contrar, se returneazã partiøia precedentã şi procedura se terminã.

Implementarea tehnicii de interschimbare a grupurilor se bazeazã pe structura de date propusã de Fiduccia şi Mattheyses [76]. Fie (A, A') tãietura rezultatã din deplasarea iterativã. Câştigul raportului pentru mutarea modulului i cu dimensiunea $s(i)$ din A' în A este exprimat prin $r(i) = g(i) / [|A| + s(i)] \times |A'| - s(i)]$, unde $g(i)$ este decrementul numãrului de conexiuni tãiate dacã modulul i ar fi mutat în celãlalt subset. Deoarece fiecare $g(i)$ este un întreg în domeniul $-p(i)$ la $p(i)$, unde $p(i)$ este numãrul de pini din modulul i, structura de date poate fi reprezentatã printr-un tabel a cãrei intrare k conøine o listã dublu înlãnøuitã de module cu valorile $g(i)$ egale cu k la un moment dat. Se pãstreazã douã asemenea structuri de date, câte una pentru fiecare subset.

Faza de iniøializare şi cea de deplasare iterativã pot fi implementate ca şi cazuri speciale ale interschimbãrii grupurilor. Primele douã faze realizeazã deplasarea modulelor într-o singurã direcøie, în timp ce interschimbarea grupurilor are flexibilitatea de a muta modulele în ambele direcøii.

Interschimbarea grupurilor care utilizeazã structura de date descrisã în [76] are complexitatea $O(P)$, unde P este numãrul de pini. Gãsirea cãii celei mai lungi prin cãutare în lãøime are de asemenea complexitatea $O(P)$, ca şi fazele de iniøializare şi deplasare iterativã [175]. Astfel, complexitatea întregului algoritm rãmâne liniarã cu numãrul de pini.

În algoritmul prezentat anterior a fost eliminatã restricøia asupra dimensiunii subseturilor, metoda furnizând automat subseturile care sunt grupuri naturale în cadrul circuitului. Astfel, existã o libertate deplinã în ceea ce priveşte dimensiunile subseturilor rezultate. Uneori însã este necesar sã se impunã o restricøie de dimensiune asupra subseturilor rezultate datoritã limitãrilor fizice ale circuitelor.

Pentru a se realiza o partiøionare cu restricøii de dimensiune, se specificã o limitã superioarã a dimensiunii subsetului. La fiecare aplicare a algoritmului original de tãieturã proporøionalã, se obøin douã subseturi. Dacã dimensiunea unuia din subseturi este mai mare decât limita superioarã, se eliminã subsetul mai mic şi se continuã aplicarea algoritmului asupra subsetului rãmas în circuitul redus pânã când dimensiunile tuturor subseturilor sunt mai mici decât limita superioarã.

Ultimul pas este repunerea grupurilor mici care au fost eliminate şi distribuirea lor în subseturile potrivite. Se poate considera subsetul mai mare din ultima partiøionare ca fiind un subset, şi toate celelalte subseturi generate în timpul procesului ca un alt subset. Algoritmul aplicã o iteraøie a operaøiilor de deplasare la stânga/dreapta pentru a muta un anumit numãr de module din subsetul mai mare în subsetul mai

mic, şi gãseşte tãietura proporøionalã cea mai bunã care satisface restricøia de dimensiune. Calitatea partiøionãrii poate fi sacrificatã pentru a se putea satisface restricøia de dimensiune.

Fiind dat un circuit N şi un întreg *lim_sup*, algoritmul cu restricøie de dimensiune poate fi descris astfel [175]:

1. Se iniøializeazã $\Psi = \{N\}$.
2. Se alege $N' \in \Psi$ astfel ca $|N'| = \max |A_i|$, $A_i \in \Psi$. Se setează $\Psi = \Psi - \{N'\}$. Se aplicã algoritmul de tãieturã proporøionalã pentru a obøine o tãieturã (A, A'), unde $N' = A \cup A'$, presupunând $|A| \geq |A'|$.
3. Dacã ($|A| \leq lim\_sup$), se continuã cu pasul 4; în caz contrar, fie $\Psi = \Psi \cup \{A, A'\}$, şi se continuã cu pasul 2.
4. Fie A un subset, iar $A'' = N - A$ al doilea subset. Se executã o iteraøie a operaøiilor de deplasare la dreapta/stânga între A şi A'' pentru a gãsi tãietura proporøionalã cea mai bunã care satisface restricøia de dimensiune.
5. Se returneazã rezultatul final.

Comparativ cu algoritmul Fiduccia-Mattheyses, rezultatele experimentelor efectuate în [175] aratã cã rezultatele sunt îmbunãtãøite în medie cu 40%, atunci când este eliminatã restricøia de dimensiune. Aceasta deoarece sunt explorate domenii mai largi ale spaøiului soluøiilor. În cazul impunerii restricøiilor de dimensiune, îmbunãtãøirea medie faøã de algoritmul Fiduccia-Mattheyses este de 28% în privinøa capacitãøii tãieturii şi de 20% în privinøa raportului de tãieturã.

3.4.6 Partiționarea cu performanțe stabile

Performanøele algoritmului Kernighan-Lin s-au dovedit a fi foarte dependente de alegerea partiøiei iniøiale, iar rezultatele finale variazã în mod semnificativ ca o consecinøã a diferitelor partiøii de început [50]. Pentru a se evita blocarea în minime locale, este necesar un numãr mare de rulãri ale algoritmului asupra unor partiøii iniøiale generate aleator, proces care necesitã un timp ridicat. În plus, probabilitatea de a gãsi soluøia optimã într-o singurã încercare scade exponenøial pe mãsurã ce dimensiunea circuitului creşte.

Kernighan şi Lin au sugerat faptul cã o altã rulare a algoritmului de partiøionare poate îmbunãtãøi rezultatele generate de prima încercare. Presupunem, de exemplu, cã se obøine rezultatul unei partiøionãri (A, B) pornind de la o anumitã partiøie iniøialã. Dacã se împarte A în A_1 şi A_2, iar B în B_1 şi B_2, se poate rula din nou algoritmul Kernighan-Lin fie asupra $(A_1 \cup B_1, A_2 \cup B_2)$, fie asupra $(A_1 \cup B_2, A_2 \cup B_1)$. Dacã se obøine o partiøie mai bunã, procedura poate fi repetatã. Totuşi, îmbunãtãøirea obøinutã din rezultatul final precedent necesitã mai mult timp, complexitatea fiind aceeaşi cu cea a problemei originale. În plus, trebuie sã se indice o anumitã dimensiune a subsetului înaintea generãrii partiøiei iniøiale. Acest proces nu avantajează identificarea grupelor naturale din circuit, deoarece este imposibil sã se determine dimensiunea grupelor înaintea partiøionãrii.

Pe baza acestor observaøii, Cheng şi Wei [50] ajung la concluzia cã o tehnicã de grupare de sus în jos este esenøialã pentru o partiøionare eficientã. Pentru a reduce dimensiunea circuitelor, se utilizează o tehnicã de partiøionare recursivã de sus în jos, şi se divide întregul circuit în grupuri mici, puternic conectate. Astfel, grupurile generate la fiecare partiøionare se reduc ca dimensiune. Ultimul pas constã în rearanjarea grupurilor în douã subseturi care respectã restricøia de dimensiune. Deoarece numãrul grupurilor este relativ mic comparativ cu numãrul modulelor circuitului, se pot efectua numeroase încercãri ale operaøiei de rearanjare. De asemenea, există o probabilitate ridicatã de a se obøine o soluøie apropiatã de cea optimã, deoarece numãrul grupurilor

generate este mic. Algoritmul de partiøionare se bazează pe tãietura proporøionalã [175].

Circuitul este divizat mai întâi în grupuri mici, prin execuøia recursivã a algoritmului de tãieturã proporøionalã. Fiecare grup va conøine un numãr diferit de module de dimensiuni diferite. Aceste grupuri sunt rearanjate apoi în douã subseturi, respectându-se restricøiile de dimensiune. Scopul principal al acestei etape este de a se minimiza ponderea tãieturii între cele douã subseturi. Se aplicã apoi algoritmul Fiduccia-Mattheyses circuitului contractat, în mod repetat, pentru a se obøine rezultate bune. În final, se executã din nou algoritmul Fiduccia-Mattheyses asupra circuitului original expandat.

Un circuit având conexiuni multipin poate fi definit cu ajutorul un model de hipergraf. Fie $N = (V, E)$ reprezentând circuitul având m module şi n conexiuni, unde $V = \{v_1, v_2, \ldots, v_m\}$ şi $E = \{e_1, e_2, \ldots, e_n\}$. Fiecare muchie, care corespunde unei conexiuni multipin, este un subset al lui V, deci $e_i \subseteq V$, $|e_i| \geq 2$, $1 \leq i \leq n$. Se aplicã în mod recursiv algoritmul de tãieturã proporøionalã circuitului N şi se divide V într-un numãr de grupuri mici, m'. Fie Ψ un set pentru colecøia grupurilor generate, $\Psi = \{V_i \mid 1 \leq i \leq m'\}$. Se construieşte apoi un circuit contractat $H = (V', E')$ dupã cum urmează. În primul rând, fiecare element din Ψ va fi un nod în circuitul contractat, deci $V' = \{i \mid V_i \in \Psi\}$. Apoi, pentru fiecare muchie e_j din E se construieşte o nouã muchie e_j'. Muchia e_j' conectează nodul i din V' dacã e_j conectează oricare module din V_i, deci $e_j' = \{i \mid$ existã $v_k \in V_i$, astfel încât $v_k \in e_j\}$. În final, $E' = \{e_j' \mid |e_j'| \geq 2\}$.

Fiind dat un circuit $N = (V, E)$, un întreg g indicând numãrul grupurilor aşteptate, un întreg *nr_rep* indicând numãrul de repetãri, iar *dim1* şi *dim2* restricøiile de dimensiune ale celor douã subseturi rezultante, algoritmul de partiøionare cu performanøe stabile poate fi descris astfel:

1. Se iniøializează $\Psi = V$ şi se calculează dimensiunea totalã a circuitului *dimt*.
2. Fie V^* un subset din Ψ astfel încât $|V^*| = \max_{V_i \in \Psi} |V_i|$. Cât timp $|V^*| >$ *dimt*/*g*, repetã pasul 3.
3. Se setează $\Psi = \Psi - \{V^*\}$. Se aplicã algoritmul de tãieturã proporøionalã [175] subsetului V^* pentru a obøine o tãieturã (A, A') unde $V^* = A \cup A'$. Se setează $\Psi = \Psi \cup \{A, A'\}$.
4. Se construieşte un circuit contractat $H = (V', E')$.
5. Se aplicã algoritmul Fiduccia-Mattheyses de *nr_rep* ori circuitului H cu restricøiile de dimensiune *dim1*, *dim2*.
6. Se utilizează rezultatul cel mai bun de la pasul 5 pentru circuitul N. Se aplicã algoritmul Fiduccia-Mattheyses o singurã datã circuitului N cu restricøiile de dimensiune *dim1*, *dim2*.

Fiind datã o valoare g, fie $f(g)$ numãrul de grupuri generate dupã execuøia partiøionãrii recursive în paşii 2 şi 3. Deoarece tãietura proporøionalã necesitã $O(p)$ operaøii [175], unde p este numãrul total de pini, iteraøiile din paşii 2 şi 3 necesitã cel mult $O(f(g) \times p)$ operaøii. Complexitatea pasului 4 este $O(p)$, iar complexitatea paşilor 5 şi 6 este $O(nr\_rep \times p)$. Astfel, complexitatea întregului algoritm este limitatã la $O((f(g) + nr\_rep) \times p)$.

În experimentele efectuate Cheng şi Wei [50], valoarea medie a ponderii tãieturii este ridicatã atunci când g are valoare micã, $g < 25$. Pentru valori peste 100, numãrul de subseturi din Ψ se apropie de numãrul total de module ale circuitului, iar operaøiile se comportã similar cu algoritmul Fiduccia-Mattheyses. Rezultatele cele mai bune s-au obøinut pentru g cuprins între 25 şi 100, pentru alte experimente fiind aleasã valoarea $g = 50$. Cu algoritmul de partiøionare cu performanøe stabile s-a obøinut o îm-

bunãtãøire de 25% faøã de cele mai bune rezultate şi de 48% faøã de media rezultatelor obøinute cu algoritmul Fiduccia-Mattheyses [50].

3.4.7 Partiționarea prin metode spectrale

O clasã importantã de tehnici de partiøionare este reprezentatã de metodele "spectrale", care utilizeazã valori proprii şi vectori proprii ale matricilor obøinute din graful circuitului. Un circuit poate fi reprezentat ca un graf nedirecøionat $G = (V, E)$ cu $|V| = n$ vârfuri $v_1, \ldots, v_n$. G se poate reprezenta printr-o *matrice de adiacenøã* de $n \times n$ elemente $A = A(G)$, unde $A_{ij} = 1$ dacã $(v_i, v_j) \in E$ şi $A_{ij} = 0$ în caz contrar. Dacã G are muchii ponderate, A_{ij} este egal cu ponderea muchiei $(v_i, v_j) \in E$, şi prin convenøie $A_{ii} = 0$ pentru orice $i = 1, \ldots, n$. Dacã se noteazã cu $d(v_i)$ gradul vârfului v_i (suma ponderilor tuturor muchiilor incidente lui v_i), se obøine *matricea de grad*, notatã cu D, care este o matrice diagonalã definitã prin $D_{ii} = d(v_i)$. Uneori, se utilizeazã d_i pentru a nota $d(v_i)$. Valorile proprii şi vectorii proprii ale unor asemenea matrici sunt studiate de un subdomeniu al teoriei grafurilor care se ocupã de spectrele grafurilor.

Studii teoretice efectuate de Barnes, Donath şi Hoffman au stabilit relaøii între proprietãøile spectrale şi proprietãøile de partiøionare ale grafurilor [86]. Au fost utilizate metode bazate pe vectori proprii şi valori proprii pentru plasarea modulelor în sisteme CAD, şi pentru bisecøionarea cu lãøime minimã. În contextul amplasãrii circuitelor, aceste studii formuleazã problema de partiøionare ca asignarea sau plasarea modulelor în grupuri de dimensiune limitatã. Problema este transformatã apoi într-o optimizare cuadraticã, şi o formulare a unui Lagrangian conduce apoi la calculul unor vectori proprii.

Hagen şi Kahng au arãtat în [86] legãtura teoretricã existentã între spectrele grafurilor şi tãieturile proporøionale optime. În teoria dezvoltatã se utilizeazã vectori proprii ale matricii $Q = D - A$, numitã Laplacian al lui G, unde D şi A sunt definite mai sus, matrice care a fost utilizatã şi de cãtre Hall [87]. Boppana, Donath şi Hoffman, ca şi aløi autori, utilizeazã matrici diferite derivate din graful circuitului, dar se bazeazã pe proprietãøi matematice similare pentru a elabora formulãri bazate pe valori proprii şi a defini relaøia existentã cu partiøionarea.

Dupã cum a arãtat Hall [87], vectorii matricii $Q = D - A$ soluøionează problema de *plasare cuadraticã* uni-dimensionalã pentru determinarea vectorului $x = (x_1, x_2, \ldots, x_n)$ care minimizează

$$z = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j)^2 A_{ij} \tag{3.21}$$

cu restricøia $|x| = (x^T x)^{1/2} = 1$.

Se poate arãta cã $z = x^T Q x$, astfel cã pentru a minimiza z se poate forma un Lagrangian

$$L = x^T Q x - \lambda (x^T x - 1) \tag{3.22}$$

Dacã se considerã prima derivatã parøialã a lui L faøã de x şi se egalează cu zero rezultã

$$2Qx - 2\lambda x = 0 \tag{3.23}$$

care se poate rescrie ca

$$(Q - \lambda I)\, x = 0 \tag{3.24}$$

unde I este matricea de identitate. Aceasta este o formulare bazatã pe valori proprii pentru λ, iar vactorii proprii ai lui Q sunt singurele soluøii pentru x care nu sunt banale. Valoarea proprie minimã 0 conduce la soluøia neinteresantã $x = (1/\sqrt{n}, 1/\sqrt{n},$

…, $1/\sqrt{n}$), şi de aceea se utilizează vectorul propriu corespunzător celei de-a doua valori proprii.

Hagen şi Kahng [86] au arătat următoarele proprietăøi de bază ale matricii Q:

a) Q este simetrică şi non-negativă, deci (i) $x^TQx = \sum_{i,j} Q_{ij} x_i x_j \geq 0$ $\forall x$, şi (ii) toate valorile proprii ale lui Q sunt ≥ 0.

b) Valoarea proprie minimă a lui Q este 0, cu vectorul propriu **1** = (1, 1, …, 1).

c) Produsul intern x^TQx corespunde "lungimii pătrate", deci

$$x^TQx = x^TDx - x^TAx = \sum_i d_i x_i^2 - \sum_{i,j} A_{ij} x_i x_j = \sum_i d_i x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j$$

care se poate scrie ca pătratul complet

$$x^TQx = \sum_{(i,j)\in E} (x_i - x_j)^2 \tag{3.25}$$

d) În final, principiul Minimax Courant-Fischer [86] implică

$$\lambda = \min_{x \perp \mathbf{1},\, x \neq \mathbf{0}} \frac{x^TQx}{|x|^2} \tag{3.26}$$

unde λ este a doua valoare proprie cea mai mică a lui Q.

Proprietăøile c) şi d) stabilesc o nouă relaøie între costul tăieturii proporøionale optime şi a doua valoare proprie λ a matricii $Q = D - A$. Aceasta este formulată în următoarea teoremă [86].

> **Teorema 3.4.7.1**. Fiind dat un graf $G = (V, E)$ cu matricea de adiacenøă A, matricea diagonală de grad D, şi $|V| = n$, a doua valoare proprie cea mai mică λ a lui $Q = D - A$ indică o limită inferioară a costului c al partiøiei tăieturii proporøionale optime, cu $c \geq (\lambda/n)$.

Rezultatele acestei teoreme sugerează următoarea metodă de partiøionare: se calculează $\lambda(Q)$ şi vectorul propriu corespunzător x, şi apoi se utilizează x pentru a construi o euristică pentru tăietura proporøională. Dacă circuitul este specificat ca un hipergraf $H = (V, E')$, atunci acesta trebuie transformat mai întâi într-un graf $G = (V, E)$.

Pentru transformarea hipermuchiilor din modelul circuitului în muchii ale grafului G se consideră că fiecare conexiune cu k pini contribuie cu un subgraf complet conøinând cele k module ale sale, ponderea fiecărei muchii fiind egală cu $1/(k\text{-}1)$. Deci, matricea de adiacenøă A este construită astfel: Pentru fiecare pereche de module v_i şi v_j cu $p \geq 1$ conexiuni în comun, fie $|s_1|$, $|s_2|$, …, $|s_p|$ numărul modulelor din conexiunile comune s_1, s_2, …, s_p. Atunci

$$A_{ij} = \sum_{l=1}^{p} \frac{1}{|s_l| - 1} \tag{3.27}$$

Se pot considera şi diferite metode de rarificare a matricii Q, de exemplu ignorarea conexiunilor mai puøin importante (non-critice), sau aproximarea cu 0 a elementelor cu valoare mică ale matricii. Aceste metode sunt importante deoarece cei mai muløi algoritmi numerici vor avea un timp de execuøie mai redus în cazul intrărilor reprezentate prin matrici rare.

Deoarece trebuie calculat un singur vector propriu al unei matrici simetrice, complexitatea acestui calcul este relativ redusă. În plus, matricile de adiacenøă ale cir-

cuitelor tind sã fie rare datoritã organizãrii ierarhice a circuitului şi a restricøiilor tehnologice. Aceasta permite aplicarea tehnicilor numerice pentru matricile rare, în particular a metodei Lanczos.

Fiind datã o matrice Q de $n \times n$ elemente, algoritmul Lanczos calculează în mod iterativ o matrice simetricã tridiagonalã T a cãrei valori proprii vor fi foarte apropiate de valorile proprii ale lui Q. Dacã numãrul de valori proprii care trebuie calculate este redus, numãrul de iteraøii necesare pentru obøinerea matricii T va fi de obicei mult mai mic decât n. Deoarece T este tridiagonalã şi simetricã, se poate calcula rapid o valoare proprie λ a matricii T, care se poate utiliza pentru calculul vectorului propriu x corespunzãtor matricii Q.

Pornind de la al doilea vector propriu x, se pot utiliza urmãtoarele euristici pentru construirea partiøiei pe baza tãieturii proporøionale [86]:

a) Partiøionarea modulelor pe baza sgn (x), deci U = {modul i: $x_i \geq 0$} şi W = {modul i: $x_i < 0$}.

b) Partiøionarea modulelor în jurul valorii mediane x_i, depunând prima jumãtate în U şi a doua jumãtate în W.

c) Utilizarea relaøiei euristice între x şi o plasare cuadraticã uni-dimensionalã, conform cãreia un interval în lista sortatã a valorilor x_i indicã o partiøie naturalã.

d) Sortarea x_i pentru a obøine o ordonare liniarã a modulelor, iar apoi determinarea indexului de divizare r care produce tãietura proporøionalã cea mai bunã.

În cazul d), cele n componente x_i ale vectorului propriu sunt sortate, rezultând o ordonare $v = v_1, \ldots, v_n$ a modulelor. Este determinat apoi indexul de divizare r, $1 \leq r \leq n$-1, care produce cel mai bun cost al tãieturii proporøionale atunci când modulele cu indexul > r sunt plasate în U, iar cele cu indexul ≤ r sunt plasate în W.

Algoritmul de partiøionare care utilizează euristica d) pentru construirea partiøiei este prezentat în Figura 3.9.

Algorithm *EIG1*;
begin
$H = (V, E')$ este hipergraful circuitului;
Pas 1. Se transformă fiecare hipermuchie cu k pini din H într-un subgraf complet din $G = (V, E)$ cu ponderea uniformă a muchiilor $1 / k$ - 1;
Pas 2. Se calculează A = matricea de adiacenţă şi D = matricea de grad a grafului G;
Pas 3. Se calculează a doua valoare proprie cea mai mică a matricii $Q = D - A$ prin algoritmul Lanczos;
Pas 4. Se calculează vectorul propriu asociat v;
Pas 5. Se sortează componentele lui v şi se determină indexul de divizare care produce tăietura proporţională cea mai bună;
Pas 6. Se returnează partiţia cea mai bună găsită
end.

Figura 3.9. Algoritmul de partiøionare pe baza vectorilor proprii.

Un numãr de autori au arãtat faptul cã gãsirea grupurilor naturale din cadrul circuitelor este utilã pentru mai multe aplicaøii, de exemplu: a) partiøionarea cu cãi multiple, în special atunci când dimensiunile partiøiilor nu sunt cunoscute dinainte; b) amplasarea constructivã a modulelor; c) situaøiile în care dimensiunea circuitului este foarte mare şi trebuie utilizatã gruparea pentru a reduce dimensiunea intrãrii asupra cãreia se aplicã partiøionarea. Aceste grupuri pot fi gãsite în mod eficient prin

utilizarea metodei spectrale, deoarece al doilea vector propriu *x* conøine atât informaøii de partiøionare, cât øi de grupare. Hagen øi Kahng au arãtat cã o interpretare directã a celui de-al doilea vector propriu sortat poate identifica imediat grupurile naturale din cadrul grafului. Rezultatele obøinute sunt foarte bune mai ales pentru circuite de dimensiuni mari.

