Universitatea "Babeş-Bolyai" Cluj-Napoca
Facultatea de Matematică şi Informatică

Rezumatul tezei de doctorat

Algoritmi evolutivi în VLSI-CAD

Doctorand:
Doina Logofătu

Conducător ştiinţific:
Prof. Univ. Dr. **D. Dumitrescu**

2010

Cuprins

1

Structură, cuvinte cheie, contribuții originale

1.1 Structura lucrării

Teza conține aspecte teoretice şi experimentale privitoare la două probleme de optimizare actuale din domeniul *Very Large Scale Integration* (VLSI), mai exact privind ordonarea şi compactarea datelor. O serie de caracteristici şi proprietăți reies imediat din descrierea problemei şi ele sunt prezentate detaliat în secțiunile introductive ale capitolelor respective. *Data Ordering Problem* (DOP), *Data Ordering Problem with Inversion* (DOPI) şi *Data Compaction Problem* (DCP) sunt NP-dificile.

După primul capitol introductiv, următoarele două conțin aspecte de bază privind problemele de optimizare şi calculul evolutiv.

Al patrulea capitol conține aspecte, metode şi experimente privitoare la Problema Ordonării Datelor cu şi fără Inversiune. Datorită extinderii folosirii dispozitivelor pe bază de baterii (de ex. PDA-uri,

laptopuri), aspectul diminuării energiei consumate devine unul dintre ţelurile principale din domeniul proiectării circuitelor ultimilor ani. În acest context, Problema Ordonării Datelor cu şi fără Inversiune (*Data Ordering Problem with/without Inversion*) – DOP(I) – este foarte importantă. Cuvintele binare trebuie să fie ordonate şi (eventual) negate cu scopul de a minimiza numărul total de tranziţii. Această problemă are multiple aplicaţii în practică, cum ar fi de exemplu planificarea instrucţiunilor, optimizarea compilatoarelor, secvenţierea modelelor de test sau *cache write-back*. Aplicaţii utile ale problemei se regăsesc şi în biologia computaţională. Pe lângă o serie de abordări deja existente, autoarea propune o serie de noi algoritmi pentru ambele probleme DOP şi DOPI, printre care şi trei evolutivi, precum şi noi operatori genetici. Sunt propuşi: un algoritm de tip greedy eficient – *Greedy Min Simplified* – GMS_DOPI [Log06c, Log08d, Log09a], doi algoritmi evolutivi nedistribuiţi MUT_DOPI (algoritm evolutiv cu mutaţie) şi GA_DOPI (algoritm genetic hibrid) [Log06a], un algoritm genetic paralel folosind noua paradigmă MapReduce – *Map Reduce Parallel Genetic Algorithm* – MRPGA_DOPI [Log10c]. Noii operatori genetici propuşi – *Simple Cycle Inversion Operator* (SCIM), *Cycle PMX* (CPMX) şi *Cycle OX* (COX) [Log06a] – sunt folosiţi în cadrul algoritmilor MUT_DOPI şi GA_DOPI. A treia abordare, cea distribuită folosind paradigma MapReduce, este cea mai eficientă din punct de vedere al rezultatelor. În plus, este şi capabilă sa manipuleze date de dimensiuni foarte mari. Se oferă deasemenea rezultatele unor serii de experimente variate, în formă de tabele, grafice şi teste statistice, care compară metodele propuse cu cele prezente în literatură şi certifică eficienţa lor. Autoarea oferă şi un proiect ştiinţific de cercetare *Open Source*, disponibil la adresa: http://dopisolver.sourceforge.net/.