Deoarece partiøionarea pe baza vectorilor proprii ignorã informaøiile despre suprafaøa modulelor, aceastã metodã este adecvatã pentru circuite bazate pe celule standard sau reøele de porøi [86]. Faptul cã metoda spectralã ignorã ponderile modulelor nu constituie o dificultate pentru numeroase aplicaøii de partiøionare utilizate în proiectarea asistatã, de exemplu pentru test sau simulare hardware, unde intrarea este reprezentatã de hipergraful circuitului cu ponderi uniforme ale nodurilor.

În urma experimentelor efectuate în [86], prin metoda spectralã s-a obøinut o îmbunãtãøire medie de 9% a raportului de tãieturã faøã de rezultatul obøinut de Wei øi Cheng [175] prin metoda tãieturii proporøionale, implementatã prin programul RCut1.0. Timpul de calcul este competitiv cu cel al programului RCut1.0. Chiar øi partiøiile iniøiale generate prin metoda spectralã au calitate mai bunã decât partiøiile obøinute în urma îmbunãtãøirilor iterative.

3.4.8 Partiționarea pe baza rețelelor de flux

Teorema fluxului maxim øi tãieturii minime (Ford øi Fulkerson) pentru reøele este o importantã tehnicã combinatoricã de optimizare. Aceasta are numeroase aplicaøii în proiectarea circuitelor VLSI, ca de exemplu plasarea liniarã sau maparea tehnologicã pentru circuitele FPGA. Tehnica fluxului maxim øi tãieturii minime este o metodã naturalã de aflare a tãieturii minime într-un graf. Totuøi, ea nu a fost utilizatã pe scarã largã pentru partiøionarea circuitelor din urmãtoarele motive:

1) Cele douã componente obøinute pot fi dezechilibrate.
2) Deøi se poate obøine o tãieturã echilibratã prin aplicarea repetatã a tãieturii minime componentei celei mai mari, aceastã metodã poate necesita *n* operaøii de calcul a fluxului maxim, unde *n* este dimensiunea reøelei.
3) Tehnica tradiøionalã de flux în reøele opereazã asupra grafurilor, dar hipergrafurile reprezintã modele de acurateøe mai mare pentru listele de conexiuni ale circuitelor decât grafurile.

Yang øi Wong [180] au propus o metodã pentru modelarea exactã a unei liste de conexiuni (sau, în mod echivalent, a unui hipergraf) printr-o reøea de flux, øi o euristicã de bipartiøionare echilibratã bazatã pe utilizarea repetatã a tehnicii fluxului maxim øi tãieturii minime. Aceøtia utilizeazã o noøiune de *bipartiøie r-echilibratã*, care este o bipartiøie astfel încât o componentã are o pondere care este o fracøiune *r* a ponderii totale *W*. În cazul special când $r = ½$, o bipartiøie *r*-echilibratã este o bipartiøie echilibratã. Deoarece în practicã nu este necesarã impunerea strictã a criteriului de *r*-echilibrare, se poate introduce un factor de deviere ε pentru a permite devierea ponderii unei componente de la $(1 - \varepsilon)\ rW$ la $(1 + \varepsilon)\ rW$.

O reøea de flux $G = (V, E)$ este un graf orientat în care fiecare muchie $e \in E$ are o *capacitate* $c(e) \geq 0$. Sunt specificate douã noduri *s* øi *t* din *V*: nodul *s* este numit *sursã*, iar nodul *t* este numit *destinaøie* (Figura 3.10). Un *flux s-t* (sau *flux*) din *G* este o funcøie cu valori reale $f : E \rightarrow R$ astfel încât

1) pentru toate muchiile $e \in E$, $0 \leq f(e) \leq c(e)$, øi
2) pentru toate nodurile $u \in V \setminus \{s, t\}$, suma fluxului de intrare în *u* este egalã cu suma fluxului de ieøire din *u*.

O muchie *e* din *E* este *saturatã* dacã *f*(*e*) = *c*(*e*). *Valoarea* |*f*| a unui flux *f* este definitã ca suma fluxului de ieşire din *s*, care este egalã cu suma fluxului de intrare în *t*. Un *flux maxim* din *G* este un flux cu valoare maximã din *s* în *t*.

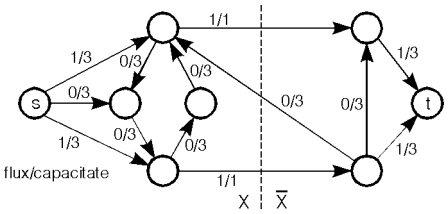

Figura 3.10. Un exemplu de reøea de flux *G* şi tãietura corespunzãtoare de capacitate minimã.

O *tãieturã s-t* (sau *tãieturã*) (*X*, *X'*) a unei reøele de flux *G* = (*V*, *E*) este o bipartiøie a lui *V* în *X* şi *X'* astfel încât $s \in X$ şi $t \in X'$. O muchie cu nodul de început în *X* şi nodul de sfârşit în *X'* este numitã *muchie directã*. O muchie cu nodul de sfârşit în *X* şi nodul de început în *X'* este numitã *muchie inversã*. *Capacitatea tãieturii* (*X*, *X'*), notatã prin *cap*(*X*, *X'*), este suma capacitãøilor numai pe muchiile directe din *X* în *X'*. O *cale de augmentare* din *u* în *v* este o cale simplã din *u* în *v* din graful nedirecøionat obøinut din reøea prin ignorarea direcøiei muchiilor, care se poate utiliza pentru a crea un flux suplimentar din *u* în *v*.

Teorema fluxului maxim şi tãieturii minime poate fi enunøatã astfel [180]:

Teorema 3.4.8.1. Fiind dat un flux maxim *f* din *G*, fie $X = \{v \in V : \exists$ o cale de augmentare în *G* din *s* în *v* $\}$, şi fie $X' = V \setminus X$. Atunci (*X*, *X'*) este o tãieturã de capacitate minimã (care este egalã cu |*f*|), şi *f* saturează toate muchiile directe din *X* în *X'*.

Figura 3.10 exemplificã un flux maxim în *G* şi tãietura corespunzãtoare de capacitate minimã. Eticheta *x*/*y* a unei muchii indicã faptul cã fluxul şi capacitatea muchiei sunt *x*, respectiv *y*. Muchiile îngroşate sunt muchii directe ale tãieturii (*X*, *X'*).

În continuare se descrie modelarea unei conexiuni într-o reøea de flux.

Lista de conexiuni a unui circuit secvenøial se poate reprezenta ca un *digraf* (care poate conøine cicluri direcøionate) *N* = (*V*, *E*), unde *V* este un set de noduri reprezentând porøi combinaøionale şi registre, iar *E* este un set de muchii reprezentând interconexiunile dintre porøi şi registre. Fiecare nod *v* din *V* are o pondere asociatã $w(v) \in R^+$. Ponderea totalã a unui subset $U \subseteq V$ este notatã prin *w*(*U*). Fie *W* = *w*(*V*) ponderea totalã a circuitului *N*.

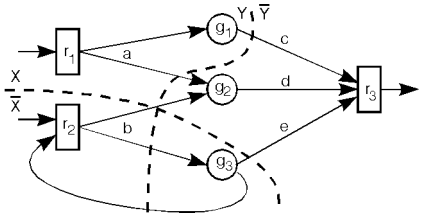

Figura 3.11. Un digraf *N* reprezentând un circuit secvenøial şi tãieturile conexiunilor acestuia.

O *conexiune* $n = (v; v_1, \ldots, v_l)$ din N este un set de muchii de ieșire din nodul v. De exemplu, în Figura 3.11 conexiunea a constă din două muchii (r_1, g_1) și (r_1, g_2). Fiind date două noduri s și t din N, o *tăieturã* s-t (sau *tăieturã*) (X, X') din N este o bipartiție a nodurilor din V astfel încât $s \in X$ și $t \in X'$. *Conexiunile tăiate* ale tăieturii, notate cu $net(X, X')$, sunt reprezentate de setul de conexiuni din N care sunt incidente atât nodurilor din X, cât și celor din X'. În Figura 3.11, dacă se alege $s = g_1$ și $t = g_3$, atunci setul de conexiuni tăiate $net(Y, Y')$ corespunzătoare tăieturii (Y, Y') constă din conexiunile c, a, b, e, unde conexiunile c, a, b conectează noduri din Y cu cele din Y', iar conexiunea e conectează un nod din Y' cu unul din Y. O tăieturã (X, X') este o *tăieturã minimă a conexiunilor* dacă $|net(X, X')|$, deci numărul de conexiuni din $net(X, X')$, este minim dintre toate tăieturile lui N. În Figura 3.11, $net(X, X') = \{b, e\}$, $net(Y, Y') = \{c, a, b, e\}$, iar (X, X') este o tăieturã minimă a conexiunilor.

Problema găsirii unei tăieturi minime a conexiunilor în $N = (V, E)$ se poate reduce la problema găsirii unei tăieturi de capacitate minimă, care se poate rezolva prin teorema fluxului maxim și tăieturii minime. Dacă toate muchiile tăiate au o capacitate unitară, problema este echivalentă cu găsirea unei tăieturi cu numărul minim de muchii directe din X în X'.

Construirea unei rețele de flux $N' = (V', E')$ din $N = (V, E)$ se poate realiza prin următoarea procedură [180] (Figura 3.12):

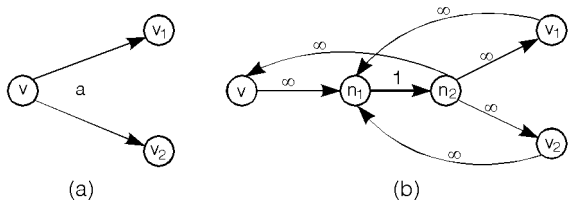

Figura 3.12. (a) O conexiune n în circuitul N. (b) Nodurile și muchiile conexiunii n din N'.

1. V' va conține inițial toate nodurile din V.
2. Pentru fiecare conexiune $n = (v: v_1, \ldots, v_l)$ din N, se adaugă două noduri n_1 și n_2 în V' și o muchie punte (n_1, n_2) în E'.
3. Pentru fiecare nod $u \in \{v, v_1, \ldots, v_l\}$ incident conexiunii n, se adaugă două muchii (u, n_1) și (n_2, u) în E'.
4. Fie s sursa lui N' și t destinația lui N'.
5. Se asignează capacități unitare tuturor muchiilor punte și capacități infinite tuturor celorlalte muchii din E'.
6. Pentru un nod $v \in V'$ corespunzător unui nod din V, $w(v)$ este ponderea lui v din N. Pentru un nod $u \in V'$ divizat dintr-o conexiune, $w(u) = 0$.

Lema 3.4.8.2 arată că dimensiunea rețelei de flux N' este mai mare doar cu un factor constant față de dimensiunea digrafului N.

Lema 3.4.8.2. Fie $N' = (V', E')$ rețeaua de flux construită dintr-un digraf $N = (V, E)$ utilizând procedura descrisă anterior. Atunci $|V'| \leq 3|V|$ și $|E'| \leq 2|E| + 3|V|$.

Procedura de construire a unei rețele de flux se poate aplica și atunci când circuitul N este reprezentat printr-un hipergraf.

Ihler, Wagner şi Wager au arãtat în [97] cã modelarea hipergrafurilor prin grafuri (cu ponderi pozitive) cu aceleaşi proprietãøi de tãieturã minimã nu este posibilã. Metoda descrisã mai sus modeleazã un hipergraf numai pentru algoritmii de partiøionare bazaøi pe reøele de flux. Diferenøele dintre modelul utilizat în [97] şi modelul de sus sunt urmãtoarele:

1) Ponderea unei tãieturi în [97] este calculatã ca suma tuturor muchiilor tãiate cu ponderi fixe, în timp ce în procedura descrisã ponderea unei tãieturi este calculatã ca suma capacitãøilor muchiilor directe.
2) În [97] se încearcã modelarea hipergrafurilor pentru un domeniu larg de algoritmi de partiøionare elaboraøi numai pentru grafuri obişnuite, în timp ce în procedura de sus se modeleazã un hipergraf numai pentru algoritmii bazaøi pe reøele de flux.

În Teorema 3.4.8.3 şi Corolarul 3.4.8.4 [180] se aratã cã problema determinãrii unei tãieturi minime a conexiunilor din N se poate reduce la problema determinãrii unei tãieturi cu capacitate minimã din N'.

Teorema 3.4.8.3. N are o tãieturã cu dimensiunea tãieturii conexiunilor de cel mult C dacã şi numai dacã N' are o tãieturã de capacitate cel mult C.

Corolarul 3.4.8.4. Fie (Y, Y') o tãieturã de capacitate minimã C din N', şi fie (X, X') tãietura din N construitã astfel: Se eliminã toate conexiunile din N corespunzãtoare muchiilor directe de la Y la Y'. N va fi divizat atunci în mai multe componente disjuncte, iar s şi t vor fi în componente diferite. Se alege X ca fiind componenta conøinând s, iar X' reuniunea celorlalte componente. Atunci (X, X') este o tãieturã minimã a conexiunilor din N, iar $|net(X, X')| = C$.

Ca rezultat al corespondenøei între hipergrafuri şi reøelele de flux se poate gãsi o tãieturã minimã a conexiunilor într-un circuit, utilizând urmãtorul algoritm.

1. Se construieşte reøeaua de flux $N' = (V', E')$ pentru N dupã procedura descrisã anterior.
2. Se determinã un flux maxim în N' de la s la t.
3. Se determinã o tãieturã (Y, Y') de capacitate minimã în N', în modul descris în teorema de fluxului maxim şi tãieturii minime (Teorema 3.4.8.1).
4. Se determinã o tãieturã minimã a conexiunilor (X, X') în N, dupã modul descris la Corolarul 3.4.8.4.

Bipartiøia cu tãietura minimã poate fi dezechilibratã. Calcularea fluxului maxim defineşte un set de tãieturi minime cu aceeaşi dimensiune a tãiaturii, dar cu ponderi diferite în cele douã partiøii. Se pune problema gãsirii unei tãieturi minime care care este cea mai *r-echilibratã* dintre toate tãieturile definite de un flux maxim, deci gãsirea dintre toate tãieturile minime posibile (X, X') definite de un flux maxim a tãieturii minime astfel încât $|w(X) - r\,w(V)|$ sã fie cât mai apropiat de 0. Această problemã este NP-completã [180].

Prin aplicarea repetatã a tehnicii fluxului maxim şi tãieturii minime pentru divizarea partiøiei celei mai mari se poate obøine în final o bipartiøie echilibratã. Această metodã nu este însã una viabilã pentru partiøionarea circuitelor datoritã complexitãøii sale ridicate (sunt posibile $|V|$ calculãri ale fluxului maxim). Yang şi Wong [180] au elaborat o euristicã pentru determinarea unei bipartiøii r-echilibrate care minimizează numãrul de conexiuni tãiate, numitã bipartiøionare cu echilibrarea fluxului.

Fiind dat un circuit $N = (V, E)$, algoritmul de bipartiøionare cu echilibrarea fluxului selectează în mod aleator o pereche de noduri s şi t din N, şi apoi încearcã gãsirea unei bipartiøii r-echilibrate care separã s şi t, şi care minimizează numãrul de conexiuni

tãiate. Fie *W* ponderea totalã a circuitului *N*. Deoarece nu este necesarã impunerea strictã a criteriului de *r*-echilibrare, se permite devierea ponderii unei componente de la (1 - ε) *rW* la (1 + ε) *rW*. Fiind dat un subcircuit *X* al lui *N*, fie *w*(*X*) ponderea totalã a nodurilor din *X*.

Algoritmul este prezentat în Figura 3.13.

Pasul 1 poate fi implementat prin algoritmul pentru determinarea tãieturii minime a conexiunilor. În pasul 4.2 este necesarã comasarea în *s* a unui nod $v \in X'$ incident pe o conexiune tãiatã, deoarece în caz contrar în urmãtoarea iteraøie în pasul 1 se va alege acelaši set de conexiuni din *C* ca ši tãieturã minimã a conexiunilor. Prin comasarea *v* în *s*, algoritmul poate explora o altã tãieturã a conexiunilor cu un subcircuit *X* mai mare în urmãtoarea iteraøie. În mod similar se procedeazã ši în pasul 5.2.

Un dezavantaj al euristicii fluxului maxim este cã timpul de execuøie este ridicat, datoritã execuøiei iterative a algoritmului pentru determinarea tãieturii minime a conexiunilor. O implementare mai eficientã se poate realiza pe baza observaøiei cã nu este necesar sã se execute calculul pentru fluxul maxim de la fluxul 0 în fiecare iteraøie. Valoarea fluxului din reøea poate fi reøinutã, ši în fiecare iteraøie se poate gãsi un flux suplimentar pentru a satura muchiile punte.

Algorithm *FBB*;
begin
Pas 1. Se alege în mod aleator o pereche de noduri *s* şi *t* din *N*;
Pas 2. Se determină o tăietură minimă a conexiunilor *C* din *N*;
Fie *X* subcircuitul accesibil din *s* prin căi de augmentare în reţeaua de flux, şi *X'* restul circuitului;
Pas 3. **if** (1- ε) $rW \le w(X) \le$ (1+ ε) *rW* **then**
STOP;
Se returnează *C*
endif;
Pas 4. **if** $w(X) <$ (1- ε) *rW* **then**
4.1 Se comasează toate nodurile din *X* în *s*;
4.2 Se comasează în *s* un nod $v \in X'$ adiacent cu *C*;
4.3 **goto** Pas 1
endif;
Pas 5. **if** $w(X) >$ (1+ ε) *rW* **then**
5.1 Se comasează toate nodurile din *X'* în *t*;
5.2 Se comasează în *t* un nod $v \in X$ adiacent cu *C*;
5.3 **goto** Pas 1
endif
end.

Figura 3.13. Algoritmul de bipartiøionare cu echilibrarea fluxului.

În urma experimentelor efectuate în [180] algoritmul de bipartiøionare cu echilibrarea fluxului (*FBB*) a fost comparat cu algoritmul Krishnamurthy [105] ši cu o altã euristicã de tip Kernighan-Lin. Rezultatele obøinute aratã cã prin algoritmul FBB se obøine o bipartiøie la care numãrul de conexiuni tãiate este mai redus în medie cu 24%, respectiv cu 19% faøã de cea obøinutã prin euristicile amintite. Comparativ cu algoritmul *EIG1* bazat pe metode spectrale, descris în secøiunea 3.4.7, rezultatele obøinute prin algoritmul *FBB* sunt mai bune în medie cu 11%.

S-a observat cã în cazul în care timpul de execuøie al algoritmului *FBB* este mai mare decât cel mediu, calitatea soluøiilor generate este foarte redusã [180]. Aceasta se poate explica prin faptul cã dimensiunea tãieturii conexiunilor nu este descrescãtoare cu numãrul de iteraøii. Aceastã proprietate a algoritmului *FBB* este în contrast cu euristica Kernighan-Lin ši cea de cãlire simulatã, la care un timp de execuøie mai mare are

ca efect generarea unei soluøii mai bune. Aceastã proprietate a algoritmului *FBB* indicã un mod de îmbunãtãøire a eficienøei acesteia. Se poate alege o limitã superioarã a timpului de execuøie a algoritmului, oprind execuøia algoritmului atunci când timpul de execuøie depãšește aceastã limitã, relansând algoritmul cu o nouã pereche de noduri *s* și *t*.

Algoritmul *FBB* are o comportare predictibilã în privinøa dimensiunilor celor douã partiøii. Timpul de execuøie și dimensiunea tãieturii sunt funcøii descrescãtoare cu factorul de deviere ε. Alegerea perechii de noduri *s* și *t* ca și configuraøie iniøialã are o influenøã mai micã asupra soluøiei decât alegerea unei bipartiøii iniøiale.

3.4.9 Multipartiționarea

Multipartiøionarea sau partiøionarea cu cãi multiple este o extensie importantã a bipartiøionãrii deoarece asigurã un model natural și de acurateøe mai mare pentru numeroase aplicaøii de partiøionare. Acest tip de partiøionare a fost abordat prin diferite metode: prin cãlire simulatã [105], prin utilizarea vectorilor proprii ale matricilor provenite din lista de conexiuni a circuitului [39], [86], [87], sau prin migrarea grupurilor [105], [155]. Acestea din urmã se concentreazã asupra evaluãrii mutãrilor care determinã îmbunãtãøirea maximã a partiøiei curente. Au fost propuse mai multe modele pentru evaluarea mutãrilor: modelul câštigului cu nivele multiple, modelul probabilistic al conexiunilor intersectate, sau modelul clicii ponderate.

Yeh *et al*. [183] au propus un algoritm de partiøionare cu îmbunãtãøire iterativã care utilizeazã un nou model pentru procesul de mutare. În plus, algoritmul poate utiliza diferite funcøii obiectiv cu aplicaøii în proiectarea circuitelor VLSI.

Un sistem este reprezentat printr-un hipergraf $H(V, E)$, unde $V = \{v_i \mid i = 1, 2, \ldots, n\}$ este setul nodurilor și $E = \{e_u \mid u = 1, 2, \ldots, m\}$ este setul conexiunilor. Fiecare conexiune e_u este un subset al lui V cu cardinalitatea $|e_u| \geq 2$. Fiecare nod v_i are dimensiunea $s(v_i)$. $S(V) = \sum_{v \in V} s(V_i)$ reprezintã dimensiunea hipergrafului H. O partiøionare cu k cãi asigneazã nodurile lui V în k partiøii, cu fiecare partiøie b conøinând un subset nevid V_b al lui V. Se definešte *acoperire* a unei conexiuni e ca fiind 0 dacã e aparøine unei singure partiøii, și f dacã e conecteazã f partiøii, cu $f \geq 2$.

Yeh *et al*. [183] au definit trei funcøii obiectiv diferite.

Funcøia obiectiv 1. Pentru circuitele integrate scopul este minimizarea numãrului maxim de pini de I/E a fiecãrei partiøii (capsule). Fie numãrul de pini de I/E ai partiøiei b notat cu $|\{e_u \mid acoperire(e_u) \geq 2,\ e_u \cap V_b \neq \varnothing\}|$. Funcøia obiectiv se poate exprima astfel:

$$\Psi_{\max IE}: \min \max_{b \in \{1 \ldots k\}} |\{e_u \mid acoperire(e_u) \geq 2,\ e_u \cap V_b \neq \varnothing\}| \tag{3.28}$$

Funcøia obiectiv 2. În cazul proiectãrii arhitecturale la nivel de plãci, este de dorit sã se minimizeze semnalele de interfaøã între diferite circuite. Se notează cu $|\{e_u | acoperire(e_u) \geq 2\}|$ numãrul de conexiuni între circuite. Funcøia obiectiv este de a minimiza acest numãr.

$$\Psi_{net}: \min |\{e_u \mid acoperire(e_u) \geq 2\}| \tag{3.29}$$

Funcøia obiectiv 3. În cazul rutării fizice, complexitatea poate fi redusã prin minimizarea numãrului total de pini de I/E din toate partiøiile. Funcøia obiectiv este

$$\Psi_{pin}: \min \sum_{e_u \in E} acoperire(e_u) \tag{3.30}$$

Pentru fiecare funcøie obiectiv existã restricøia

$$C_m \leq |V_b| \leq C_M \quad \text{pentru fiecare } b = 1, 2, \ldots, k \tag{3.31}$$

unde C_m şi C_M sunt douã constante care indicã limitele dimensiunii fiecãrei partiøii şi $0 < C_m \leq C_M < S(V)$.

Euristica de partiøionare cu cãi multiple propusã de Yeh *et al*. se numeşte algoritmul *Primal-Dual* (PD). Algoritmul constã din trei faze: 1) partiøionare recursivã prin tãietura proporøionalã pentru formarea grupurilor; 2) iteraøie *Primal-Dual* asupra sistemului grupat; 3) iteraøie *Primal-Dual* asupra sistemului original.

Algoritmul este prezentat în Figura 3.14.

```
Algorithm Primal-Dual (H, lim_buclare, C_s);
begin
        H_s = Rcut-recursiv (H, C_s);
        contor = 0;
        partiţie = NULL;
        while (contor < lim_buclare)
                P_s = partiţie iniţială aleatoare a lui H_s;
                P_PD = Iteraţie-Primal-Dual (P_s);
                if (cost (P_PD) < cost (partiţie)) then
                        partiţie = P_PD;
                endif;
                contor := contor + 1;
        endwhile
        Se expandează H_s în H;
        P = Iteraţie-Primal-Dual (partiţie);
end.
```

Figura 3.14. Algoritmul Primal-Dual.

Faza 1: Partiøionare recursivã prin tãietura proporøionalã

Algoritmii de bipartiøionare de tip Kernighan-Lin [76], [105] au dezavantajul cã adesea obøin un minim local atunci când dimensiunea circuitului creşte. În cazul partiøionãrii cu cãi multiple, problema minimului local este mai criticã, deoarece partiøiile multiple cresc în mod semnificativ spaøiul soluøiilor. Un mod obişnuit de a elimina acest dezavantaj este de a se grupa subcircuitele puternic conectate şi apoi de a se condensa aceste grupuri în super-noduri înaintea execuøiei algoritmilor de tip Kernighan-Lin [105].

```
Procedure Rcut-recursiv (H, C_s);
begin
        Ψ = {V};
        do
                Se alege un subset V* ∈ Ψ astfel încât S(V*) = max_{V_j ∈ Ψ} S(V_j);
                Ψ = Ψ - {V*};
                Se aplică algoritmul de partiţionare prin tăietura proporţională
                [175] asupra V* pentru a obţine o tăietură (A, A'), A ⊆ V*;
                Ψ = Ψ ∪ {A, A'};
        while (S(V*) > C_s);
        return;
end
```

Figura 3.15. Partiøionarea recursivã prin tãietura proporøionalã.

Pentru identificarea subcircuitelor puternic conectate se poate utiliza în mod recursiv bipartiøionarea prin tãietura proporøionalã [175]. Aceastã procedurã este ilustratã în Figura 3.15.

Este importantã alegerea corectã a dimensiunii maxime a unui grup (C_s), care determinã numãrul de grupuri utilizate în faza urmãtoare. Dacã C_s este prea mic, numãrul de grupuri generate va fi mare, ceea ce poate micšora performanøele urmãtoarei proceduri de partiøionare. Dacã C_s este prea mare, se pot pierde informaøii de conectivitate de cãtre procedura urmãtoare. În [183] rezultatele cele mai bune s-au obøinut pentru $C_s \approx 0.02 \times S(V)$.

Faza 2: Iteraøia Primal-Dual asupra sistemului grupat

Se va descrie mai întâi modelul propus de Yeh *et al.* [183] pentru selectarea nodurilor care vor fi mutate. În cazul algoritmilor de tip Kernighan-Lin, este mutat un singur nod la un moment dat dintr-o partiøie în cealaltã. Pentru a se permite mutarea simultanã a mai multor noduri conectate, este relaxatã restricøia de mutare a unui singur nod.

Fiind datã o conexiune e_u ši o partiøie b, se definešte *setul critic* al conexiunii e_u faøã de partiøia b ca

$$S_{ub} = \{v \mid v \in e_u \text{ şi } v \in V_b\} \tag{3.32}$$

reprezentând toate nodurile din partiøia b care sunt conectate prin conexiunea e_u. Se definešte de asemenea *setul critic complementar* al conexiunii e_u faøã de partiøia b ca

$$S_{u\bar{b}} = \{v \mid v \in e_u \text{ şi } v \in \overline{V_b}\} \tag{3.33}$$

reprezentând toate nodurile conectate prin conexiunea e_u care nu sunt în partiøia b. Definiøia unei mutãri poate fi reformulatã folosind setul critic šI setul critic complementar. În cazul funcøiei obiectiv Ψ_{maxIE} šI Ψ_{pin}, o mutare asociatã cu o conexiune eu este definitã prin plasarea setului critic S_{ub} într-o partiøie diferitã de V_b. În cazul funcøiei obiectiv Ψ_{net}, o mutare asociatã cu o conexiune eu este definitã prin plasarea setului critic complementar $S_{u\bar{b}}$ în partiøia V_b. Câštigul fiecãrei mutãri este calculat prin evaluarea modificãrii costului datoritã mutãrii setului critic sau al celui critic complementar.

Acest model este numit “model de mutare bazat pe conexiuni”, spre deosebire de cel utilizat în algoritmul Fiduccia-Mattheyses [76], care este numit “model de mutare bazat pe noduri”.

Deši modelul de mutare bazat pe conexiuni contribuie în mai mare mãsurã la îmbunãtãøirea partiøiei curente, acest model este mai complex deoarece sunt implicate mai multe noduri în fiecare mutare. Cele douã modele se pot utiliza în mod alternativ, fiecare aducând anumite îmbunãtãøiri faøã de dualul sãu. Procedura corespunzãtoare este ilustratã în Figura 3.16, unde *Primal* se referã la utilizarea modelului de mutare bazat pe noduri, iar *Dual* se referã la utilizarea modelului de mutare bazat pe conexiuni.

Procedura *Primal* este similarã cu algoritmul Fiduccia-Mattheyses cu urmãtoarea adaptare pentru partiøionarea cu cãi multiple: Pentru fiecare partiøie b, se pãstreazã o listã sortatã de mutãri care deplaseazã noduri din partiøia b în fiecare din celelalte partiøii. Aceastã listã are aceeaši structurã cu cea din algoritmul Fiduccia-Mattheyses [76]. Câštigurile mutãrilor sunt calculate conform cu funcøia obiectiv Ψ_{maxIE}, Ψ_{net} sau Ψ_{pin}.

Procedura *Dual* este similarã cu procedura *Primal* cu excepøia faptului cã toate operaøiile se bazeazã pe conceptul seturilor critice šI critice complementare. Principalele diferenøe sunt urmãtoarele: 1) În locul unui singur nod, se mutã un set critic sau

set critic complementar, în funcøie de tipul funcøiei obiectiv. 2) Operaøia de blocare se efectueazã asupra unei conexiuni, deci dacã setul critic sau cel critic complementar al unei conexiuni a fost mutat, atunci toate mutãrile iniøiate de aceastã conexiune vor fi interzise în continuare. O mutare este *iniøiatã* de o conexiune e_u dacã aceastã mutare este compusã din deplasarea setului critic sau al celui complementar asociat cu e_u.

```
Procedure Iteraţie-Primal-Dual (P);
begin
        P = Primal (P);
        faza_pd = 1;
        do
                if (faza_pd = 0) then
                        P' = Primal (P);
                else if (faza_pd = 1) then
                        P' = Dual (P);
                endif;
                if (cost (P') < cost (P)) then
                        îmbunătăţire = TRUE;
                        P = P';
                        faza_pd = 1 - faza_pd;
                else
                        îmbunătăţire = FALSE;
                endif;
        while (îmbunătăţire = TRUE);
        return;
end
```

Figura 3.16. Procedura de iteraøie Primal-Dual.

Faza 3: Iteraøia Primal-Dual asupra sistemului original

În faza a doua a algoritmului *Primal-Dual* partiøia curentã este optimizatã prin gestiunea grupurilor ca noduri condensate. Este posibil ca mutarea acestor grupuri sã nu permitã optimizarea cu fineøe a partiøiei. În faza a treia, se expandeazã grupurile øi se executã o iteraøie suplimentarã *Primal-Dual*. Prin execuøia acestei faze se obøine o îmbunãtãøire suplimentarã a calitãøii soluøiei.

Complexitatea algoritmului *Primal-Dual* se poate determina analizând complexitatea fiecãrei faze. În faza de partiøionare recursivã numãrul de operaøii executate este de cel mult $O(nd)$, unde n este numãrul de noduri, iar d este numãrul maxim de conexiuni pe nod. Complexitatea procedurii *Primal* este similarã cu cea a algoritmului Fiduccia-Mattheyses, care este $O(nd)$. Deoarece se construiesc partiøii cu k cãi, fiecare mutare a unui nod are k-1 direcøii posibile. Deci, complexitatea procedurii *Primal* este de k-1 ori mai mare, care este $O(k \times nd)$. Dacã p este numãrul maxim de pini pe conexiune, iar m este numãrul total de conexiuni, complexitatea procedurii *Dual* este de $O(mkp^2d^2)$ [183].

3.4.10 Partiționarea prin metode probabilistice

Cele mai multe metode de partiøionare prin îmbunãtãøire iterativã calculeazã câøtigurile datorate mutãrii nodurilor pe baza informaøiilor locale din lista de conexiuni care urmãresc îmbunãtãøirea imediatã a tãieturii. Asemenea metode sunt algoritmul Kernighan-Lin sau variantele Schweikert-Kernighan øi Fiduccia-Mattheyses ale acesteia. Krishnamurthy [105] a sugerat o metodã de calcul a câøtigurilor pe baza unor informaøii mai globale, obøinându-se partiøii de calitate mai bunã. Necesarul de memorie al acestui algoritm este foarte ridicat. Niciuna din aceste metode nu poate însã prevedea cu acurateøe starea viitoare a unei conexiuni.

Dutt şi Deng [69], [70] au propus o metodã pe baze probabilistice pentru calculul câştigurilor, care poate determina implicaøiile globale şi viitoare ale mutării unui nod în orice etapã a procesului de partiøionare. Fiecărui nod u i se asociazã o probabilitate $p(M(u))$ (prescurtatã ca $p(u)$) a evenimentului $M(u)$ ca u sã fie mutat efectiv în celãlalt subset în pasul curent al procesului de partiøionare. Din această probabilitate se calculează câştigurile probabilistice (sau potenøiale) $g(u)$ ale nodurilor, ceea ce furnizează o indicaøie corectã a beneficiului care se obøine în urma mutării acestora în celãlalt subset.

Problema care se pune este modul de obøinere iniøialã a probabilitãøilor $p(u)$ ale nodurilor. Aceste probabilitãøi se calculează din câştigurile nodurilor - cu cât câştigul este mai mare, cu atât este mai mare probabilitatea unui nod de a fi mutat efectiv în celãlalt subset. Această interdependenøã între probabilitãøi şi câştiguri este eliminată prin determinarea unei estimări ale probabilitãøilor $p(u)$ prin una din două metode. În prima metodã, la începutul unei etape se asignează tuturor nodurilor aceeaşi probabilitate p_{init}, de exemplu 0.8. În cea de-a doua metodã, se calculează mai întâi câştigurile deterministice ale nodurilor, similar cu cele calculate prin metoda Fiduccia-Mattheyses. Din aceste câştiguri deterministice se determinã probabilitãøile iniøiale ale nodurilor.

După ce s-au determinat probabilitãøile iniøiale, se calculează câştigurile probabilistice ale nodurilor. Din aceste câştiguri, se recalculează probabilitãøile nodurilor, şi din acestea se obøin câştiguri mai exacte ale nodurilor. Acest proces se repetă pentru un numãr de iteraøii. După terminarea acestui proces iniøial, nodurile cu câştigurile cele mai mari sunt mutate între cele două subseturi, ca şi în cazul altor metode iterative. După fiecare mutare, se actualizează câştigurile şi probabilitãøile nodurilor. De asemenea, la fiecare mutare se calculează câştigul imediat obøinut, care este numãrul de conexiuni eliminate din setul de tãieturã minus numãrul de conexiuni care sunt adãugate în setul de tãieturã prin această mutare. La sfârşitul etapei curente, se efectuează în mod efectiv mutările până în punctul în care se obøine un câştig imediat maxim.

Câştigul probabilistic este util pentru determinarea nodurilor care vor fi mutate şi care vor produce în final îmbunãtãøirea maximã a setului de tãieturã, chiar dacã câştigul imediat al acelei mutări poate fi redus sau chiar negativ. Datoritã unei asemenea mutări, este de aşteptat ca o mutare viitoare sã producã un câştig imediat de valoare mare. Acest mod de determinare a câştigurilor probabilistice, iniøial şi după fiecare mutare, este cheia pentru obøinerea unor performanøe mai bune faøã de metodele de îmbunãtãøire iterativã deterministice sau bazate pe câştiguri imediate [69].

Algoritmul de partiøionare pe baza câştigurilor probabilistice este descris în Figura 3.17.

Se descrie în continuare modul de calcul al câştigurilor probabilistice ale nodurilor din probabilitãøile nodurilor. Pentru fiecare nod, se calculează câştigurile corespunzãtoare fiecãrei conexiuni din care face parte; câştigul total este suma acestor câştiguri ale conexiunilor. O conexiune n_i este numită *blocatã* în V_1 (V_2) dacă fiecare nod din conexiune este blocat în V_1 (V_2). O conexiune este *blocatã în setul de tãieturã* dacã ea este blocată atât în V_1, cât şi în V_2.

Pentru calculul câstigurilor nodurilor există două cazuri.

1) *Conexiunea se aflã în setul de tãieturã*

Fie $u \in V_1$ care face parte din conexiunea n_i, conexiune aflatã în setul de tãieturã. Se notează setul $n_i \cap V_r$ prin $n_{i,r}$, $r = 1, 2$. Fie $n_i^{1\to 2}$ ($n_i^{2\to 1}$) evenimentul prin care n_i este eliminat din setul de tãieturã prin mutarea tuturor nodurilor din $n_{i,1}$ ($n_{i,2}$) în V_2 (V_1). Se defineşte

$p(n_i^{1\to 2} \mid u)$ = (probabilitatea ca n_i sã fie eliminat din setul de tãieturã prin mutarea tuturor nodurilor din $n_{i,1}$ în V_2 dacã u a fost mutat)

$p(n_i^{2\to1} \mid u^c)$ = (probabilitatea ca n_i sã fie eliminat din setul de tãieturã prin mutarea tuturor nodurilor din $n_{i,2}$ în V_1 dacã u nu a fost mutat)

Atunci câştigul $g_{ni}(u)$ al nodului u corespunzãtor conexiunii n_i este definit ca

$$g_{ni}(u) = c(n_i)\,[p(n_i^{1\to2} \mid u) - p(n_i^{2\to1} \mid u^c)] \tag{3.34}$$

unde $c(n_i)$ este costul (ponderea) conexiunii n_i.

Algorithm *PROP* (*H*);
/* *H* este hipergraful de partiţionat */

begin

Pas. 1. Se partiţionează *H* în două subseturi de dimensiuni egale (sau aproape egale) V_1 şi V_2, în mod aleator sau utilizând tehnici de grupare;

repeat

Pas 2. Pentru fiecare nod *u*, se stabileşte $p(u) = p_{init}$, sau se determină $p(u)$ din câştigurile deterministice ale nodurilor;

Pas 3. Pentru fiecare nod, se calculează câştigurile pe baza ecuaţiilor (3.37) şi (3.38), şi probabilităţile $p(u)$;

repeat

Pas 4. Se selectează nodul *u* cu câştigul maxim din oricare subset pentru a fi mutat în celălalt subset dacă condiţia de echilibru este satisfăcută. În caz contrar, se mută nodul *u* cu câştigul maxim pentru care condiţia de echilibru nu este violată;

Pas 5. Se memorează câştigul imediat al mutării curente;

Pas 6. Se blochează toate nodurile mutate în iteraţia curentă, şi se actualizează câştigurile nodurilor vecine care nu sunt blocate;

until (toate nodurile sunt blocate, sau nu mai există mutări pentru care condiţia de echilibru este satisfăcută);

Pas 7. Se calculează sumele câştigurilor imediate ale tuturor mutărilor efectuate şi se memorează maximul G_{max} al acestor sume;

if ($G_{max} > 0$) **then**
if (G_{max} corespunde mutării *p*) **then**
Se efectuează efectiv primele *p* mutări;
endif
else EXIT; **endif**;

until ($G_{max} \le 0$)

end.

Figura 3.17. Algoritmul *PROP* de partiøionare pe baza câştigurilor probabilistice.

Motivul pentru termenul negativ din ecuaøia 3.34 este cã mutarea nodului *u* previne apariøia evenimentului $n_i^{2\to1}$, şi astfel împiedicã posibilitatea eliminării n_i din setul de tãieturã în acest fel. Utilizând probabilitãøi condiøionate şi faptul cã cele mai multe conexiuni dintr-un circuit VLSI au un numãr mic de noduri, se obøine:

$$p(n_i^{1\to2} \mid u) \approx \prod_{u_x \in (n_{i,1} - \{u\})} p(u_x) \tag{3.35}$$

$$p(n_i^{2\to 1}|u^c) \approx \prod_{u_y \in n_{i,2}} p(u_y) \tag{3.36}$$

Se obøine astfel urmãtoarea aproximare a câøtigului $g_{ni}(u)$ [70]:

$$g_{ni}(u) \approx c(n_i)[\prod_{u_x \in (n_{i,1}-\{u\})} p(u_x) - \prod_{u_y \in n_{i,2}} p(u_y)] \tag{3.37}$$

2) *Conexiunea nu se aflã în setul de tãieturã*

Se considerã acum contribuøia la câøtigul nodului u pe care o are conexiunea n_i care nu se aflã în setul de tãieturã, øi care nu este blocatã în subsetul din care face parte, de exemplu V_1. Conexiunea va fi introdusã în setul de tãieturã atunci când u este mutat din V_1 în V_2. Astfel, $g_{ni}(u)$ va fi negativ øi este dat de ecuaøia [70]:

$$g_{ni}(u) \approx -c(n_i)(1 - \prod_{u_x \in n_{i,1}-\{u\}} p(u_x)) \tag{3.38}$$

Dupã ce câøtigurile iniøiale ale fiecãrui nod au fost calculate printr-una din metodele amintite, se calculeazã probabilitãøile lor utilizând o funcøie monoton crescãtoare $p(u) = f(g(u))$ a câøtigurilor acestora.

Existã douã probleme legate de calculul probabilitãøilor. Prima este cã, deoarece nu existã certitudini în privinøa mutãrii nodurilor, este rezonabilã stabilirea unei probabilitãøi maxime $p_{max} < 1$ øi a unei probabilitãøi minime $p_{min} > 0$, interval în care se aflã probabilitãøile tuturor nodurilor. A doua problemã este de a stabili praguri superioare øi inferioare ale câøtigurilor g_{sup} øi g_{inf}, astfel încât toate nodurile cu câøtiguri mai mari sau egale cu g_{sup} vor avea probabilitatea p_{max}, iar cele cu câøtiguri mai mici decât g_{inf} vor avea probabilitatea p_{min}. Motivul pentru stabilirea acestor praguri este cã nodurile cu câøtiguri mari, de exemplu mai mari decât 2, vor fi mutate necondiøionat în celãlalt subset, iar cele cu câøtiguri foarte mici, de exemplu mai mici decât -1, cel mai probabil nu vor fi mutate în etapa curentã. Un exemplu de asemenea funcøie este o funcøie de probabilitate liniarã între $p(u)$ øi $g(u)$ atunci când $g_{inf} \le g(u) \le g_{sup}$.

Se descrie în continuare modul în care se actualizeazã nodurile care nu sunt blocate. Atunci când o conexiune n_i este blocatã în V_2, probabilitatea $p(n_i^{1\to 2})$ este datã de ecuaøia 3.35, deoarece aceastã probabilitate este condiøionatã implicit de mutãrile precedente din V_1 în V_2 ale nodurilor blocate în mod curent în $n_{i,2}$. De asemenea, în acest caz $p(n_i^{2\to 1}) = 0$, deoarece $p(u) = 0$ pentru un nod blocat. Rezultã cã atunci când n_i este blocat în V_2, pentru un nod neblocat $u_x \in n_{i,1}$, $p(n_i^{1\to 2} | u_x) = p(n_i^{1\to 2})/p(u_x)$, øi deci

$$g_{ni}(u_x) = c(n_i) \cdot p(n_i^{1\to 2} | u_x) = c(n_i) \cdot p(n_i^{1\to 2}) / p(u_x) \tag{3.39}$$

Pentru un nod neblocat u_y din $n_{i,2}$, unde n_i este blocat în V_2, utilizând ecuaøia 3.37 øi faptul cã $p(n_i^{2\to 1}) = 0$, se obøine

$$g_{ni}(u_y) = -c(n_i) \cdot p(n_i^{1\to 2} | u_y^c) = -c(n_i) \cdot p(n_i^{1\to 2}) \tag{3.40}$$

Se pot scrie expresii similare pentru cazul în care n_i este blocat în V_1. Aceste ecuaøii sunt utile pentru actualizarea eficientã a nodurilor dupã o mutare.

Dupã mutarea unui nod u, de exemplu din V_1 în V_2, se actualizeazã mai întâi $p(n_i^{1\to 2})$ øi $p(n_i^{2\to 1})$ pentru fiecare conexiune n_i din care face parte u. Se actualizeazã apoi câøtigurile tuturor nodurilor care fac parte din fiecare asemenea conexiune n_i (vecinii nodului u), conform ecuaøiilor 3.39 øi 3.40. Apoi se seteazã $p(u) = 0$, indicând faptul cã u este blocat. În final, se actualizeazã câøtigurile unui numãr redus de noduri, de exemplu 5, din cele clasate primele în fiecare subset, utilizând ecuaøiile 3.37 øi 3.38.

Aceastã operaøie este necesarã deoarece unele din aceste noduri pot fi vecini ai vecinilor nodului *u*, ale cãrui probabilitãøi au fost actualizate. Operaøia se executã numai pentru nodurile clasate primele, care sunt candidate pentru urmãtoarea mutare, timpul necesar fiind mult mai redus decât pentru o actualizare completã.

Dacã *n* este numãrul de noduri ale circuitului, *p* este numãrul mediu de pini ai unui nod, *q* numãrul mediu de pini pe conexiune, $d = p(q - 1)$ va fi numãrul mediu de vecini ai unui nod. Se defineşte $m = pn = qe$ numãrul total de pini ai circuitului, unde *e* este numãrul de conexiuni. Complexitatea în privinøa timpului de execuøie pentru o etapã a algoritmului PROP este $O(nd \log n) = O(mq \log n)$. Pentru circuitele VLSI, *q* este o constantã de valoare micã, de exemplu 4, şi deci complexitatea este $O(m \log n)$ [69].

3.5 Partiționarea pentru circuitele FPGA

Atunci când dimensiunea circuitului proiectat este prea mare pentru a fi configurat într-un singur circuit FPGA, circuitul trebuie partiøionat în subcircuite. Partiøionarea circuitelor FPGA multiple trebuie sã satisfacã restricøii suplimentare asupra dimensiunii subcircuitelor şi a numãrului terminalelor de I/E. Pentru a øine cont de restricøiile suplimentare, au fost publicaøi un numãr de algoritmi de partiøionare pentru circuitele FPGA [40], [89], [101], [176].

McDermith a descris o metodã de partiøionare de jos în sus pentru circuitele FPGA, în care funcøiile sunt selectate şi plasate pe un cip, una câte una. Deoarece aceastã metodã plaseazã în mod constructiv fiecare funcøie, soluøia nu este, în general, apropiatã de cea optimã. Woo şi Kim [176] au publicat un algoritm de îmbunãtãøire iterativã pentru circuitele FPGA. În fiecare etapã, celulele sunt mutate între blocuri şi este aleasã partiøia cea mai bunã. Mutãrile sunt efectuate astfel încât sã se reducã numãrul de pini ai subcircuitelor sursã şi/sau destinaøie.

Kuznar *et al.* au descris o metodã de partiøionare a unui circuit de dimensiuni mari într-o colecøie de subcircuite care pot fi implementate prin dispozitive selectate dintr-o bibliotecã datã. Fiecare dispozitiv din bibliotecã poate avea un cost, dimensiune şi numãr de terminale diferite.

Kim şi Shin [101] au elaborat un algoritm de partiøionare orientat pe performanøe, în scopul minimizãrii întârzierilor pe cãile critice. Ei au studiat de asemenea partiøionarea circuitelor pentru emularea logicã prin circuite FPGA, elaborând un algoritm de partiøionare a unui circuit care este emulat printr-o reøea de circuite FPGA. Pentru a asigura rutarea circuitului utilizând reøeaua FPGA, algoritmul øine cont de rutabilitate şi de întârzierea unei cãi estimatã prin numãrul de conexiuni între capsule.

Chan *et al.* [40] au descris o metodã spectralã de multipartiøionare în care scopul principal este de a obøine subcircuite rutabile, maximizând în acelaşi timp utilizarea circuitului FPGA. A fost dezvoltatã o teorie de predicøie a rutabilitãøii subcircuitelor, înainte de partiøionare.

Au fost publicaøi de asemenea algoritmi de partiøionare pentru circuite FPGA bazate pe structuri de tip PLA. Hasan *et al*. [89] au elaborat un algoritm care partiøionează în mod eficient o logicã într-un numãr minim de grupuri, astfel ca fiecare din acestea sã poatã fi implementat într-un bloc de tip PLA al circuitului FPGA.

3.5.1 Partiționarea ierarhică pentru circuite FPGA multiple

În aceastã secøiune este descris un algoritm de partiøionare orientat pe performanøe pentru implementarea sistemelor prin circuite FPGA multiple [101]. Intrarea furnizatã algoritmului este o listã de conexiuni a unor blocuri ale unui circuit FPGA destinaøie. De exemplu, circuitele Xilinx au un singur tip de blocuri, numite CLB.

Fie un circuit cu n noduri (sau blocuri CLB) şi fie V setul nodurilor, iar E setul conexiunilor, unde o conexiune $e \in E$ cuprinde două sau mai multe elemente din V. Problema de partiøionare pentru circuite FPGA multiple poate fi definită formal după cum urmează.

Fiind dat un hipergraf $H(V, E)$, o funcøie de cost f şi un set de I circuite FPGA cu restricøiile de dimensiune şi de pini S_1, …, S_I, respectiv P_1, …, P_I, trebuie să se găsească o mapare o: $V \rightarrow \{1, 2, \ldots, I\}$ astfel încât

i. $a_i \leq S_i$ pentru fiecare i = 1, 2, …, I, unde $a_i = \{|v \mid o(v) = i, v \in V\}|$. De notat că dimensiunea fiecărui nod (CLB) este 1.

ii. $b_i \leq P_i$ pentru fiecare i = 1, 2, …, I, unde b_i este numărul de conexiuni dintre un nod din subcircuitul i şi un alt nod din subcircuitul $j \neq i$.

iii. Funcøia de cost f este minimizată.

Metodele de partiøionare nu øin cont de obicei de întârzierile conexiunilor. Unul din principalele dezavantaje ale circuitelor FPGA faøă de circuitele ASIC este că viteza acestora este mult mai redusă datoritã programabilitãøii. De aceea, întârzierile semnalelor trebuie luate în considerare, iar întârzierile excesive trebuie penalizate în timpul partiøionării.

În algoritmul elaborat de Kim şi Shin, funcøia de cost include atât dimensiunea tăieturii, cât şi întârzierea introdusă de conexiune. Sunt estimate întârzierile pe căile critice, şi se øine cont de acestea în timpul grupării şi partiøionării. Cea mai bună cale de a minimiza întârzierile pe o cale criticã este de a se plasa toate nodurile căii respective în acelaşi circuit. De aceea, se pot grupa mai multe elemente apropiate conectate, grupul respectiv fiind considerat ca un obiect în timpul primei faze a partiøionării. Măsura de apropiere dintre două elemente (noduri) conectate este reprezentată de o pondere a unei muchii.

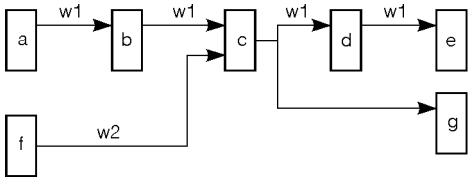

Figura 3.18. Un exemplu de ponderare a căilor.

De exemplu, în Figura 3.18, fie (a, b, c, d, e) calea cea mai criticã, cu întârzierea d_1, muchiile aparøinând acestei căi având ponderea w_1. Dacă a doua cale criticã este (f, c, d, e), cu întârzierea d_2, atunci muchia (f, c) are ponderea w_2, unde $w_1 > w_2$. Ponderile sunt determinate astfel încât să normalizeze întârzierile căilor, după cum se aratã în Figura 3.19.

Metoda de partiøionare încearcă să minimizeze costul, satisfăcând în acelaşi timp restricøiile asupra numărului de terminale de I/E şi asupra dimensiunii circuitelor. Algoritmul de partiøionare constă din două faze. În prima fază, cea de partiøionare iniøialã, se execută o partiøionare ierarhicã bazatã pe grupare, care este modificată din [156]. Funcøia obiectiv a primei faze este suma ponderată a dimensiunii tăieturii şi a întârzierii maxime. Deoarece pentru minimizarea funcøiei obiectiv sunt mutate în mod iterativ noduri dintr-un subset în altul, ca şi în cazul algoritmului Fiduccia-Mattheyses [76], rezultatul poate fi dependent de partiøia iniøialã [156], [183]. Pentru a elimina aceastã dependenøă, se partiøionează grupurile de *N1* ori pornind de la partiøia iniøialã

generatã aleator, şi apoi se alege rezultatul cel mai bun. Acest rezultat este expandat şi optimizat din nou pentru a finaliza faza de partiøionare iniøialã.

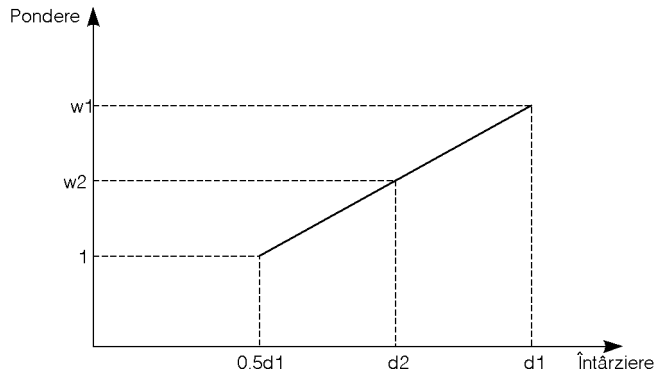

Figura 3.19. Ponderea conexiunilor în funcøie de întârzierea cãilor.

În faza a doua, cea de îmbunãtãøire a partiøiei, se îmbunãtãøeşte partiøia obøinutã în prima fazã pentru a satisface toate restricøiile datorate circuitelor FPGA date. Algoritmul de partiøionare este prezentat în Figura 3.20.

```
Algorithm Part_FPGA_mult;
begin
        /* Partiţionare iniţială */
        Se generează grupuri;
        Se execută partiţionarea grupurilor de N1 ori;
        Se expandează cel mai bun rezultat obţinut;
        Se partiţionează blocurile CLB;

        /* Îmbunătăţirea partiţiei */
        while (nr_iteraţie < 3 sau partiţia este îmbunătăţită în precedentele 3 iteraţii)
                îmbun_part_curentă ();
        endwhile
end.
```

Figura 3.20. Algoritmul de partiøionare pentru circuite FPGA multiple.

Pentru generarea grupurilor şi partiøionarea blocurilor CLB se utilizează metoda de partiøionare cu forøarea gradualã a restricøiilor GCEP [156]. La început, restricøiile de dimensiune sunt relaxate şi dimensiunile subseturilor partiøionate pot fi puternic dezechilibrate. Aceasta permite mutări foarte flexibile ale nodurilor între subseturi. Dupã fiecare etapã de optimizare, restricøiile de dimensiune sunt forøate în mod gradual, astfel încât în etapa finalã ele devin cele dorite. Astfel algoritmul GCEP efectueazã o cãutare într-un spaøiu mai larg al soluøiilor.

Algoritmul utilizat pentru partiøionarea grupurilor şi a blocurilor CLB este descris în Figura 3.21. O celulã este un grup de blocuri CLB pentru partiøionarea grupurilor şi este un bloc CLB pentru partiøionarea blocurilor CLB. *K* este numãrul de circuite (sau grupuri).

Îmbunãtãøirea partiøiei este executatã pe baza câştigului (reducerii costului) în urma mutării unui bloc CLB de la un circuit la un altul. Mutările sunt efectuate în ordinea descrescãtoare a câştigurilor. Violările restricøiilor asupra numărului de blocuri CLB şi de pini sunt reflectate în cost prin adãugarea unor termeni de penalizare. Atunci când existã violări rãmase la sfârşitul unei iteraøii, valorile de penalizare pentru

restricøiile violate sunt mărite. Dacă rezultatul violat nu este îmbunătăøit în timpul a trei iteraøii consecutive, algoritmul se termină.

Funcøia de cost care cuprinde şi termenii de penalizare se poate scrie astfel:

$$cost = dim\_tăietură + r \cdot cost\_întârziere + \alpha \cdot depăşire\_CLB + \beta \cdot depăşire\_pin$$

unde r este un factor de ponderare. Valorile lui α şi β sunt ajustate în mod dinamic. Ultimii doi termeni devin zero atunci când restricøiile asupra numărului de blocuri CLB şi a numărului de pini sunt satisfăcute.

Algoritmul de îmbunătăøire a partiøiilor este similar cu cel de partiøionare iniøială. Diferenøele constau din următoarele: 1) restricøiile de dimensiune nu sunt relaxate; 2) α este mărit cu unu dacă numărul de blocuri CLB este violat pentru un subcircuit (grup), iar β este mărit cu unu dacă numărul de pini este violat, după fiecare execuøie a procedurii de îmbunătăøire; 3) blocurile CLB sunt mutate numai dacă câştigul este pozitiv; 4) bucla principală de optimizare se termină atunci când numărul de violări sau costul nu sunt îmbunătăøite în timpul a trei iteraøii consecutive.

```
Algorithm GCEP;
begin
        partiționare_inițială;
        evaluare_câştig_inițial;
        grup_curent = 1;
        echilibru = ECHIL_INIT;  /* ECHIL_INIT = 0.5 */
        dim_dif = dim_med × echilibru;
        for (TRUE) do
                /* Istoria mutărilor şi câştigurile sunt memorate în lista_mutări */
                /* Mutare în exterior cât timp dimensiunea grup_curent
                   nu este mai mică decât (dim_med - dim_dif) */
                move_ext (grup_curent, dim_dif);
                /* Mutare în interior cât timp dimensiunea grup_curent
                   nu este mai mare decât (dim_med + dim_dif) */
                move_int (grup_curent, dim_dif);
                /* Găseşte punctul optim din lista_mutări şi anulează
                   mutările efectuate după acest punct */
                undo_move;
                grup_curent ++;
                /* Dacă o iterație este terminată */
                if (grup_curent == K + 1) then
                        grup_curent = 1;
                        echilibru = echilibru × RATA_REDUCERE;
                                /* RATA_REDUCERE = 0.9 */
                        if (echilibru ≤ ECHIL_FINAL) then
                                /* ECHIL_FINAL = 0.2 */
                                break;    /* Partiționare terminată */
                        endif;
                        dim_dif = dim_med × echilibru;
                        if (dim_dif  < dim_med × ECHIL_FINAL) then
                                dim_dif  = dim_med × ECHIL_FINAL;
                        endif
                endif
        endfor
end.
```

Figura 3.21. Algoritmul de partiøionare cu forøarea gradualã a restricøiilor.

3.5.2 Partiționarea pentru circuite FPGA cu structuri de tip PLA

Maparea unor ecuaøii booleene într-un numãr minim de circuite este o sarcinã dificilã. O parte a acestei probleme o reprezintã partiøionarea logicii în numãrul minim de structuri PLA (*Programmable Logic Array*) de dimensiune fixã. Existã un numãr mic de lucrãri legate de partiøionarea logicii pentru arhitecturi de tip PLA. Hasan *et al.* [89] au elaborat o metodã pentru multipartiøionarea unei funcøii logice cu ieșiri multiple într-un numãr minim de subfuncøii pentru maparea în structuri PLA de dimensiune fixã sau blocuri funcøionale ale unui circuit FPGA.

Fiind dat un set de ecuaøii booleene, problema este de a le partiøiona într-un numãr minim de grupuri, astfel încât fiecare din acestea sã poatã fi implementat într-un bloc funcøional al circuitului. Obiectivul este de a se minimiza numãrul de blocuri funcøionale utilizate de mapare.

Fiecãrei funcøii de ieșire i se asociazã un cost, reprezentând restricøiile pe care le impune pentru algoritmul de partiøionare. Restricøiile pentru blocurile funcøionale sunt numãrul limitat de intrãri și termeni produs partajaøi de o ieșire. Costul se exprimã ca o combinaøie ponderatã a acestor restricøii. Astfel, costul C_n al ieșirii n se calculeazã astfel:

$$C_n = W_i I_n + W_p P_n$$

unde W_i și W_p sunt ponderile restricøiilor pentru numãrul de intrãri, respectiv pentru numãrul termenilor produs. I_n și P_n sunt numãrul de intrãri, respectiv numãrul de termeni produs partajaøi de ieșirea n. Din experimente rezultã cã trebuie sã se aleagã o valoare mai mare pentru ponderea W_i, deoarece aceasta are tendinøa sã grupeze ieșirile cu intrãri comune și sã creascã numãrul termenilor produs partajaøi.

Urmãtoarele trei condiøii determinã dacã o funcøie de ieșire poate fi implementatã într-un bloc funcøional.

- Fiecare bloc funcøional are un numãr limitat de linii de ieșire. Dacã numãrul liniilor de ieșire dintr-un bloc este O_m, blocul poate implementa numai O_m ieșiri.
- Fiecare bloc funcøional are un numãr limitat de linii de intrare. Fie numãrul liniilor de intrare ale unui bloc funcøional I_m. Dacã ieșirile curente din bloc utilizeazã x linii de intrare, o nouã ieșire poate fi implementatã în acel bloc numai dacã necesitã un numãr de noi linii de intrare $\leq (I_m - x)$. Noile intrãri sunt cele care nu sunt conectate deja la blocul în cauzã.
- Fiecare bloc funcøional poate avea numai P_m termeni produs partajaøi. Dacã ieșirile existente în blocul funcøional utilizeazã y termeni produs partajaøi, o nouã ieșire poate fi implementatã în acel bloc numai dacã necesitã un numãr de noi termeni produs $\leq (P_m - y)$.

O nouã ieșire poate fi implementatã într-un bloc funcøional numai dacã satisface toate cele trei condiøii.

Pentru minimizarea numãrului total de termeni produs utilizaøi de o funcøie se pot utiliza utilitare de minimizare logicã, de exemplu *Espresso* (dezvoltatã la Universitatea din California, Berkeley). Minimizarea întregii funcøii înainte de partiøionare poate face însã imposibilã utilizarea termenilor partajaøi care sunt generaøi astfel, deoarece distribuøia ieșirilor între diferite blocuri va necesita duplicarea termenilor produs partajaøi. Minimizarea efectuatã dupã partiøionare este inutilã, deoarece ieșirile vor putea fi implementate în blocurile respective chiar și fãrã minimizare. Acestã problemã a fost rezolvatã în [89] prin urmãtorul procedeu:

Se începe prin efectuarea unei partiøionãri de tip greedy cu restricøii relaxate, pentru a se obøine blocurile partiøiei iniøiale. Prin partiøionare de tip greedy se înøelege aici alegerea setului cel mai mare de ieșiri care pot fi implementate într-un bloc. Re-

stricøiile fiind relaxate, este posibil ca $O_i > O_m$ şi $P_i > P_m$, unde O_i şi P_i sunt parametrii care definesc dimensiunea blocului funcøional. Iniøial se alege un numãr mai mare de ieşiri decât cel maxim implementabil, şi se minimizează acel grup de ieşiri. Se aleg apoi O_m ieşiri din grupul minimizat. Blocurile partiøiei iniøiale conøin un numãr mai mare de termeni produs partajaøi. Se aşteaptã ca numãrul termenilor produs partajaøi să scadã în timpul procedurii de minimizare.

Pentru procedura de partiøionare de tip greedy se considerã urmãtorul set de parametri pentru dimensiunea unui bloc funcøional: numãrul liniilor de ieşire *O*, numãrul liniilor de intrare *I*, şi numãrul termenilor produs partajaøi disponibili *P*. Procedura încearcã să selecteze *O* ieşiri care pot fi implementate într-un bloc funcøional. Iniøial, se selectează prima ieşire din baza de date şi se asignează primului bloc.

Baza de date este aranjatã în ordine descrescãtoare a costurilor ieşirilor; astfel, ieşirea cea mai costisitoare va fi asignatã primului bloc. Procedura selectează apoi urmãtoarea ieşire cea mai costisitoare din baza de date şi verificã dacã această ieşire satisface restricøiile pentru numãrul de intrãri şi numãrul termenilor produs. În caz afirmativ, ieşirea va fi asignatã la acel bloc. Procedura se terminã atunci când a plasat *O* ieşiri în primul bloc sau când nu mai existã noi ieşiri care pot fi asignate acelui bloc.

```
Procedure rafinare (O, I, P, bloc);
begin
Pas 1.   Se elimină ieşirea cea mai puţin costisitoare;
Pas 2.   Se încearcă adăugarea unor noi ieşiri;
Pas 3.   if (nr_ieşiri ≤ n) then
                Se înlocuieşte ieşirea din bloc;
                Se elimină următoarea ieşire cea mai puţin costisitoare;
                if (următoarea ieşire cea mai puţin costisitoare =
                   ieşirea cea mai costisitoare) then
                        Se înlocuieşte ieşirea din bloc;
                        goto Pas 6;
                endif;
         endif;
Pas 4.   if (nr_ieşiri > n) then
                Se elimină ieşirea în mod permanent din această partiţie;
         endif;
Pas 5.   if (nr_ieşiri = O) then
                goto Pas 6;
         else
                goto Pas 1;
         endif;
Pas 6.   return;
end
```

Figura 3.22. Procedura de rafinare.

Dupã procedura de partiøionare de tip greedy se executã o procedurã de rafinare. Aceasta are urmãtorii parametri de intrare: *O*, *I*, *P*, şi un bloc valid de ieşiri (o partiøie). Un bloc valid de ieşiri este cel care poate fi implementat într-un bloc funcøional al circuitului. Presupunem cã primul bloc are *n* ieşiri: O_1, O_2, …, O_n, unde $n < O$. Aceste ieşiri sunt sortate în ordinea descrescãtoare a costului. Procedura presupune cã o ieşire în această partiøie utilizează toate resursele, prevenind asignarea altor ieşiri la acel bloc. Se încearcã eliminarea acelei ieşiri din bloc în felul urmãtor: Se eliminã temporar ieşirea cea mai puøin costisitoare, O_n, din bloc, şi se încearcã potrivirea unor noi ieşiri în bloc. În acest caz, pot apare douã situaøii:

- Nu se poate gãsi nici o nouã ieşire care poate fi potrivitã în acel bloc; deci, existã acum (n - 1) ieşiri în blocul respectiv. Aceasta înseamnã cã ieşirea O_n nu a prevenit asignarea unor noi ieşiri la acel bloc. De aceea se plasează înapoi ieşirea O_n în blocul funcøional, se eliminã temporar ieşirea O_{n-1}, şi se încearcã din nou potrivirea unor noi ieşiri în bloc. Procedura se continuã pânã când se ajunge la ieşirea O_1 (cea mai costisitoare din bloc). Aceasta nu se eliminã din bloc, şi procedura se terminã.
- Se poate gãsi o nouã ieşire care poate fi potrivitã în acel bloc; deci, existã acum un numãr $\geq n$ de ieşiri în bloc. Aceasta înseamnã cã ieşirea care a fost eliminatã temporar va fi eliminatã permanent din bloc. Se revine apoi la primul pas al procedurii.

Procedura se terminã dacã existã numãrul maxim de ieşiri permise în bloc sau dacã s-a încercat eliminarea tuturor ieşirilor cu excepøia celei mai costisitoare. Procedura de rafinare este descrisã în Figura 3.22.

Întregul algoritm de partiøionare poate fi descris astfel:

1. Se executã *partiøionare_greedy* (O_i, I_m, P_i);
2. Dacã $n < O_i$ se apeleazã procedura *rafinare*;
3. Se minimizează ieşirile selectate;
4. Se selectează O_m ieşiri utilizând *partiøionare_greedy* şi *rafinare*, şi aceste ieşiri nu vor fi luate în considerare în continuare;
5. Se continuã cu pasul 1 pânã când nu mai existã ieşiri care trebuie partiøionate;
6. Se executã *rafinare_finalã*.

După sortarea tuturor ieşirilor în ordine descrescãtoare a costurilor, algoritmul de partiøionare începe cu pasul de partiøionare de tip greedy. Procedurii de partiøionare greedy i se transmit urmãtorii parametri: $O = O_i$, $I = I_m$, $P = P_m$, unde $O_i > O_m$ şi $P_i > P_m$. Restricøiile au fost deci relaxate în aceastã etapã pentru a executa pasul de minimizare. La sfârşitul pasului de partiøionare greedy, se testează numãrul ieşirilor din primul bloc. Dacã acest numãr este mai mic decât O_i, se executã procedura de rafinare. Apoi se minimizează ieşirile astfel generate. Pot exista însã mai mult de O_m ieşiri în acest bloc. De aceea se executã din nou procedurile de partiøionare greedy şi rafinare asupra acestui grup mai mic de ieşiri. Se utilizează parametrii $O = O_m$, $I = I_m$ şi $P = P_m$ pentru a obøine primul bloc care poate fi implementat într-un bloc funcøional.

Se marchează ieşirile din primul bloc ale partiøiei ca fiind eliminate, şi se repetã întreaga procedurã pentru ieşirile rãmase. Cele O_i - O_m ieşiri, care nu sunt asignate la primul bloc, nu sunt marcate ca fiind eliminate. Această procedurã se repetã pânã când toate ieşirile vor fi plasate într-o partiøie validã. În această etapã se apelează procedura de rafinare finalã, care încearcã mutarea unor ieşiri dintr-un bloc în altul pentru a creşte densitatea logicã a blocurilor.

Algoritmul descris executã partiøionarea cu minimizarea logicii, optimizând utilizarea blocurilor PLA disponibile în cadrul circuitului. Acest algoritm a fost elaborat pentru circuitele Xilinx din seria 7200, dar poate fi utilizat pentru orice circuit cu o arhitecturã similarã de tip PLA.

3.6 Metode neconvenøionale de partiøionare

În literaturã au fost raportate un numãr relativ redus de soluøii ale problemei de partiøionare a circuitelor, obøinute prin tehnici neconvenøionale, ca algoritmi ge-

netici, automate de învãøare sau reøele neuronale. Saab şi Rao [145], [189] au utilizat metodologia de evoluøie stohasticã pentru soluøionarea problemei de bipartiøionare la care nodurile circuitului au dimensiuni diferite. Aceeaşi autori au extins algoritmul de bipartiøionare prin evoluøie stohasticã pentru problema de multipartiøionare, în care diferitele subseturi ale partiøiei cu *k* cãi sunt considerate simultan, şi nu douã câte douã. În secøiunea 3.6.1 se prezintã un algoritm general care utilizeazã metodologia de evoluøie stohasticã, şi apoi adaptarea acestui algoritm pentru soluøionarea problemelor de bipartiøionare şi multipartiøionare a circuitelor.

Bui şi Moon [31], [32] au publicat un algoritm genetic hibrid pentru partiøionarea grafurilor, care conøine şi o euristicã rapidã de îmbunãtãøire localã. La începutul algoritmului se executã o fazã de preprocesare a schemelor, care reordoneazã poziøiile vârfurilor în cadrul cromozomilor, cu scopul de a îmbunãtãøi posibilitãøile algoritmului de cãutare în spaøiul soluøiilor. Algoritmul este descris în secøiunea 3.6.2, care prezintã de asemenea terminologia utilizatã de algoritmii genetici şi structura generalã a unui asemenea algoritm.

Oommen şi St. Croix [129] au utilizat automatele de învãøare pentru partiøionarea grafurilor. Aceştia au propus o metodã în care problema de partiøionare a grafurilor este rezolvatã prin considerarea acesteia ca o problemã de partiøionare a obiectelor. În secøiunea 3.6.3 este prezentat un automat de învãøare pentru partiøionarea grafurilor.

Yih şi Mazumder au utilizat reøelele neuronale pentru bipartiøionarea circuitelor [146]. Bipartiøionarea este reprezentatã sub forma stãrilor reøelei neuronale. Criteriul de selecøie a componentelor care îşi vor modifica poziøia este descris în termenii conexiunilor neuronilor şi a ponderilor corespunzãtoare între neuroni. Este utilizat paralelismul masiv al reøelelor neuronale, rezultatele obøinute fiind comparabile cu cele ale euristicii Fiduccia-Mattheyses. În capitolul 4 se va prezenta utilizarea reøelelor neuronale pentru rezolvarea problemei de plasare a componentelor.

3.6.1 Partiționarea prin evoluție stohastică

Metodologia de evoluøie stohasticã (*Stochastic Evolution* - *SE*) utilizeazã analogia dintre tehnicile de îmbunãtãøire iterativã şi evoluøia biologicã, executând paşi selectaøi în mod arbitrar. În evoluøia biologicã, speciile eliminã unele din caracteristicile nedorite ale vechii generaøii pe mãsurã ce ele evoluează de la o generaøie la alta. În acest proces, fiecare membru al speciei decide sã reøinã caracteristicile sale în mediul curent sau sã le modifice. Atunci când se utilizează acestã metodã, soluøia curentã a problemei de optimizare este consideratã ca fiind generaøia curentã. Caracteristicile nedorite al soluøiei curente sunt identificate şi eliminate pentru a genera o soluøie mai bunã.

Algoritmii *SE* aparøin clasei euristicilor adaptive, ca şi cãlirea simulatã. Un algoritm adaptiv utilizează un set de parametri de control care sunt modificaøi fie de utilizator, fie de algoritmul însuşi. Astfel, algoritmul se adaptează problemei particulare. Adaptarea parametrilor afectează calitatea soluøiei rezultate. Pentru a se utiliza evoluøia stohasticã, trebuie definitã o funcøie de cost care mãsoarã calitatea soluøiei. Spre deosebire de cãlirea simulatã, funcøia de cost în cazul evoluøiei stohastice poate fi sub forma unui vector, şi se poate utiliza o relaøie de ordonare pentru a se compara diferiøi vectori de cost.

În Figura 3.23 se prezintã un algoritm general de tip *SE* [189]. Acest algoritm poate fi aplicat pentru o mare varietate de probleme de combinatoricã.

Parametrii de intrare ai algoritmului *SE* constau dintr-o soluøie iniøialã S_0, o valoare iniøialã a parametrului de control p_0, şi un parametru *R* care controlează numãrul de iteraøii. Algoritmul reøine soluøia cu costul minim dintre cele produse de funcøia *Perturb*. De fiecare datã când se gãseşte o soluøie cu un cost mai mic decât soluøia

curentã cea mai bunã, algoritmul decrementează contorul prin *R*, recompensându-se astfel prin creşterea numărului de iteraøii.

În procesul de evoluøie al speciilor biologice, fiecare caracteristicã a unei specii din generaøia curentã trebuie sã demonstreze cã mediul acesteia nu trebuie schimbat în generaøia urmãtoare. Funcøia *Perturb* implementează această caracteristicã prin cerinøa ca fiecare nod al circuitului sã demonstreze cã nu trebuie sã-şi schimbe subsetul din care face parte în urmãtoarea partiøie generatã de algoritm.

```
Algorithm SE (S0, p0, R);
begin
        S = S0;                                  /* soluția inițială */
        Sbest = S;
        Cbest = Cost (S);
        p = p0;                                  /* inițializare parametru de control */
        contor = 0;
        repeat
                Cpre = Cost (S);
                S = Perturb (S, p);
                Ccurent = Cost (S);
                Update (p, p0, Cpre, Ccurent);
                if (Ccurent < Cbest) then
                        Sbest = S;               /* salvează soluția cea mai bună */
                        Cbest = Ccurent;
                        contor = contor - R;
                else
                        contor = contor + 1;
                endif;
        until (contor > R);
        return (Sbest);
end.
```

Figura 3.23. Structura unui algoritm de evoluøie stohasticã.

Funcøia *Perturb* scanează nodurile circuitului într-o anumitã ordine. Fie $P = (V_1, \ldots, V_k)$ partiøia curentã atunci când este scanat nodul *i*. În această etapã, este sugeratã o mutare constând din transferul nodului *i* din subsetul sãu curent *P* într-un nou subset, şi se evaluează un câştig *GAIN*(*i*) pentru această mutare. Fie *P'* partiøia rezultatã astfel. Funcøia *Perturb* decide dacã acceptã sau nu mutarea sugeratã. Această decizie este luatã în mod stohastic cu ajutorul unui parametru de control non-pozitiv *p*. Valoarea *GAIN*(*i*) este comparatã cu un întreg aleator *r* cu valori în intervalul [*p*, 0]. Dacã *GAIN*(*i*) > *r*, partiøia *P'* înlocuieşte partiøia curentã *P*; în caz contrar se reøine partiøia curentã *P*. După scanarea tuturor nodurilor, *Perturb* poate decide inversarea sau schimbarea unora din ultimele mutãri acceptate.

Procedura *Update* decrementează parametrul de control *p* ori de câte ori se suspectează faptul cã algoritmul este blocat într-un minim local. Fie C_{pre} şi C_{curent} costul partiøiei curente înainte, respectiv după apelul funcøiei *Perturb*. Costul unei partiøii poate fi un vector. Dacã $C_{pre} = C_{curent}$, parametrul *p* este decrementat cu 1; în caz contrar, *p* este resetat la valoarea sa iniøialã p_0. Valoarea iniøialã p_0 este de obicei un întreg negativ apropiat de 0, de exemplu -1.

Parametrul *R* controlează numărul de iteraøii, şi are rolul numărului aşteptat de iteraøii necesare până când se gãseşte o partiøie care este mai bunã decât cele gãsite până în acel moment. Dacã o îmbunãtãøire apare după *contor* < *R* iteraøii, *contor* este decrementat cu *R*. Astfel, algoritmul va mai putea executa încã *R* - *contor* iteraøii. În consecinøã, dacã *T* este numărul total de iteraøii ale algoritmului, numărul de îmbunãtãøiri întâlnite este *T*/*R*.

3.6.1.1 Bipartiționarea prin evoluție stohastică

De obicei, algoritmii de bipartiøionare nu pot trata în mod eficient cazul nodurilor cu dimensiuni diferite. Se prezintă în continuare adaptarea algoritmului *SE* pentru soluøionarea problemei de bipartiøionare la care nodurile circuitului au dimensiuni diferite [145], [189]. Acest algoritm, numit *2_SE*, poate fi complet specificat prin indicarea partiøiei iniøiale și a funcøiei *Perturb*.

Generarea partiøiei iniøiale este prezentată în Figura 3.24.

Fiind dată o partiøie iniøială (V_1, V_2), mutarea asociată cu fiecare nod i este reprezentată de transferul acestui nod din subsetul său curent în subsetul complementar al partiøiei. De exemplu, dacă $i \in V_1$, mutarea asociată cu i determină noua partiøie (V_1 - {i }, $V_2 \cup$ {i }). Câștigul *GAIN*(i) asociat cu fiecare nod i este reducerea dimensiunii tăieturii atunci când este executată mutarea asociată cu nodul i. Câștigul unui nod poate fi negativ sau pozitiv.

```
Procedure Part_init;
begin
        Fie l1, …, ln nodurile sortate în ordinea descrescătoare a dimensiunii;
        V1 = V2 = ∅;
        S (V1) = S (V2) = 0;
        for (i = 1 to n) do                    /* n este numărul de noduri */
                if (S (V1) < S (V2)) then
                        V1 = V1 ∪ {li};
                        S (V1) = S (V1) + s (li);
                else
                        V2 = V2 ∪ {li};
                        S (V2) = S (V2) + s (li);
                endif;
        endfor;
        return (V1, V2);
end
```

Figura 3.24. Generarea partiøiei iniøiale pentru algoritmul *2_SE*.

Funcøia *Perturb* constă din doi pași: selecøie și interschimbare.

Pasul de selecøie începe prin scanarea tuturor nodurilor în ordine descrescătoare a dimensiunii și prin iniøializarea a două seturi A_1 și A_2 ca seturi vide. Atunci când este scanat nodul i, *GAIN*(i) este comparat cu un întreg aleator r în intervalul [p, 0]. Dacă *GAIN*(i) > r, nodul i dorește mutarea în subsetul complementar. În acest caz, dacă $i \in V_1$, i este mutat în V_2 și este inclus în A_1. Dacă $i \in V_2$, i este mutat în V_1 și este inclus în A_2. După executarea unei mutări, câștigurile tuturor nodurilor sunt actualizate. Astfel, după scanarea tuturor nodurilor, setul A_1 conøine un subset de noduri din V_1 care au fost mutate în V_2, iar A_2 conøine un subset de noduri din V_2 care au fost mutate în V_1. Scopul acestui pas este de a minimiza dimensiunea tăieturii partiøiei.

În pasul de interschimbare, este permisă menøinerea doar a unui subset B_k de noduri din A_k în $V_{3\text{-}k}$ pentru fiecare k = 1, 2. Seturile B_1 și B_2 sunt alese astfel încât noua partiøie ($V_1 \cup B_2 - B_1$, $V_2 \cup B_1 - B_2$) să aibă un echilibru mai bun al dimensiunii faøă de vechea partiøie (V_1, V_2). În acest pas principalele operaøii sunt executate de funcøia *Echilibru*, descrisă în continuare.

Scopul funcøiei *Echilibru* este de a produce o nouă partiøie cu un echilibru mai bun al dimensiunii. Fie (V_1, V_2) o partiøie dată, astfel încât V_1 este partea cea mai mare a partiøiei. Fie $S(V_1) - S(V_2) = x > 0$. Este de dorit ca algoritmul să selecteze pentru interschimbare un subset $B_1 \subseteq A_1$ și $B_2 \subseteq A_2$ astfel încât $|S(B_1) - S(B_2) - x|$ să fie

minim. Această problemă de alegere a subseturilor B_1 şi B_2 este NP-completă [189]. Funcţia *Echilibru* rezolvă un caz special al acestei probleme ţinând cont de faptul că în pasul de selecţie un nod care este selectat pentru mutarea în subsetul complementar afectează câştigurile altor noduri. De aceea, funcţia *Echilibru* preferă pentru interschimbare nodurile selectate anterior faţă de cele selectate recent. Dacă $A_1 \cup A_2 \neq \emptyset$, funcţia garantează că $B_1 \cup B_2 \neq \emptyset$. Astfel, diferenţa între dimensiunile celor două părţi poate creşte, dar aceasta asigură algoritmului o probabilitate mai ridicată de a evita minimele locale.

Parametrii de intrare ai funcţiei *Echilibru* sunt două tablouri $A_1[\,]$ şi $A_2[\,]$ de noduri sortate în ordinea descrescătoare a dimensiunilor, şi un întreg x. Fie a_1 şi a_2 numărul elementelor din tablourile A_1, respectiv A_2. Se presupune că tabloul $A_1[\,]$ conţine noduri din subsetul mai mare. Funcţia *Echilibru* determină numerele întregi nonnegative k şi l astfel încât $|S(B_1) - S(B_2) - x|$ este minimizat, unde $B_1 = \{A_1[1], \ldots, A_1[k]\}$, iar $B_2 = \{A_2[1], \ldots, A_2[l]\}$. Cazurile de egalitate sunt rezolvate prin alegerea celui mai mare k şi l posibil. Dacă $k = 0$, atunci B_1 este vid. Similar, dacă $l = 0$, atunci B_2 este vid.

Funcţia *Echilibru* este descrisă în Figura 3.25.

```
Procedure Echilibru (a1, a2, A1, A2, x);
begin
        SA1 = 0;
        SA2 = 0;
        MIN = ∞;
        k = 0;
        l = 0;
        j = 0;
        for (j = 0 to a1) do
                SA1 = SA1 + s (A1[i]);
                min = |SA1 - SA2 - x|;
                îmbunătăţire = FALSE;
                if (a2 ≠ 0) then îmbunătăţire = TRUE; endif;
                while ((j ≤ a2) and (îmbunătăţire)) do
                        îmbunătăţire = FALSE;
                        w = |SA1 - SA2 - x - s (A2[j+1])|;
                        if (w < min) then
                                min = w;
                                j = j + 1;
                                SA2 = SA2 + s (A2[j]);
                                îmbunătăţire = TRUE;
                        endif;
                endwhile;
                if (min ≤ MIN) then
                        MIN = min;
                        k = i;
                        l = j;
                endif
        endfor;
        return (k, l);
end
```

Figura 3.25. Procedura *Echilibru* pentru algoritmul *2_SE*.

Execuţia funcţiei *Echilibru* necesită un timp liniar, deoarece i şi j nu sunt decrementate niciodată. Motivul pentru care j nu este decrementat este următorul. Presupunem că pentru un i dat, j este întregul pentru care $|SA1 - SA2 - x|$ este minim.

Atunci, pentru *i' > i*, întregul *j'* care minimizează |*SA1 - SA2 - x*| trebuie să satisfacă inegalitatea *j' > j*. De notat că *SA1* = $\sum_{l=1}^{i} s(A_1[l])$ și *SA2* = $\sum_{l=1}^{j} s(A_2[l])$.

În pasul de selecție, nodurile sunt scanate în ordinea descrescătoare a dimensiunilor, astfel că tablourile A_1 și A_2 pot fi construite într-un timp liniar fără sortare, iar ordinea elementelor lor reflectă ordinea în care nodurile au fost selectate.

Algoritmul *2_SE* poate fi modificat pentru a ține cont de ponderile diferite ale conexiunilor, sau pentru rezolvarea unor probleme similare cu bipartiționarea, cum este problema de bisecție a conexiunilor [189].

3.6.1.2 Multipartiționarea prin evoluție stohastică

Una din metodele prin care poate fi realizată multipartiționarea cu *k* căi este de a aplica în mod recursiv algoritmul de bipartiționare. Această metodă are însă un dezavantaj important. La primul nivel, algoritmul de bipartiționare încearcă să minimizeze dimensiunea tăieturii. Deoarece minimizarea dimensiunii tăieturii este echivalentă cu maximizarea numărului de conexiuni interne din ambele părți, partiționarea la primul nivel va fi în conflict cu partiționarea la al doilea nivel, unde algoritmul de bipartiționare va încerca să minimizeze dimensiunea tăieturii pentru un set de noduri puternic conectate. Această metodă recursivă de partiționare executată pe mai multe nivele poate genera o soluție globală de calitate redusă.

O altă metodă de multipartiționare este de a se începe cu anumite partiții inițiale de *k* părți. Apoi, se poate utiliza un algoritm de bipartiționare pentru a optimiza numărul de conexiuni tăiate între fiecare pereche de subseturi ale partiției cu *k* căi. Dezavantajul acestei metode este faptul că nu sunt considerate interschimbări mai complexe ale nodurilor între mai mult de două subseturi ale partiției originale cu *k* căi. Este necesară o interschimbare mai complexă a nodurilor pentru a reduce dimensiunea tăieturii.

Algoritmul de evoluție stohastică se poate utiliza și pentru rezolvarea problemei de partiționare cu *k* căi. În acest caz, diferitele subseturi ale partiției cu *k* căi vor fi considerate simultan, și nu două câte două. Saab și Rao [189] au elaborat un algoritm numit *k_SE* pentru rezolvarea următoarei probleme de partiționare cu *k* căi:

Fiind dat un set de noduri *V*, un set de conexiuni N_1, …, N_n între ele, un subset de conexiuni numit *TEST*, o dimensiune *s*(*i*) pentru fiecare $i \in V$, o pondere $W(N_j)$ pentru fiecare conexiune N_j, o limită a dimensiunii *S*, și o limită a numărului de pini *E*, să se găsească o partiție (V_1, …, V_k) care minimizează

$$\left(|\{V_i : S(V_i) < S\}|, |\{V_i : E_i < E\}|, \sum_{i=1}^{n} W(N_i) C(N_i) \right) \tag{3.41}$$

astfel încât fiecare conexiune din setul de testare *TEST* este tăiată. În formularea acestei probleme, fiind date limitele *S* și *E*, costul unei partiții (V_1, …, V_k), este vectorul:

$$COST(V_1, \ldots, V_k) = \left(|\{V_i : S(V_i) < S\}|, |\{V_i : E_i < E\}|, \sum_{i=1}^{n} W(N_i) C(N_i) \right) \tag{3.42}$$

Algoritmul *k_SE* rezolvă această problemă prin execuția următorilor pași:

1. Se determină numărul de părți în care se divide circuitul.
2. Conexiunile din setul *TEST* sunt păstrate externe prin blocarea unora din nodurile lor în diferitele părți ale partiției. Nodurile blocate nu pot fi mutate într-o altă parte.

3. Se rezolvã problema de bipartiøionare prin evoluøie stohasticã considerând nodurile care nu sunt blocate.

Numãrul iniøial de pãrøi în care se divide circuitul este determinat prin simularea unui algoritm de împachetare. Intrãrile pentru acest algoritm sunt nodurile în ordinea descrescãtoare a dimensiunii, øi o limitã a dimensiunii S. Algoritmul presupune cã existã un numãr infinit de recipiente, fiecare având o capacitate S. Pe mãsurã ce un nod este scanat, acesta este împachetat în primul recipient a cãrui dimensiune este suficientã pentru nodul respectiv.

Presupunem cã numãrul de pãrøi determinat în primul pas este k, deci setul de noduri trebuie partiøionat în k subseturi V_1, …, V_k. Fie $T = \bigcup_{N \in TEST}$ setul tuturor nodurilor care aparøin cel puøin unei conexiuni testabile. Pentru a asigura ca fiecare conexiune din *TEST* sã fie externã, trebuie blocat un subset $L \subseteq T$ de noduri din T în diferitele subseturi V_1, …, V_k astfel încât fiecare conexiune din *TEST* sã aibã o intersecøie nevidã cu cel puøin douã din subseturile V_1, …, V_k, øi $S(V_i) \leq S$ (i = 1, …, k). Fiecare nod din L, dupã ce este blocat într-unul din subseturile V_1, …, V_k, nu poate fi mutat într-un subset diferit în etapele viitoare ale algoritmului. De aceea, este de dorit ca numãrul nodurilor din setul L sã fie cât mai mic posibil.

Fie $G_i = \{N : N \in TEST,\ i \in N\}$ subsetul de conexiuni testabile care conøin nodul $i \in T$, øi fie $\rho(i) = |G_i|$ cardinalitatea lui G_i. Pentru fiecare $i \in T$ øi subset V_j fie $a_{ij} = |\{N : N \in G_i, N \cap V_j = \emptyset\}|$ numãrul de conexiuni din G_i care nu se intersectează cu V_j. Numãrul a_{ij} poate fi considerat ca atracøia pe care partea V_j o exercitã asupra nodului i. Iniøial, $a_{ij} = \rho(i)$, j = 1, …, k, dar pe mãsurã ce fiecare din nodurile lui T este blocat într-una din pãrøile V_1, …, V_k, valorile a_{ij} se modificã. Fie t_1, …, t_q setul nodurilor din T aranjate astfel încât $u < v$ implicã $\rho(t_u) \geq \rho(t_v)$. Pentru determinarea subsetului de noduri care trebuie blocate se poate utiliza urmãtoarea euristicã [189]:

1. Se setează i = 1.
2. Se determinã j care maximizează $a_{t_i j}$ astfel încât: dacã t_i este blocat în Vj atunci nici o conexiune din *TEST* nu va deveni internã, iar dimensiunea lui Vj rãmâne mai micã decât limita de dimensiune S.
3. Se eliminã t_i din T, se blochează t_i în V_j, se incrementează $S(V_j)$ cu $s(t_i)$; se actualizează valorile a_{uj} pentru fiecare $u \in T$, øi se eliminã din *TEST* toate conexiunile care devin externe ca urmare a asignării t_j la V_j.
4. Dacã $TEST \neq \emptyset$, se incrementează i øi se continuã cu pasul 2.

În acest moment, unele noduri sunt blocate în diferite pãrøi pentru a asigura cã toate conexiunile testabile sunt externe, în timp ce nodurile rãmase neblocate nu sunt asignate nici unui subset al partiøiei. Se utilizează în continuare algoritmul *SE* pentru a termina partiøionarea circuitului. Strategia generalã a acestui algoritm fiind cunoscutã, *k_SE* poate fi complet specificat prin indicarea partiøiei iniøiale øi a funcøiei *Perturb*.

Partiøia iniøialã este generatã utilizând aceeaøi euristicã ca øi pentru determinarea numãrului de pãrøi. În aceastã etapã, unele subseturi ale partiøiei au capacitãøi mai mici decât S datoritã faptului cã anumite noduri sunt deja asignate la aceste subseturi. De aceea este posibil ca algoritmul sã mãreascã numãrul de pãrøi pentru a permite plasarea tuturor nodurilor. Se presupune însã cã numãrul conexiunilor testabile este redus, astfel încât cele k subseturi sunt suficiente. În euristica utilizatã se presupune cã sunt disponibile numai k pãrøi, øi fiecare nod neblocat va gãsi un spaøiu într-una din cele k pãrøi.

Ca øi în cazul algoritmului *2_SE*, funcøia *Perturb* constã din doi paøi: selecøie øi interschimbare. Pentru fiecare nod neblocat i øi fiecare parte V_j se defineøte forøa care atrage nodul i în partea V_j ca fiind:

$$F_{ij} = \begin{cases} |N_l : i \in N_l, |N_l \cap V_j| > 0\}| & \text{dacă } i \notin V_j \\ |N_l : i \in N_l, |N_l \cap V_j| > 1\}| & \text{dacă } i \in V_j \end{cases} \quad (3.43)$$

Pasul de selecøie începe prin iniøializarea a *k* cozi Q_1, …, Q_k. Nodurile neblocate sunt scanate apoi în ordinea descrescãtoare a dimensiunii. Presupunând cã nodul curent scanat este *i*, fie

1. V_j partea curentã a nodului *i*.
2. V_l o parte astfel încât $l \neq j$ şi F_{il} este maxim.
3. e_i = valoarea cu care scade E_j dacã nodul *i* este mutat în afara pãrøii V_j.
4. g_i = valoarea cu care scade $\sum_{i=1}^{n} W(N_i)C(N_i)$ atunci când nodul *i* este mutat în partea V_l.
5. $GAIN(i) = e_i$ dacã $E_j > E$, $GAIN(i) = g_i$ în caz contrar.

$GAIN(i)$ este comparat cu un întreg aleator *r* în intervalul [*p*, 0]. Dacã $GAIN(i) > r$, nodul *i* dorešte mutarea în V_l. În caz contrar, nodul *i* rãmâne în V_j. Dacã nodul *i* se mutã în V_l, atunci este eliminat din V_j, este adãugat în coada Q_l, fiind actualizate valorile F_{uj}, e_v şi g_v ca şi cum nodul *i* ar fi deja în V_l.

În pasul de interschimbare, fiecare parte acceptã numãrul de noduri permis de restricøia sa de dimensiune, pe baza regulii primul venit, primul servit. Nodurile rãmase cãrora nu li se permite intrarea în partea lor preferatã sunt depuse într-o coadã *L*. Pãrøile sunt scanate apoi în ordinea crescãtoare a dimensiunii. Atunci când este scanatã partea V_l, aceasta permite intrarea unui numãr de noduri din *L* în funcøie de restricøia sa de dimensiune. Dacã, dupã scanarea tuturor pãrøilor, coada *L* nu este vidã, nodurile rãmase în *L* sunt returnate în pãrøile lor iniøiale.

Calitatea soluøiei gãsite de algoritmul *k_SE* depinde de alegerea parametrilor *S*, *E* şi *R*. Dacã oricare din acešti parametri are o valoare prea micã, posibilitãøile de optimizare ale algoritmului sunt mult restrânse. Dacã *S* este prea mare, existã tendinøa ca dimensiunile pãrøilor sã fie dezechilibrate. Dacã *R* este prea mare, timpul de calcul creşte.

3.6.2 Partiționarea prin automate de învățare

Automatele de învãøare au fost utilizate pentru modelarea sistemelor biologice de învãøare şi de asemenea pentru procesul de învãøare a acøiunii optime pe care o oferã un mediu aleator. Învãøarea este realizatã prin interacøiunea cu mediul şi prelucrarea rãspunsurilor sale la acøiunile care au fost alese.

Procesul de învãøare al unui automat poate fi descris astfel: Automatului i se oferã un set de acøiuni de cãtre mediul cu care acesta interacøionează, şi este constrâns sã aleagã una din aceste acøiuni. Atunci când este aleasã o acøiune, automatul este fie recompensat, fie penalizat de cãtre mediu, cu o anumitã probabilitate. Automatul va învãøa alegerea acøiunii optime, care este acøiunea cu probabilitatea minimã de penalizare. În final, automatul va alege aceastã acøiune mai frecvent decât alte acøiuni.

În secøiunea 3.6.2.1 sunt prezentate pe scurt automatele stohastice de învãøare, şi este descrisã o categorie particularã de automate, automatul pentru migrarea obiectelor. În secøiunea 3.6.2.2 este descris un automat de învãøare pentru partiøionarea grafurilor.

3.6.2.1 Automate de învățare şi partiționarea obiectelor

Automatele stohastice de învãøare pot fi clasificate în douã categorii principale:

1) Automate stohastice cu structurã fixã
2) Automate a cãror structurã evoluează în timp

Exemple din primul tip sunt automatele Tsetlin, Krinsky şi Krylov [126]. Deşi automatele din al doilea tip sunt numite cu structurã variabilã, deoarece matricile lor de tranziøie şi de ieşire sunt variabile în timp, în practicã ele sunt definite în termenii regulilor de actualizare a probabilitãøilor de acøiune.

Un automat stohastic cu structurã fixã (*ASSF*) este un cvintuplu (α, Φ, β, F, G), unde:

1) $\alpha = \{\alpha_1, \ldots, \alpha_R\}$ este setul acøiunilor din care automatul trebuie sã aleagã.
2) $\Phi = \{\phi_1, \ldots, \phi_S\}$ este setul stãrilor automatului.
3) $\beta = \{0, 1\}$ este setul intrãrilor, unde "1" reprezintã o penalizare şi "0" o recompensã.
4) F: $\Phi \times \beta \rightarrow \Phi$ este maparea de tranziøie, care defineşte tranziøia stãrii automatului la recepøionarea unei intrãri. F poate fi stohasticã.
5) G: $\Phi \rightarrow \alpha$ este maparea de ieşire, care determinã acøiunea executatã de automat dacã acesta este în starea ϕ_i.

Acøiunea selectatã serveşte ca intrare pentru mediu, care la rândul sãu transmite un rãspuns stohastic $\beta(n)$ la momentul n. $\beta(n)$ este un element din $\beta = \{0, 1\}$ şi este rãspunsul de reacøie al mediului pentru automat. Mediul penalizează ($\beta(n) = 1$) automatul cu penalizarea c_i, care este dependentã de acøiune. Pe baza rãspunsului $\beta(n)$, starea $\phi(n)$ a automatului este actualizatã şi este aleasã o nouã acøiune la momentul n+1.

Problema de partiøionare a grafurilor poate fi rezolvatã în mai multe moduri, considerând aceastã problemã ca una de cãutare sau de exersare bazatã pe parametri. Oommen şi St. Croix [129] au propus o metodã în care problema de partiøionare a grafurilor este rezolvatã prin considerarea acesteia ca o problemã de partiøionare a obiectelor. În acest caz, scopul este de a modela problema astfel încât sã se determine nodurile din V care se potrivesc cu alte noduri cu proprietãøi similare. Se încearcã deci impunerea unei mãsuri inteligente de similaritate pentru noduri, astfel încât nodurile similare sã se grupeze. Aceastã mãsurã de similaritate este legatã de apartenenøa sau nu a nodurilor la partiøia optimã.

Existã mai multe avantaje ale acestei abordãri. Spre deosebire de metodele de cãutare, cãutarea unei soluøii mai bune nu se realizează doar printr-o nouã partiøionare în "vecinãtatea" celei curente. Se realizează în schimb compararea unor perechi de noduri, şi uneori, a unui grup de trei sau patru noduri, pentru a se asigura învãøarea. De asemenea, aceastã tehnicã este adaptivã. În plus, aceastã tehnicã asigurã un mecanism prin care se poate indica nodul cel mai reprezentativ pentru fiecare subpartiøie. Acest nod, care poate fi considerat ca nucleul subpartiøiei respective, este specificat în mod automat şi adaptiv de sistem. Automatul propus în [129] poate fi adaptat şi pentru problema de partiøionare a grafurilor în care costurile muchiilor nu sunt constante, ci sunt variabile aleatoare a cãror distribuøie este necunoscutã.

Problema de *partiøionare a obiectelor* poate fi definitã astfel: Fie $\boldsymbol{A} = \{A_1, \ldots, A_W\}$ un set de W obiecte care trebuie partiøionate în K clase $\{\pi_1, \ldots, \pi_K\}$. Obiectele sunt partiøionate astfel încât cele care sunt accesate împreunã în mod frecvent sã fie plasate în aceeaşi clasã. Aceasta implicã existenøa unei partiøionãri corecte, numitã grupare, a obiectelor. În modul cel mai simplu de partiøionare a obiectelor, acestea sunt accesate în perechi $<A_i, A_j>$, printr-o interogare. De aceea, soluøia acestei probleme va prelucra o secvenøã de interogãri astfel încât sã creascã probabilitatea ca perechea de obiecte accesate sã facã parte din aceeaşi clasã la o interogare ulterioarã identicã.

Problema de partiøionare a grafurilor este strâns legatã de un caz special al partiøionãrii obiectelor, cel în care toate clasele au aceeaši dimensiune. Aceastã problemã este numitã *problema de echipartiøionare*. O soluøie a acestei probleme este metoda propusã de Yu. În aceastã metodã, iniøial fiecare obiect membru, A_i, este asociat cu un numãr real X_i. La fiecare procesare a unei interogãri $<A_i, A_j>$, cantitãøile $\{X_i, X_j\}$ sunt mutate cãtre nucleul lor cu o anumitã cantitate δ_1. Apoi, o pereche aleatoare $<X_p, X_q>$, $(1 \le p, q \le W)$ este îndepãrtatã de nucleul acesteia cu o anumitã cantitate δ_2.. Rezultatul este cã obiectele similare se vor grupa împreunã, iar cele nesimilare se vor separa.

O altã soluøie a acestei probleme o reprezintã *automatul pentru migrarea obiectelor* (*AMO*), propus de Oommen şi Ma [130]. Acest automat modificã stãrile tuturor celor *W* obiecte, spre deosebire de automatul tradiøional la care obiectele sunt trecute dintr-o stare în alta. Deci, atunci când acest automat este utilizat pentru rezolvarea problemei de echipartiøionare, o soluøie nu este definitã prin starea curentã a *AMO*, ci prin întreaga structurã a automatului. Funcøia de mapare care defineşte tranziøia automatului dintr-o stare în alta specificã mutarea a douã sau trei obiecte în cadrul acestei structuri. La o interogare $<A_i, A_j>$, dacã A_i şi A_j fac parte din aceeaşi clasã, ambele sunt recompensate şi sunt mutate cu un pas cãtre starea cea mai interioarã asociatã cu acea clasã. Dacã A_i şi A_j fac parte din clase diferite, ele sunt mutate cu un pas cãtre stãrile care sunt învecinate cu stãrile din alte clase, şi atunci când unul din obiecte se aflã într-una din aceste stãri limitã, va migra spre starea corespunzãtoare la o altã acøiune. Rezultatele obøinute cu *AMO* aratã cã acesta converge cãtre soluøia corectã în toate cazurile, într-un timp care este cu un ordin de mãrime mai redus decât cel al metodei propuse de Yu.

3.6.2.2 Automat de învățare pentru partiționarea grafurilor

Automatul de învãøare pentru partiøionarea grafurilor (*APG*) are ca intrãri setul de noduri $V = \{v_1, \ldots, v_{2N}\}$ care trebuie partiøionat în douã subpartiøii de dimensiuni egale, V_1 şi V_2, şi setul de ponderi ale muchiilor. *APG* poate fi generalizat pentru cazul partiøionãrii setului de noduri *V* în *k* subpartiøii prin proiectarea acestuia astfel încât sã dispunã de *k* acøiuni. În continuare se presupune cã automatul trebuie sã rezolve aceastã ultimã problemã de echipartiøionare a setului *V* în *k* subseturi.

Iniøial, toate obiectele (fiecare reprezentând un nod) sunt plasate aleator în cele *k* acøiuni. *APG* se defineşte ca un 7-tuplu astfel:

$(V, E, \{\phi_1, \phi_2, \ldots, \phi_{kM}\}, \{\alpha_1, \alpha_2 \ldots, \alpha_k\}, \beta, Q, G)$, unde:

1) $V = \{v_1, \ldots, v_{kN}\}$ este setul de noduri care trebuie partiøionat.
2) *E* este setul de muchii, cãruia i se asociazã matricea costurilor *C*.
3) $\{\phi_1, \phi_2, \ldots, \phi_{kM}\}$ este setul stãrilor. *M* se numeşte adâncimea memoriei automatului.
4) $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ este setul celor *k* acøiuni, fiecare reprezentând o anumitã subpartiøie în care trebuie plasate elementele lui *V*.
5) $\beta = \{0, 1\}$ este setul intrãrilor, unde "1" reprezintã o penalizare şi "0" o recompensã.
6) *Q* este funcøia de tranziøie, care specificã modul în care trebuie mutate obiectele între diferitele stãri.
7) Funcøia *G* partiøioneazã setul stãrilor. Pentru fiecare acøiune α_j existã un set de stãri $\{\phi_{(j-1)M+1}, \ldots, \phi_{jM}\}$. Deci,

$$G(\phi_i) = \alpha_j \text{ dacã } (j-1)M + 1 \le i \le jM \qquad (3.44)$$

Aceasta înseamnã cã nodul automatului alege α_1 dacã se aflã în oricare din primele M stãri, alege α_2 dacã se aflã în oricare din stãrile de la ϕ_{M+1} la ϕ_{2M} etc. Se presupune cã $\phi_{(j-1)M+1}$ este starea cea mai interioarã a acøiunii α_j, iar ϕ_{jM} este starea de graniøã. Acestea reprezintã starea de *Certitudine_Maximã*, respectiv de *Certitudine_Minimã*.

Dacã stãrile ocupate de noduri sunt date, subpartiøiile se pot obøine din relaøia (3.44). Aceasta va specifica complet setul subpartiøiilor dictate în mod curent de *APG*. Fie $\omega_i(n)$ indexul stãrii ocupate de nodul v_i la momentul de timp n. Pe baza acestui index øi a relaøiei (3.44) *APG* va decide o partiøionare curentã a setului V în subpartiøii.

În timpul procesului de învãøare scopul este de a se colecta noduri similare ale grafului în aceeaøi subpartiøie. Principiul utilizat pentru cuantificarea acestei similaritãøi este analog cu cel utilizat de Kernighan øi Lin. Se considerã cã douã noduri sunt similare dacã ele sunt puternic conectate, øi deci costul muchiei acestora este redus. Explicit, se considerã cã nodurile sunt puternic conectate dacã muchia corespunzãtoare lor are un cost care este de $(1 + \rho)$ ori costul mediu al muchiilor pentru întregul graf, unde ρ este un parametru specificat de utilizator. Similar, nodurile sunt slab conectate dacã muchia lor are un cost care este de $(1 - \rho)$ ori costul mediu al muchiilor [129].

Diferitele stãri din cadrul unei subpartiøii date cuantificã mãsura de certitudine pe care o are sistemul pentru un nod dat aparøinând subpartiøiei respective. La început toate nodurile sunt plasate în subpartiøii alese în mod aleator, în starea limitã (de *Certitudine_Minimã*) a acestor subpartiøii, indicând faptul cã sistemul este nesigur de plasarea corectã a nodurilor. Pe mãsura procesului de învãøare, nodurile similare ale grafului vor fi recompensate pentru apartenenøa lor la aceeaøi subpartiøie, øi vor migra cãtre starea cea mai interioarã a subpartiøiei (de *Certitudine_Maximã*), indicând faptul cã sistemul este mai sigur de plasarea nodurilor în subpartiøiile corecte. În acelaøi timp, alte noduri vor fi penalizate øi vor fi mutate fie spre starea lor limitã, fie în altã subpartiøie, indicând ambiguitatea sistemului în privinøa asocierii lor cu subpartiøia curentã.

Iniøial, automatul are o fazã de preprocesare în care se evalueazã costul mediu al muchiilor. Urmeazã apoi bucla principalã de învãøare a algoritmului. Muchiile sunt procesate în mod repetat, într-o ordine aleatoare. La procesarea unei muchii e_{ij}, dacã v_i øi v_j sunt similare øi aparøin aceleiaøi subpartiøii, nodurile v_i øi v_j sunt recompensate. Acest mod de recompensare se numeøte *Recompensare_Noduri_Similare*. Dacã ele sunt asignate însã unor subpartiøii diferite, automatul este penalizat. Acest mod de penalizare se numeøte *Penalizare_Noduri_Similare*.

Dacã v_i øi v_j sunt nesimilare øi aparøin unor subpartiøii diferite, automatul (øi, în particular, nodurile v_i øi v_j) sunt recompensate. Acest mod de recompensare se numeøte *Recompensare_Noduri_Nesimilare*. Dacã însã nodurile sunt asignate aceleiaøi subpartiøii, automatul este penalizat. Prin analogie cu cele de sus, acest mod de penalizare se numeøte *Penalizare_Noduri_Nesimilare*.

Ciclul continuã apoi cu urmãtoarea iteraøie, unde fazele de recompensare øi penalizare se repetã.

Se prezintã în continuare tranziøiile descrise de funcøia Q pentru fiecare din aceste operaøii.

Modul *Recompensare_Noduri_Similare*. Acesta este cazul întâlnit atunci când nodurile v_i øi v_j sunt considerate similare øi se aflã simultan în aceeaøi subpartiøie, de exemplu α_k. Din acest motiv, ambele noduri sunt mutate cãtre starea cea mai interioarã a acelei subpartiøii, $\phi_{(k-1)M+1}$, cu câte un pas în fiecare moment de timp (Figura 3.26). Dacã oricare din noduri se aflã în starea cea mai interioarã, va rãmâne în aceastã stare.

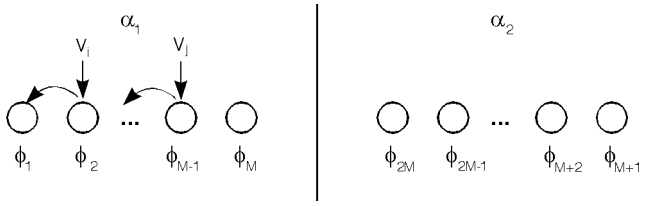

Figura 3.26. Tranziøii de recompensare pentru *APG* cu 2M stări. În acest caz (modul *Recompensare_Noduri_Similare*), v_i şi v_j sunt similare şi se aflã în aceeaşi subpartiøie.

Modul *Recompensare_Noduri_Nesimilare*. În acest caz nodurile v_i şi v_j sunt considerate nesimilare şi se aflã în subpartiøii diferite, de exemplu α_k, respectiv α_m. Ambele noduri sunt mutate cãtre starea cea mai interioarã a subpartiøiei respective, $\phi_{(k\text{-}1)M+1}$ şi $\phi_{(m\text{-}1)M+1}$, cu câte un pas în fiecare moment de timp.

Modul *Penalizare_Noduri_Similare*. Acesta este cazul întâlnit atunci când douã noduri similare, v_i şi v_j, sunt plasate în subpartiøii diferite, de exemplu α_k, respectiv α_m. Nodul v_i se aflã în starea ω_i, unde $\omega_i \in \{\phi_{(k\text{-}1)M+1}, \ldots, \phi_{kM}\}$, iar nodul v_j se aflã în starea ω_j, unde $\omega_j \in \{\phi_{(m\text{-}1)M+1}, \ldots, \phi_{mM}\}$. Din acest motiv, nodurile sunt mutate din stările ω_i şi ω_j,, dupã cum urmeazã:

1) Dacã $\omega_i \neq \phi_{kM}$ şi $\omega_j \neq \phi_{mM}$, se mutã v_i şi v_j cu o stare cãtre ϕ_{kM}, respectiv ϕ_{mM}. Astfel, nodurile se mutã cãtre stările limitã ale subpartiøiilor respective (Figura 3.27).

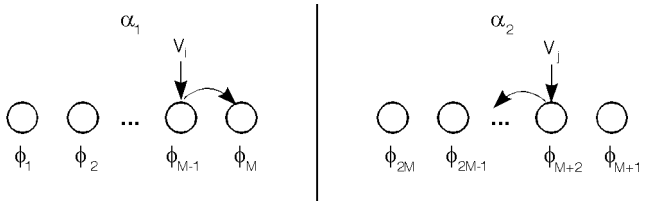

Figura 3.27. Tranziøii de penalizare pentru *APG* cu 2M stări – modul *Penalizare_Noduri_Similare*: v_i şi v_j sunt similare, dar se aflã în subpartiøii distincte.

2) Dacã cel puøin unul din nodurile v_i şi v_j se aflã în starea limitã (deci, $\omega_i = \phi_{kM}$ sau $\omega_j = \phi_{mM}$), atunci se mutã nodul din starea limitã a subpartiøiei sale în starea limitã a subpartiøiei celuilalt nod. De exemplu, dacã nodul v_j se aflã în starea limitã, v_j se mutã în starea ϕ_{kM}, care este starea limitã a subpartiøiei α_k. În acest caz, deoarece va rezulta un exces de noduri în partiøia α_k, unul din nodurile diferite de v_i din α_k este mutat în starea ϕ_{mM}, care este starea limitã din α_j. Se alege mutarea nodului cel mai apropiat de ϕ_{kM} (Figura 3.28).

Modul *Penalizare_Noduri_Nesimilare*. Acesta este cazul întâlnit atunci când douã noduri nesimilare, v_i şi v_j, sunt plasate în aceeaşi subpartiøie, de exemplu α_k. Din acest motiv, ambele noduri sunt mutate din starea ω_i, dupã cum urmeazã:

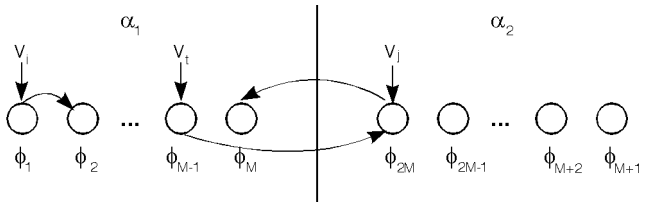

Figura 3.28. Tranziøii de penalizare pentru *APG* cu 2M stãri – modul *Penalizare_Noduri_Similare*. În acest caz v_i øi v_j sunt similare, dar se aflã în subpartiøii distincte. Unul din noduri (v_j) se aflã într-o stare limitã.

1) Dacã $\omega_i \neq \phi_{kM}$ øi $\omega_j \neq \phi_{mM}$, se mutã v_i øi v_j cu o stare cãtre starea limitã ϕ_{kM}, (Figura 3.29).

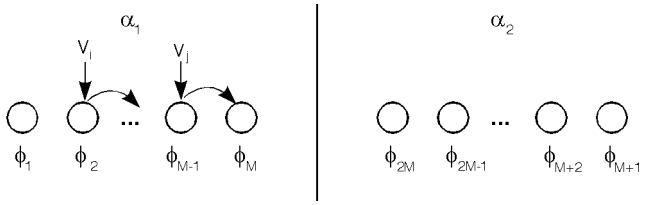

Figura 3.29. Tranziøii de penalizare pentru *APG* cu 2M stãri – modul *Penalizare_Noduri_Nesimilare*. În acest caz v_i øi v_j sunt nesimilare, dar se aflã în aceeaøi subpartiøie. Niciunul din noduri nu se aflã într-o stare limitã.

2) Dacã cel puøin unul din nodurile v_i øi v_j se aflã în starea limitã (deci, $\omega_i = \phi_{kM}$ sau $\omega_j = \phi_{kM}$), atunci se mutã nodul aflat în starea limitã, de exemplu v_j, în starea ϕ_{ZM}, care este starea limitã a subpartiøiei α_Z. În cazul bipartiøionãrii, aceasta este întotdeauna subpartiøia diferitã de cea în care se aflã nodul. În cazul multipartiøionãrii, modul migrat este mutat în mod repetat în toate cele k-1 subpartiøii, øi în final este reøinut în cea mai bunã dintre ele. Aceasta este cea care minimizeazã costurile dintre seturi. Øi în acest caz, deoarece va rezulta un exces de noduri în partiøia α_Z, unul din nodurile din α_Z, cel mai apropiat de ϕ_{ZM}, este mutat în starea ϕ_{mM} (Figura 3.30).

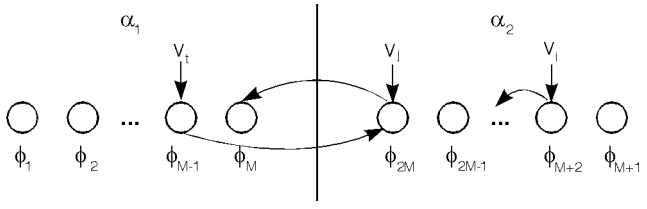

Figura 3.30. Tranziøii de penalizare pentru *APG* cu 2M stãri – modul *Penalizare_Noduri_Nesimilare*. În acest caz v_i øi v_j sunt nesimilare, dar se aflã în aceeaøi subpartiøie. Unul din noduri (v_j) se aflã într-o stare limitã.

Algoritmul automatului pentru multipartiøionarea grafurilor este prezentat în Figura 3.31.

Deşi principiile fundamentale ale migrării nodurilor sunt bazate pe filozofia utilizată de automatul pentru migrarea obiectelor *AMO*, algoritmul este diferit [129]. Spre deosebire de *AMO*, unde migraøiile se efectuează atunci când utilizatorul execută o cerere, în cazul *APG* migraøiile se execută pentru toate perechile de noduri procesate. De asemenea, *APG* are o strategie de penalizare a elementelor neaccesate prin considerarea gradului de similaritate a nodurilor din aceeaşi subpartiøie. *APG* permite cuantificarea gradului de potrivire a unui nod pentru o anumită subpartiøie. Nodul cel mai apropiat de starea cea mai interioară poate fi considerat ca fiind nodul care reprezintă cel mai bine acea subpartiøie.

Deşi *APG* obøine soluøii într-un timp redus, acestea nu reprezintă de obicei optimul global. Cu alte cuvinte, *APG* determină într-un timp scurt regiunea apropiată de o soluøie optimă. Aceasta din cauza nodurilor care au tendinøa să migreze între diferitele subpartiøii, şi deci au tendinøa de a rămâne în stările limită ale acestor subpartiøii. De aceea, se obøine o îmbunătăøire dacă algoritmul este urmat de o singură iteraøie a unui algoritm de căutare locală de tip Kernighan-Lin.

```
Algorithm APG;
        /* Intrarea este setul V = {v1, …, vkN } care trebuie partiţionat în k subpartiţii.   */
        /* Ieşirile sunt subpartiţiile {V1, V2, …, Vk }.                                      */
        /* C este matricea de adiacenţă a costurilor.                                         */
        /* ρ este un parametru utilizat pentru a decide similaritatea nodurilor.              */
begin
        Se calculează costul mediu pe muchie Cost_Mediu;
        Se echipartiţionează V în mod aleator în {V1, V2, …, Vk };
        Se asignează nodurilor stările limită ale sub-partiţiilor;
        for (Iter = 1 to Max_Iter) do
                for (o muchie aleatoare eij) do
                        if (Cij > (1+ρ) · Cost_Mediu) then
                                if (vi şi vj sunt în aceeaşi clasă) then
                                        Recompensare_Noduri_Similare (i, j);
                                else
                                        Penalizare_Noduri_Similare (i, j);
                                endif
                        else
                                if (Cij < (1-ρ) · Cost_Mediu) then
                                        if (vi şi vj sunt în aceeaşi clasă) then
                                                Penalizare_Noduri_Nesimilare (i, j);
                                        else
                                                Recompensare_Noduri_Nesimilare (i, j);
                                        endif
                                endif
                        endif
                endfor
        endfor
end.
```

Figura 3.31. Algoritmul automatului de învăøare pentru partiøionarea grafurilor.

3.7 Algoritmi de partiționare propuşi pentru circuite FPGA cu resurse limitate de rutare

Circuitele FPGA utilizează comutatoare programabile care permit configurarea acestora de către utilizator. Acest avantaj se obøine cu costul unui grad de utilizare mai redus al porøilor disponibile. Unul din motivele reducerii gradului de utilizare al porøilor este disponibilitatea limitatã a resurselor de rutare în circuitele FPGA comerciale: existenøa unui numãr fix de piste într-un canal de rutare şi limitări ale interconexiunilor disponibile între piste. În special în cazul circuitelor FPGA cu resurse limitate de rutare, cum sunt şi circuitele FPGA *Atmel*, este foarte importantã utilizarea unor tehnici care să asigure rutabilitatea circuitelor.

Pentru asigurarea rutabilitãøii circuitelor, trebuie realizatã o plasare corespunzãtoare a celulelor. În cazul utilizării unor algoritmi de plasare bazaøi pe partiøionare, algoritmul de partiøionare trebuie să aibã în vedere acest obiectiv al rutabilitãøii. Cei mai utilizaøi algoritmi de partiøionare se bazează pe metoda tãieturii minime, cu scopul de a minimiza numãrul interconexiunilor intersectate de o serie de linii de tãieturã. Plasarea care utilizează această partiøionare are rolul de a grupa cât mai mult celulele interconectate, lungimea totalã a interconexiunilor fiind componenta principalã a funcøiei de cost utilizate de algoritmii tradiøionali de plasare.

Lungimea totalã a interconexiunilor nu este însã o metricã adecvatã pentru circuitele FPGA cu resurse limitate de rutare. Deoarece algoritmul va încerca plasarea apropiatã a celulelor conectate, este posibil ca plasarea rezultatã să fie dificil sau imposibil de rutat. În această secøiune se vor prezenta algoritmi de partiøionare care au ca obiectiv echilibrarea numãrului de conexiuni din cadrul partiøiilor, minimizând în acelaşi timp lungimea totalã a interconexiunilor. În secøiunea 3.7.1 se va prezenta un algoritm de partiøionare bazat pe metoda tãieturii minime. În secøiunea 3.7.2 se va prezenta un algoritm genetic pentru partiøionare, cu un obiectiv similar cu cel al algoritmului bazat pe tãietura minimã.

3.7.1 Algoritm de partiționare cu echilibrarea numărului de conexiuni

Pentru a elabora o procedurã eficientã de partiøionare, este necesarã o metricã pentru a mãsura congestia canalelor de rutare. Un indicator al acestei congestii este dimensiunea tãieturii, deci numãrul de conexiuni intersectate de o linie de tãieturã din circuit. Într-un circuit FPGA cu $a \times b$ celule logice, există $a - 1$ linii de tãieturã orizontale şi $b - 1$ linii de tãieturã verticale. Fiecãrui canal de rutare îi corespunde o linie de tãieturã. Dimensiunea tãieturii unei linii de tãieturã este definitã prin numãrul de conexiuni care au terminale de ambele pãrøi ale liniei de tãieturã. Această dimensiune reprezintã o limitã inferioarã a numãrului de piste necesare pentru rutarea completã a circuitului. Deci, dacã dimensiunea tãieturii unei linii de tãieturã orizontale (verticale) este mai mare decât numãrul pistelor verticale (orizontale), rutarea completã nu este posibilã.

O linie de tãieturã care trece printr-o zonã cu un numãr mare de conexiuni are de obicei o dimensiune mare a tãieturii. Dacã dimensiunea maximã a tãieturii, deci dimensiunea care este cea mai mare dintre toate liniile de tãieturã orizontale şi verticale, scade, posibilitatea existenøei unor zone congestionate din punct de vedere al conexiunilor scade în mod corespunzãtor. De aceea, este de dorit minimizarea dimensiunii maxime a tãieturii. Suma dimensiunii tãieturilor pentru toate liniile de tãieturã orizontale şi verticale indicã lungimea totalã a conexiunilor atunci când această lungime pentru o conexiune cu terminale multiple este estimatã prin metoda semiperimetrului.

Algoritmii de plasare bazaøi pe partiøionarea de tip Kernighan-Lin executã în mod recursiv bipartiøionarea grafului (sau hiper-grafului) care reprezintã circuitul, pânã când graful este suficient de simplu (de exemplu, un graf cu un singur nod) pentru a fi plasat într-o celulã a circuitului. Obiectivul fiecãrei bipartiøionãri este de a minimiza dimensiunea tãieturii cu restricøia ca dimensiunile (numãrul de noduri) celor douã partiøii obøinute sã fie aproximativ egale. În cazul acestor algoritmi, singura metricã în cadrul funcøiei de cost minimizate este dimensiunea tãieturii. De aceea, se poate obøine o partiøie cu o dimensiune redusã a tãieturii, în care una din porøiuni conøine un numãr mare de conexiuni, în timp ce numãrul conexiunilor din a doua porøiune este redus. Se propune un algoritm de bipartiøionare care øine cont nu numai de dimensiunile celor douã porøiuni, ci øi de distribuøia interconexiunilor din cadrul celor douã porøiuni.

Conexiunile cu terminale multiple se considerã reprezentate printr-un hipergraf. În general, pentru a conecta k terminale, sunt necesare $\max\{k-1, 0\}$ cãi de conectare. Se defineøte *dezechilibrul* unei conexiuni ca fiind diferenøa dintre numãrul cãilor de interconectare necesare pentru conectarea terminalelor din porøiunea din stânga øi numãrul cãilor de interconectare necesare pentru conectarea terminalelor din porøiunea din dreapta. Dezechilibrul unei bipartiøii este definit ca suma valorilor de dezechilibru pentru toate conexiunile. Valoarea absolutã a dezechilibrului unei bipartiøii indicã diferenøa dintre numãrul cãilor de conectare necesare în porøiunea din stânga øi acelaøi numãr din porøiunea din dreapta.

Fiind datã o bipartiøie iniøialã, se poate calcula dezechilibrul acesteia într-un timp de $O(|T|)$ prin examinarea tuturor conexiunilor, unde T este setul tuturor terminalelor. Fãrã restrângerea generalitãøii, presupunem cã existã un numãr mai mare de cãi de interconectare în porøiunea din stânga decât în cea din dreapta, deci dezechilibrul unei bipartiøii este pozitiv. Dacã un nod v este mutat din porøiune din stânga în cea din dreapta, se poate calcula funcøia $f(v, e)$, reprezentând cantitatea cu care scade dezechilibrul unei conexiuni e, prin examinarea setului tuturor vecinilor lui v, $N(v)$. Câøtigul care indicã reducerea dezechilibrului este definit prin

$$F(v) = \sum_{\forall v \in e} f(v, e) \tag{3.45}$$

Dacã nodul v este mutat dintr-o porøiune în cealaltã, este necesar un timp de $O(|N(v)|)$ pentru actualizarea valorii funcøiilor de câøtig. Atunci când un nod v dintr-o porøiune este interschimbat cu un alt nod u din cealaltã porøiune, în cazul în care nu existã o muchie care conectează cele douã noduri, reducerea dezechilibrului bipartiøiei este suma dintre $F(v)$ øi $F(u)$. În cazul în care existã muchii între cele douã noduri, pentru fiecare asemenea muchie, e, se scade valoare $f(u, e) + f(v, e)$ din suma $F(v) + F(u)$.

Se utilizează urmãtoarea funcøie de cost pentru a øine cont de distribuøia numãrului de conexiuni în cadrul unei partiøii:

$$Cost = Dim\_tăietură + PONDERE \times Dezechilibru \tag{3.46}$$

unde *PONDERE* este o constantã. Dacã aceastã constantã este setatã la zero, algoritmul este acelaøi cu cel convenøional. Prin setarea corespunzãtoare a ponderii se poate controla importanøa echilibrãrii numãrului de conexiuni.

Se considerã un graf $G = (V, E)$ cu $2n$ vârfuri. Algoritmul de bipartiøionare cu echilibrarea numãrului de conexiuni este prezentat în Figura 3.32 [14].

Tabloul *Blocat* memorează valoarea 0 dacã un nod este disponibil pentru interschimbare øi 1 dacã nodul este blocat. Procedura *PART_INIT* (G, n) selectează un subgraf de dimensiune n. E_i øi I_i reprezintã costul extern, respectiv costul intern al nodului i (3.4.1). Procedura *INTERSCH* (V_1, V_2, v_i, v_j) interschimbã nodurile $v_i \in V_1$ øi $v_j \in V_2$ din douã subgrafuri G_1, respectiv G_2. Tabloul *gain* memorează câøtigul pentru fiecare pereche de noduri, iar tablourile *max1* øi *max2* pãstrează indicii acestor

noduri. Tabloul *GAIN* memorează câştigul acumulat pentru o secvenţă de *n* interschimbări. Variabilele *bestk* şi *bestGAIN* conţin indexul, respectiv valoarea câştigului maxim acumulat.

```
Algorithm Partit_echil (G, n, PONDERE);
begin
    G1 = PART_INIT (G, n);
    G2 = G - G1;
    bestGAIN = ∞;
    while bestGAIN > 0 do
        nr_dezechilibru = DEZECHILIBRU (n);
        bestGAIN = 0; bestk = 0;
        for i = 1 to 2n do Blocat (i) = 0; endfor;
        for k = 1 to n do
            gain (k) = 0; GAIN (k) = 0;
            /* Se caută nodurile care vor fi interschimbate */
            for all v_i ∈ G1 AND Blocat (i) = 0 do
                for all v_j ∈ G2 AND Blocat (j) = 0 do
                    gain_cut = D_i + D_j - 2 c_ij;
                    nr_dezechilibru_nou = nr_dezechilibru - E_i - I_i + E_j + I_j;
                    gain_dez = | nr_dezechilibru - nr_dezechilibru_nou |;
                    if (gain_cut + PONDERE * gain_dez) > gain (k) then
                        gain (k) = gain_cut + PONDERE * gain_dez;
                        max1 (k) = i; max2 (k) = j;
                    endif;
                endfor;
            endfor;
            Blocat (max1 (k)) = Blocat (max2 (k)) = 1;
            /* Actualizează câştigul după tentativa de interschimbare */
            for i = 1 to 2n do
                if v_i ∈ G1 AND Blocat (i) = 0 then
                    D_i = D_i - c_i,max2(k) + c_i,max1(k)
                endif;
                if v_i ∈ G2 AND Blocat (i) = 0 then
                    D_i = D_i - c_i,max1(k) + c_i,max2(k);
                endif;
            endfor;
            /* Calculează câştigul acumulat */
            GAIN (k) = GAIN (k-1 ) + gain (k);
            if GAIN (k) > bestGAIN then
                bestk = k; bestGAIN = GAIN (k);
            endif;
        endfor;
        /* Interschimbă k perechi de noduri */
        for k = 1 to bestk do
            (G1, G2) = INTERSCH (V1, V2, v_max1(k), v_max2(k));
        endfor;
    endwhile;
end.
```

Figura 3.32. Algoritmul de bipartiţionare cu echilibrarea numărului de conexiuni.

3.7.2 Algoritm genetic pentru partiționare cu echilibrarea numărului de conexiuni

Algoritmii genetici reprezintă o tehnică de căutare în cadrul unui spaøiu al soluøiilor, care emulează procesul natural al evoluøiei ca o modalitate de evoluare către o soluøie optimă [146]. Acești algoritmi au fost aplicaøi pentru rezolvarea diferitelor probleme de optimizare, inclusiv pentru bipartiøionarea sau multipartiøionarea grafurilor.

3.7.2.1 Terminologia algoritmilor genetici

O soluøie a unei probleme de optimizare este reprezentată de un *cromozom*. Pentru problemele cu grafuri, dimensiunea unui cromozom este egală adesea cu numărul de vârfuri ale grafului. Un cromozom este format dintr-un șir de simboluri numite *gene*. Fiecare genă are o valoare dintr-un alfabet *S*. De exemplu, în cazul problemei de bisecøie a grafurilor, fiecare vârf al grafului este reprezentat printr-o genă dintr-un cromozom. Fiecare genă reprezentând un vârf va ocupa aceeași poziøie în fiecare cromozom.

Un subset de gene aflate în anumite poziøii, și care formează o soluøie parøială a problemei, se numește *schemă*. Într-un mod mai formal, dacă *n* este dimensiunea cromozomului, o schemă este un *n*-tuplu $<s_1, s_2, \ldots, s_n>$, unde $s_i \in S \cup \{*\}$ pentru $i = 1, 2, \ldots, n$. Într-o schemă, simbolul $*$ reprezintă poziøii indiferente, iar celelalte simboluri (numite *simboluri specifice*) indică poziøiile definitorii ale subsetului și valorile corespunzătoare ale genelor. Numărul simbolurilor specifice dintr-o schemă este numit *ordin* al schemei. Lungimea dintre simbolul specific cel mai din stânga și simbolul specific cel mai din dreapta este numită *lungimea definitorie* a schemei.

Considerând o codificare binară de lungime 3, `101` și `001` sunt exemple de cromozomi, iar `10*` este o schemă reprezentând setul cromozomilor având valoarea `1` în prima poziøie și `0` în poziøia a doua. Aceasta este o schemă de ordin 2 și lungime definitorie 1. Un cromozom de lungime *n* poate fi considerat ca o instanøiere a 2^n scheme. De exemplu, cromozomul `101` este o instanøiere a $2^3 = 8$ scheme: `***`, `1**`, `*0*`, `**1`, `10*`, `1*1`, `*01` și `101`.

3.7.2.2 Principiul algoritmilor genetici

Un algoritm genetic pornește de la un set de soluøii iniøiale (cromozomi), care reprezintă o *populaøie*. Această populaøie evoluează apoi la alte populaøii diferite pe durata mai multor iteraøii (*generaøii*). În timpul fiecărei iteraøii indivizii din populaøia curentă sunt evaluaøi utilizând o anumită măsură de viabilitate. Pe baza acestei evaluări, sunt selectaøi doi membri ai populaøiei (numiøi *părinøi*). Asupra părinøilor selectaøi se aplică apoi un număr de operatori genetici pentru a genera noi soluøii numite *urmași*. Acest proces este cunoscut ca *reproducøie*, indivizii care sunt mai viabili având o probabilitate mai mare de reproducøie. Viabilitatea este legată de funcøia de cost, iar în cazul unei probleme de minimizare a unei funcøii, viabilitatea este de obicei inversul funcøiei de cost.

Operatorii genetici combină caracteristicile ambilor părinøi. Operatorii genetici cei mai utilizaøi sunt *încrucișarea* (*crossover*), *mutaøia* și *inversiunea*. Aceștia sunt deriváøi prin analogie cu procesul biologic al evoluøiei.

Încrucișarea este principalul operator genetic. Reprezintă un mecanism ereditar prin care un urmaș moștenește unele caracteristici ale părinøilor. Operaøia de încrucișare constă din alegerea unui punct de tăietură arbitrar și generarea urmașului prin combinarea segmentului din stânga punctului de tăietură al unui părinte cu segmentul din dreapta punctului de tăietură al celuilalt părinte. Se selectează deci o anumită

schemă dintr-un părinte și o altă schemă din celălalt părinte. Operația de încrucișare crează de asemenea noi scheme care nu sunt prezente în cadrul părinților.

Mutația produce modificări aleatoare într-un singur individ. Urmașul generat prin încrucișare este modificat cu un operator de mutație pentru a introduce un spațiu de căutare neexplorat pentru populație, crescând diversitatea acestei populații (gradul de diferență între cromozomii din populație). Mutația este similară cu interschimbarea perechilor din algoritmii convenționali de partiționare. Are rolul de a evita minimele locale și împiedică înlocuirea tuturor indivizilor din populație cu copii multiple ale unui singur individ.

Inversiunea este un operator care modifică reprezentarea unui individ fără modificarea efectivă a individului, astfel încât urmașul va moșteni cu o probabilitate mai mare anumite scheme de la un părinte. Inversiunea modifică lungimea efectivă a unei scheme fără a schimba măsura de viabilitate a individului, în scopul creșterii probabilității de supraviețuire a schemelor mai lungi.

După aplicarea operatorilor genetici, noii urmași sunt evaluați pentru a testa dacă sunt corespunzători pentru populație. Se execută apoi un proces de *înlocuire*. Pentru aceasta se selectează în mod probabilistic indivizi din vechea populație și din cadrul urmașilor, pe baza viabilității lor relative. Se dispune acum de o nouă populație, terminându-se un ciclu de reproducție-evaluare pentru o generație sau iterație a algoritmului genetic. Procesul de evoluție este repetat până când este satisfăcută o anumită condiție, de exemplu pentru un anumit număr de generații.

Există algoritmi genetici care generează un singur urmaș în fiecare generație. Un asemenea algoritm se numește *cu stări stabile*, spre deosebire de un algoritm *generațional*, care înlocuiește întreaga populație sau un subset de dimensiuni mari al populației în fiecare generație. O structură tipică a unui algoritm genetic cu stări stabile este prezentată în Figura 3.33. Dacă se adaugă o euristică de îmbunătățire locală, de obicei după mutație, se obține un algoritm genetic *hibrid*.

```
Algorithm Genetic;
begin
        Se crează populația inițială de dimensiune fixată;
        repeat
                Se alege părinte1 și părinte2 din populație;
                urmaș = crossover (părinte1, părinte2);
                mutation (urmaș);
                if viabil (urmaș) then
                        replace (populație, urmaș);
                endif
        until (condiție de terminare)
end.
```

Figura 3.33. Structura tipică a unui algoritm genetic cu stări stabile.

După cum s-a menționat în secțiunea 3.7.2.1, un cromozom este o instanțiere a 2^n scheme. O operație explicită asupra cromozomului poate fi considerată și ca o operație implicită asupra celor 2^n scheme. Deci, un algoritm genetic lucrează în mod explicit cu cromozomi, și în mod implicit cu un număr exponențial de scheme. Aceasta reprezintă un *paralelism implicit* sau *paralelism intrinsec* [31].

O teoremă importantă a algoritmilor genetici este numită teorema schemelor. Această teoremă specifică faptul că numărul instanțelor unei scheme într-o populație dată crește de la o generație la alta într-o măsură proporțională cu viabilitatea sa relativă față de o altă schemă din populația prezentă.

Într-un mod mai formal, fie $P(t)$ populația la momentul t. Se notează cu $m(H, t)$ numărul așteptat de cromozomi din populația $P(t)$ conținând o schemă H. O limită a

numărului așteptat de cromozomi din populația $P(t + 1)$ conținând schema H este dată de Teorema 3.7.2.2.

Teorema 3.7.2.2 [31]. Într-un algoritm genetic care utilizează o selecție proporțională și o încrucișare cu un singur punct, este adevărată următoarea inegalitate pentru fiecare schemă H reprezentată în $P(t)$:

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{f'(t)}\left[1 - P_c \frac{\delta H}{n-1}\right] \quad (3.47)$$

unde $f(H, t)$ este valoarea de viabilitate medie a cromozomilor conținând schema H la momentul t, $f'(t)$ este valoarea de viabilitate medie a tuturor cromozomilor din $P(t)$ la momentul t, P_c este rata de încrucișare, $\delta(H)$ este lungimea definitorie a schemei H, iar n este lungimea fiecărui cromozom.

Datorită importanței sale, această teoremă este numită și *Teorema fundamentală a algoritmilor genetici* [83]. Inegalitatea (3.47) arată că schemele de calitate mai ridicată au șanse mai mari de supraviețuire pe măsura trecerii generațiilor.

Demonstrarea convergenței algoritmilor genetici este dificilă, deoarece într-o singură operație de încrucișare sunt create și distruse un număr mare de scheme, iar viabilitatea unei scheme date relativ la restul populației se modifică în fiecare generație pe măsură ce populația se schimbă [124]. De exemplu, algoritmul genetic poate fi modelat ca un proces Markov. Fiecare stare din lanțul Markov corespunde unei anumite populații de indivizi. Atunci când dimensiunea populației și parametrii genetici (ratele de încrucișare și mutație) sunt constante, se obține un lanț Markov omogen în timp, cu o distribuție finală staționară a probabilităților stărilor. Aceasta nu garantează însă că soluția optimă globală va fi găsită.

O altă problemă a algoritmilor genetici este alegerea valorilor optime pentru dimensiunea populației, rata de încrucișare, rata de mutație, rata de inversiune, și a altor parametri. Rezultatele teoretice obținute de cercetători în acest sens sunt valabile pentru clase restrânse de probleme. Cu anumite criterii restrictive de convergență, s-a arătat că dimensiunea optimă a populației este 3 [124]. Însă, cu criterii mai puțin restrictive de convergență, dimensiunea optimă a populației este legată de lungimea n a reprezentării genetice a unui individ. Pentru un șir de lungime 50, dimensiunea optimă a populației este apropiată de 2000; pentru un șir de lungime 60, această dimensiune este de peste 8000.

3.7.2.3 Algoritm genetic de partiționare cu echilibrarea numărului de conexiuni

Fie un graf $G = (V, E)$ caracterizat printr-un set de n noduri V și un set de e muchii E. Algoritmul genetic pentru partiționare poate fi descris pe scurt astfel. Se pornește de la o populație inițială de partiții. În fiecare generație, se obțin partiții urmaș printr-un proces de încrucișare între două partiții părinte, cu o rată specifică de încrucișare. Se utilizează de asemenea un proces de mutație pentru a schimba compoziția structurală a unui număr redus de partiții din populație. Din partițiile inițiale și din cele generate prin încrucișare este selectat un set de partiții care va constitui populația generației următoare. Acest proces este continuat pe parcursul mai multor generații. În final, partiția cea mai bună din cadrul populației este aleasă ca soluție a problemei de partiționare.

Procesul de evoluție genetică modifică structura elementelor din cadrul populației. De aceea reprezentarea structurală a soluției este foarte importantă. Structura conține blocuri constructive numite gene. O soluție parțială este reprezentată de o schemă. Populația de soluții poate fi considerată ca o reprezentare colectivă a unui mare număr de scheme. Pe parcursul evoluției, populația se îmbogățește cu scheme mai viabile. Viabilitatea unei soluții se reflectă în costul funcției obiectiv care se asociază cu soluția respectivă.

Fiecare soluøie a problemei este reprezentatã printr-un cromozom, care este un şir binar. Un cromozom corespunde deci unei bisecøii a grafului. Numãrul de gene ale unui cromozom este egal cu *n*, numãrul de vârfuri ale grafului. O genã are valoarea 0 dacã vârful corespunzãtor se aflã în partea stângã a bisecøiei, şi are valoarea 1 dacã vârful se aflã în partea dreaptã a bisecøiei.

Pentru *iniøializare*, algoritmul creazã în mod aleator N_p soluøii, unde N_p este dimensiunea populaøiei. De obicei, o populaøie de dimensiune mai mare implicã o soluøie finalã de calitate mai bunã, dar şi un timp de execuøie mai mare. Presupunând cã numãrul de vârfuri ale grafului este par, singura restricøie asupra unui cromozom este cã acesta trebuie sã conøinã un numãr egal de valori 0 şi 1.

Fiecãrei soluøii din cadrul populaøiei i se asignează o valoare de viabilitate calculatã din dimensiunea tãieturii. Valoarea viabilitãøii F_i a soluøiei *i* se calculează astfel:

$$F_i = 1 / (Dim\_tăietură + PONDERE \times Dezechilibru) \quad (3.48)$$

unde *Dim_tãieturã* este dimensiunea tãieturii soluøiei, *PONDERE* este o constantã, iar *Dezechilibru* este diferenøa între numãrul de conexiuni din porøiunea din stânga a partiøiei şi numãrul de conexiuni din porøiunea din dreapta a partiøiei. Un cromozom este selectat ca un pãrinte cu o probabilitate care este proporøionalã cu valoarea sa de viabilitate. Aceasta este o metodã obişnuitã de selecøie a pãrinøilor, numitã *selecøie proporøionalã*.

Un *operator de încrucişare* creazã un nou cromozom urmaş prin combinarea unor pãrøi ale celor doi cromozomi pãrinte. Cel mai simplu operator de încrucişare selectează în mod aleator un punct de tãieturã, care este acelaşi pentru ambii cromozomi pãrinte. Acest punct divide cromozomul în douã pãrøi, partea stângã şi partea dreaptã. Partea stângã a pãrintelui 1 şi partea dreaptã a pãrintelui 2 sunt copiate în aceleaşi poziøii ale cromozomului urmaş. Fie acesta urmaşul 1.

Algoritmul propus utilizează şi un alt operator de încrucişare, care este acelaşi cu cel descris, cu excepøia faptului cã se copiază valorile complementare ale pãrøii drepte a pãrintelui 2, în timp ce partea stângã a pãrintelui 1 este copiatã nemodificat. Fie acesta urmaşul 2. Algoritmul selectează cel mai bun dintre cei doi urmaşi. Motivul pentru utilizarea a doi operatori de încrucişare este urmãtorul. Dacã doi cromozomi sunt exact (sau aproape exact) complementul unuia faøã de celãlalt, aceştia reprezintã aceeaşi (sau aproape aceeaşi) bisecøie. În acest caz, primul operator va crea o inconsistenøã într-un cromozom urmaş, şi în consecinøã acest cromozom va avea o calitate redusã.

Operatorul de mutaøie utilizat funcøionează astfel: sunt selectate în mod aleator *m* poziøii în cadrul cromozomului, şi valorile lor sunt inversate. Valoarea lui *m* este un întreg în intervalul [0, *n*/10]. Dupã operaøiile de încrucişare şi mutaøie, un urmaş poate avea un numãr diferit de valori 0 şi 1. Algoritmul calculează diferenøa dintre numãrul valorilor de 1 şi 0. Apoi selectează un punct aleator din cromozom şi modificã numãrul necesar de valori de 1 în 0 (sau 0 în 1) începând din acel punct spre dreapta (continuând de la stânga dacã este necesar). Această ajustare produce de asemenea un efect de mutaøie.

Dupã generarea unui nou urmaş, algoritmul înlocuieşte un membru al populaøiei cu acest urmaş. Calitatea soluøiilor depinde în mare mãsurã de *metoda de înlocuire*. Trebuie aleasã o metodã de înlocuire care generează soluøii de calitate într-un timp rezonabil.

În literaturã au fost propuse diferite metode de înlocuire. Whitley şi Kauth au sugerat, în cadrul sistemului *Genitor*, o metodã în care un urmaş înlocuieşte membrul cel mai inferior calitativ al populaøiei. Cu această metodã, algoritmul poate converge rapid, dar diversitatea populaøiei se poate reduce în mod semnificativ în primele generaøii. De aceea, calitatea soluøiilor poate fi necorespunzãtoare.

Cavicchio a propus o metodã numitã *preselecøie*, în care un urmaş înlocuieşte pãrintele numai dacã acesta este inferior calitativ, cu scopul de a menøine diversitatea

populaøiei. Această metodă conduce de obicei la rezultate mai bune decât înlocuirea de tip *Genitor*, deoarece poate menøine o mai mare diversitate a populaøiei. Totuši, această diversitate poate fi relativ redusă, mai ales dacă părintele superior calitativ este similar cu urmašul. Deoarece soluøiilor cu valori mari de viabilitate li se atribuie probabilităøi de selecøie mai mari, această metodă supraestimează soluøiile bune. Astfel, anumite scheme de calitate din soluøiile inferioare vor fi pierdute în cazul acestei metode.

Bui și Moon [32] au propus o metodă în care un urmaš încearcă mai întâi să înlocuiască părintele cel mai similar, pe baza distanøei Hamming, și, dacă nu reušește, încearcă înlocuirea celuilalt părinte. Dezavantajul acestei metode constă în faptul că este consumatoare de timp, deoarece o mare diversitate implică un timp mare pentru a se obøine convergenøa. În particular, rata urmašilor care vor fi pierduøi în generaøiile viitoare este extrem de ridicată. În etapele ulterioare ale algoritmului, se abandonează adesea peste 99% din urmašii generaøi. Pentru a se compensa acest efect, în [32] această metodă este combinată cu preselecøia. Cu toate acestea, în etapele ulterioare ale algoritmului timpul consumat inutil este realtiv ridicat, deoarece este dificilă generarea unui urmaš de calitate mai bună decât cel puøin unul din părinøi, atunci când majoritatea membrilor populaøiei sunt de calitate foarte bună.

În algoritmul de partiøionare descris, se combină metoda de înlocuire din [32] cu cea de tip *Genitor*. Se încearcă mai întâi înlocuirea părintelui cel mai similar, pe baza distanøei Hamming; dacă urmašul este de calitate mai slabă decât ambii părinøi, se înlocuiește membrul cel mai inferior calitativ al populaøiei. Scopul este de a se menøine diversitatea populaøiei, fără a se crește în mod semnificativ timpul consumat inutil. Rezultatele obøinute prin această metodă combinată sunt superioare celor obøinute prin metoda de tip *Genitor*.

Criteriul de terminare pentru un mare număr de algoritmi genetici este execuøia pentru un număr fix de generaøii. Un criteriu mai avantajos este terminarea algoritmului atunci când diversitatea populaøiei scade sub un anumit prag. Algoritmul se termină atunci când 80% a populaøiei este ocupată cu soluøii de aceeași calitate, ale căror cromozomi nu sunt neapărat aceeași. Numărul maxim de iteraøii este limitat la 2000.

3.7.3 Rezultate experimentale

Algoritmul de partiøionare propus pentru echilibrarea numărului de conexiuni a fost implementat în limbajul C. Experimentele au fost efectuate pe un calculator IBM PC cu un procesor Pentium de 133 MHz, sub sistemul de operare Windows NT Version 4.0. S-a utilizat un număr de nouă circuite de test din cadrul setului de circuite al centrului *MCNC* (*Microelectronics Center of North Carolina*). Lista de conexiuni a circuitelor de test a fost convertită din formatul *EDIF* (*Electronic Design Interchange Format*) în formatul listei de conexiuni utilizate de programul de partiøionare. Algoritmul de partiøionare a fost aplicat asupra listelor de conexiuni obøinute astfel.

În Tabelul 3.1 se prezintă circuitele de test care au fost utilizate. Prima coloană a tabelului conøine denumirea circuitului, iar următoarele coloane conøin numărul total de celule necesare (celule logice și celule de I/E), numărul de conexiuni și numărul de terminale ale fiecărui circuit.

Tabelul 3.1. Caracteristicile circuitelor *MCNC* utilizate pentru testare.

Circuit	Nr. celule	Nr. conexiuni	Nr. terminale
b1	28	24	63
c17	23	17	37
cm138a	43	32	92
con1	33	29	74
daio	30	28	71
decod	75	59	179

Circuit	Nr. celule	Nr. conexiuni	Nr. terminale
majority	21	16	45
tcon	82	66	171
x2	58	49	155

Tabelul 3.2 prezintã rezultatele experimentelor efectuate în care s-a urmãrit care este valoarea tãieturii obøinute în urma bipartiøionãrii fiecãrui circuit în douã cazuri. Primul caz este cel în care nu se urmãreşte echilibrarea numãrului de conexiuni din cele douã partiøii (*PONDERE* = 0), caz indicat în tabel prin *P* = 0. Al doilea caz este cel în care se realizeazã şi echilibrarea numãrului de conexiuni, termenul corespunzãtor din funcøia de cost având aceeaşi pondere cu termenul pentru minimizarea lungimii conexiunilor (*PONDERE* = 1), caz indicat în tabel prin *P* = 1. Valorile s-au obøinut prin rularea algoritmilor de 20 de ori, calculându-se media tãieturilor pentru aceste rulãri (valorile fiind rotunjite la numere întregi).

Tabelul 3.2. Dimensiunea tãieturii în urma bipartiøionãrii circuitelor.

Circuit	Dimensiune tãietură	
	P = 0	**P = 1**
b1	9	11
c17	4	6
cm138a	12	16
con1	12	10
decod	12	17
majority	10	14
tcon	11	17
x2	14	16

Din Tabelul 3.2 se observã cã în cazul în care se urmãreşte şi echilibrarea numãrului de conexiuni din cele douã partiøii (*P* = 1) se obøine o anumitã creştere a dimensiunii tãieturii faøã de cazul în care se urmãreşte numai minimizarea lungimii conexiunilor (*P* = 0). Aceste lucru este explicabil, deoarece în cazul în care *P* = 0, singura metricã utilizatã este cea care indicã dimensiunea tãieturii. În cazul bipartiøionãrii, acestã dimensiune este cea a tãieturii din centrul circuitului. Pentru a mãsura dimensiuea tãieturii şi în alte zone ale circuitului, algoritmul de bipartiøionare a fost aplicat recursiv pânã la obøinerea unor zone care conøin o singurã celulã, urmãrindu-se care este suma tãieturilor în cele douã cazuri: *P* = 0 şi *P* = 1. Rezultatele sunt prezentate în Tabelul 3.3.

Tabelul 3.3. Suma tãieturilor în urma aplicãrii recursive a algoritmului de bipartiøionare.

Circuit	Suma tãieturilor		Reducere
	P = 0	**P = 1**	
b1	19	17	10.5 %
c17	10	10	0.0 %
cm138a	38	38	0.0 %
con1	30	28	6.6 %
decod	69	59	14.5 %
majority	17	15	11.7 %
tcon	60	51	15.0 %
x2	51	46	9.8 %

Din Tabelul 3.3 rezultã cã pentru şase din cele opt circuite s-a obøinut o reducere a sumei tãieturilor, pe lângã echilibrarea numãrului de conexiuni. Reducerea medie pentru circuitele de test utilizate este de 8.5 %.

3.8 Concluzii

În acest capitol a fost prezentatã o problemã importantã care apare în cadrul proiectãrii fizice, şi anume partiøionarea circuitelor. Aceastã problemã a fost prezentatã atât ca o etapã de proiectare pentru divizarea unui sistem în mai multe pãrøi care pot fi implementate prin componente separate, cât şi ca o metodã algoritmicã pentru rezolvarea problemelor complexe de optimizare care apar în sinteza logicã sau în proiectarea fizicã a circuitelor VLSI în general şi FPGA în particular. Accentul este pus pe partiøionarea circuitelor FPGA cu resurse limitate de rutare.

Au fost definite formulãrile problemei de bipartiøionare şi a celei de multipartiøionare, fiind prezentate principalele restricøii care pot fi impuse unei probleme de partiøionare. Metodele de partiøionare au fost clasificate ca fiind, pe de o parte, *deterministice* şi *stohastice*, iar pe de altã parte, ca *iterative* sau *constructive*. Au fost descrise sintetic numeroase metode de partiøionare întâlnite în literaturã, atât pentru circuitele VLSI, cât şi FPGA.

Problema de bipartiøionare în care cele douã partiøii au dimensiuni egale a fost examinatã mai detaliat, datoritã importanøei sale practice. Aceastã partiøionare este utilizatã în cadrul procedurii de plasare pe baza tãieturii minime, care este descrisã în capitolul 4. În plus, un algoritm de bipartiøionare poate fi utilizat pentru obøinerea unei proceduri de partiøionare cu k cãi, prin aplicarea recursivã a algoritmului de bipartiøionare de $\log_2 k$ ori. Avantajul acestei metode de multipartiøionare este cã procedura de bipartiøionare utilizatã este mai simplã decât o procedurã directã de multipartiøionare, dar are dezavantajul cã valoarea lui k este presupusã ca o putere a lui 2.

Algoritmul Kernighan-Lin este unul din cele mai utilizate pentru rezolvarea problemei de bipartiøionare. Este un algoritm cu îmbunãtãøire iterativã, care poate fi extins şi pentru rezolvarea unor probleme mai generale de partiøionare. Dintre acestea au fost prezentate urmãtoarele cazuri: blocuri cu dimensiuni inegale, elemente cu dimensiuni inegale, şi partiøionarea cu k cãi. O extindere a algoritmului Kernighan-Lin şi o implementare mai eficientã a acestuia a fost realizatã de Fiduccia şi Mattheyses, euristica acestora luând în considerare atât conexiunile multipin, cât şi dimensiunile elementelor de circuit.

Cãlirea simulatã este o metodã stohasticã de îmbunãtãøire iterativã utilizatã pentru rezolvarea diferitelor probleme de optimizare, inclusiv pentru cea de partiøionare. Un avantaj important al metodelor stohastice este cã pot evita minimele locale. Prin metoda de cãlire simulatã se obøin anumite avantaje în privinøa calitãøii soluøiei, dar timpul de calcul consumat poate fi foarte ridicat.

Partiøionarea prin metoda tãieturii proporøionale se bazeazã pe o metricã propusã de Wei şi Cheng, care s-a dovedit o funcøie obiectiv de succes pentru numeroase aplicaøii. Aceastã metodã are tendinøa de a identifica grupãrile naturale din circuit. Euristica de partiøionare propusã de aceiaşi autori se bazeazã pe algoritmul Fiduccia-Mattheyses.

Deoarece performanøele algoritmului Kernighan-Lin sunt dependente de alegerea partiøiei iniøiale, pentru a se evita blocarea în minime locale este necesar un numãr mare de rulãri ale algoritmului asupra unor partiøii iniøiale generate aleator. Cheng şi Wei au propus o metodã de bipartiøionare cu performanøe stabile, care nu necesitã generarea unui mare numãr de configuraøii iniøiale. Se utilizeazã o tehnicã de partiøionare recursivã de sus în jos, împãrøindu-se întregul circuit în grupuri mici, puternic conectate, care sunt apoi rearanjate în douã subseturi care respectã restricøia de dimensiune. Rezultatele sunt semnificativ mai bune faøã de cele obøinute prin algoritmul Fiduccia-Mattheyses.

Metodele spectrale propuse în ultimii ani utilizeazã vectori proprii şi valori proprii ale matricii de adiacenøã a grafului care descrie circuitul. Vectorul propriu al valorii proprii minime diferite de zero a matricii poate fi interpretat ca o plasare liniarã

sau ordonare a nodurilor grafului. Această ordonare poate fi divizatã pentru a obøine o partiøionare a nodurilor. În literaturã au fost publicate numeroase modificări ale acestei metode de bazã. Prin metoda spectralã a fost raportatã o îmbunătãøire medie de 9% a raportului de tãieturã faøã de rezultatul obøinut prin metoda tãieturii proporøionale.

Metodele bazate pe fluxul în reøele utilizează fluxul direcøionat al semnalelor pentru îmbunătãøirea performanøelor sistemului. Diferitele metode propuse au în comun faptul cã este generat un model al grafului din lista de conexiuni direcøionatã pentru a determina un flux maxim, care este echivalent cu o tãieturã minimã. În literaturã au fost raportate rezultate obøinute prin aceste metode care sunt mai bune comparativ cu cele ale unor euristici de tip Kernighan-Lin, cât şi cu cele ale unor metode spectrale.

A fost descrisã de asemenea o metodã probabilisticã de partiøionare, care poate determina implicaøiile globale şi viitoare ale mutării unui nod în orice etapã a procesului de partiøionare. Fiecărui nod i se asociază o probabilitate a evenimentului nodul respectiv sã fie mutat efectiv în celãlalt subset în pasul curent al procesului de partiøionare. Din această probabilitate se calculează câştigurile potenøiale ale nodurilor, ceea ce reprezintã o indicaøie corectã a beneficiului care se obøine în urma mutării acestora în celãlalt subset.

Au fost descrise şi unele metode neconvenøionale de partiøionare: partiøionarea prin evoluøie stohasticã şi cea prin automate de învãøare. Metoda de evoluøie stohasticã descrisã pentru soluøionarea problemei de bipartiøionare presupune cã nodurile circuitului au dimensiuni diferite. A fost descrisã de asemenea extinderea algoritmului de bipartiøionare pentru problema de multipartiøionare. Automatul de învãøare descris este numit automat pentru migrarea obiectelor. Acest automat modificã stările tuturor obiectelor, spre deosebire de automatul tradiøional la care obiectele sunt trecute dintr-o stare în alta. Atunci când acest automat este utilizat pentru rezolvarea problemei de partiøionare, o soluøie nu este definitã prin starea curentã a automatului, ci prin întreaga structurã a acestuia.

În cadrul capitolului s-au propus doi algoritmi de bipartiøionare pentru circuitele FPGA cu resurse limitate de rutare. Primul algoritm se bazează pe metoda tãieturii minime, şi urmãreşte echilibrarea numãrului de conexiuni din cadrul partiøiilor, minimizând în acelaşi timp lungimea totalã a interconexiunilor. A fost propusã o metricã mai adecvatã pentru partiøionarea utilizatã la plasarea circuitelor FPGA la care principalul obiectiv este asigurarea rutabilitãøii. Experimentele efectuate aratã cã prin aplicarea acestui algoritm se obøine o reducere a dimensiunii tãieturii, ceea ce va avea ca efect reducerea necesarului de resurse de interconectare a circuitului FPGA, atunci când algoritmul este utilizat pentru rezolvarea problemei de plasare.

Al doilea algoritm propus este un algoritm genetic pentru partiøionare, cu un obiectiv similar cu primul algoritm. Algoritmul utilizează doi operatori de încrucişare în loc de unul singur, al doilea operator fiind prevãzut pentru situaøia în care doi cromozomi sunt exact complementul unuia faøã de celãlalt, şi deci aceştia reprezintã aceeaşi bisecøie. În acest caz, primul operator ar crea un cromozom urmaş cu o calitate redusã. Metoda de înlocuire utilizatã este diferitã de cea a algoritmilor tradiøionali, având ca scop principal menøinerea diversitãøii populaøiei. Criteriul de terminare este de asemenea diferit, algoritmul fiind terminat atunci când diversitatea populaøiei scade sub un anumit prag. Algoritmul a fost comparat cu un algoritm de partiøionare prin metoda cãlirii simulate. Experimentele au arãtat cã timpul de execuøie al algoritmului genetic este mai redus, rezultatele obøinute fiind comparabile cu cele obøinute prin metoda cãlirii simulate.

Contribuøiile acestui capitol sunt următoarele:

- Propunerea unei metrici mai adecvate pentru partiøionarea utilizatã la plasarea circuitelor FPGA cu resurse limitate de rutare;

- Elaborarea şi testarea unui algoritm de partiøionare care urmãreşte echilibrarea numãrului de conexiuni din cadrul partiøiilor, minimizând în acelaşi timp lungimea totalã a interconexiunilor;
- Elaborarea şi testarea unui algoritm genetic pentru partiøionare, cu un timp de execuøie mai redus decât cel al unui algoritm bazat pe metoda cãlirii simulate, calitatea soluøiilor obøinute fiind comparabilã cu cea obøinutã prin metoda cãlirii simulate.