Al cincilea capitol prezintă algoritmi şi rezultate privitoare la Problema Compactării Datelor. În acest capitol se propun o serie de algoritmi originali pentru rezolvarea problemei compactării datelor, *Test Compaction Problem* (TCP). Această problemă importantă este deasemenea NP-dificilă. Ea se referă la activitatea de testare a circuitelor integrate şi joacă un rol important în proiectarea acestora. Algoritmii propuşi sunt doi de tip Greedy, *Greedy Naive Algorithm* (GRNV_TCP) şi *Greedy Binary Tree Algorithm* (GRBT_TCP) [Log08a, Log08e], un algoritm evolutiv eficient *Genetic Algorithm Test Compaction* (GA_TCP) [Log08b, Log09c] şi un algoritm evolutiv distribuit folosind noua tehnologie MapReduce, *MapReduce Parallel Evolutionary Algorithm* (MRPEA_TCP) [Log10b]. Deasemenea, se oferă şi un proiect ştiinţific *Open Source* aflat la adresa: http://dcpsolver.sourceforge.net/.

Al şaselea şi ultimul capitol al tezei conţine o serie de concluzii şi direcţii de cercetare, urmat de o listă bibliografică cuprinzătoare. Aceasta din urmă conţine peste 130 de surse folosite în elaborarea tezei, înrudite cu aspectele abordate.

Lucrarea se încheie cu patru anexe care conţin instrucţiuni cu referire la modul de utilizare a proiectelor *Open Source* dezvoltate, precum şi o serie de experimente extinse pentru ambele probleme DOPI şi TCP.

1.2 Cuvinte cheie

O serie de cuvinte cheie privind temele abordate în teză: probleme NP-dificile, proiectarea circuitelor integrate, VLSI-CAD, *Backtracking*, *Greedy*, grafuri, optimizare, teoria complexităţii, calcul evolutiv, *Traveling Salesman Problem* (TSP), *Set Covering Problem* (SCP), diminuarea energiei, activitatea de comutare, tranziţie, numărul total de tranziţii, *Greedy Min* (GM), *Greedy Min Simplified* (GMS), algoritmi genetici, algoritmi evolutivi, algoritmi distribuiţi, algoritmi genetici distribuiţi, operatori genetici, operatori de mutaţie, operatori de încrucişare, operatori de căutare, funcţie de evaluare, funcţia de selecţie, tehnologia MapReduce, funcţiile *Map* şi *Reduce*, *Apache Hadoop*, proiecte *Open Source*, *Data Ordering Problem* (DOP), *Data Ordering Problem with Inversion* (DOPI), *Test Compaction Problem* (TCP), biologia computaţională, acid deoxyribonucleic (ADN), algoritmii random şi exact, algoritm optimal, teste statistice *student-t*, ipoteza H_0, experimente numerice, mărginirea inferioară (*lower bound*), *Partially Mapped Crossover* (PMX), *Cycle PMX* (CPMX), *Ordered Crossover* (OX), *Cycle OX* (COX), simbol Don't Care, paradigma *bus-invert*, factor de compactare, *dopiSolver*, *dcpSolver*.

1.3 Contribuţii originale şi publicaţii

DOP şi *DOPI* se referă la un aspect important privind optimizarea disipării energiei în circuitele integrate, iar *DCP* se referă la optimizarea testării statice a acestora. Pentru ambele probleme se oferă diverşi algoritmi, dar şi o serie de idei privind continuarea cercetării lor. Printre algoritmii cercetaţi, pe lângă metode de tip *greedy*, se numără şi algoritmi evolutivi eficienţi. Pentru ambele probleme se propun abordări distribuite cu ajutorul tehnologiei MapReduce. Problemele cercetate se extind interdisciplinar şi în domeniul biologiei computaţionale, de exemplu privind ordonarea sau compactarea moleculelor ADN. Rezultate elocvente privind acest domeniu, susţinute de grafice şi observaţii specifice au fost prezentate în două lucrări la conferinţa internaţională BICS 2008. Versiuni extinse ale acestora au fost publicate în jurnalul *American Institute of Physics*.
Aceste rezultate au fost prezentate în cadrul unor conferinţe internaţionale, precum EvoHOT 2006, ROSYCS 2006, ISMVL 2008, DAS 2008, BICS 2008. Contribuţii originale expuse în lucrare:

- formalizarea unor probleme specifice din VLSI-CAD: DOP, DOPI şi DCP (4.2, 4.3, 5.2, 5.3);

- GMS_DOPI: un nou algoritm de tip greedy [Log06c] – (căutare locală) pentru DOPI, cu rezultate mai bune decât cel utilizat în industrie până în prezent – Greedy Min (4.5.4);
- MUT_DOPI: un algoritm evolutiv simplu de mutație pentru DOP/DOPI, care duce la îmbunătățiri relativ la toate metodele Greedy, folosind un nou operator de mutație SCIM [Log06a] (4.6.1);
- GA_DOPI: un algoritm genetic eficient pentru DOP/DOPI, care conduce la rezultate mai calitative decât cele de tip Greedy, folosind doi noi operatori de încrucișare Cycle PMX și Cycle OX [Log06a] (4.6.2);
- MRPGA_DOPI: un algoritm genetic distribuit pentru DOP/DOPI folosind tehnologia MapReduce pentru DOP/DOPI [Log10c] (4.6.3);
- rezultate experimentale pentru a compara îmbunătățirile aduse de gradul de libertate DOPI comparativ la DOP. Metodele Greedy vs. DOPI, algoritmi Greedy vs. algoritmii GA_DOPI și MRPGA_DOPI [Log06a, Log06c, Log08d, Log09a, Log10b] (4.6.3);
- teste statistice pentru a compara metodele Greedy relativ la cele evolutive [Log06a] (4.7);
- GRNV_DCP și GRBT_DCP: doi algoritmi de tip Greedy pentru DCP [Log08a, Log08e, Log09b] (5.7);
- GA_DCP: un algoritm genetic eficient pentru DCP [Log08b, Log09c] (5.8);
- MRPGA_DCP: un algoritm genetic paralel pentru DCP folosind tehnologia MapReduce [Log10c] (5.9);
- rezultate experimentale și grafice pentru a compara algoritmii propuși în perechi [Log08a, Log08b, Log09b, Log09c, Log10b] (5.10);
- teste statistice pentru a compara metodele Greedy cu cele evolutive [Log10b] (5.10);
- *dopiSolver* și *dcpSolver*: *framework*-uri de cercetare pentru DOP/DOPI și DCP, pentru a rula/extinde diferiții algorimi, aflate pentru utilizarea în scopuri științifice la adresele http://dopisolver.sourceforce.net, respectiv http://dcpsolver.sourceforce.net;
- noi direcții de cercetare pentru ambele probleme abordate în teză (4.9, 5.11).

O listă cu publicațiile care se referă explicit la cele două probleme de optimizare este dată mai jos.

Cărți:

Logofătu, D.: *Algorithmen und Problemlösungen mit C++*, pp. 402-411, Vieweg+Teubner-Verlag, 2010 (Germania).

Logofătu, D.: *Eine praktische Einführung in C*, pp. 207-208, entwickler.press, München, Germania, 2008 (Germania).

Logofătu, D.: *Grundlegende Algorithmen mit Java*, pp. 65-98:205-214, Vieweg-Verlag, Germany, 2008 (Germania).

Logofătu, D.: *Algoritmi fundamentali in C++. Aplicații*, pp. 127-154:265-273, Editura Polirom, Iaşi, 2007.

Logofătu, D.: *Algoritmi fundamentali in Java. Aplicații*, pp. 125-158:269-277, Editura Polirom, Iaşi, 2007.

Lucrări pentru conferințe internaționale:

Logofătu, D., Dumitrescu, D.: Distributed Genetic Algorithm for Data Ordering Problem with Inversion Using MapReduce, *11th International Conference on Parallel Problem Solving from Nature*, PPSN, trimisă spre publicare, Aprilie 2010.

Logofătu, D., Dumitrescu, D.: Parallel Evolutionary Approach of Compaction Problem using MapReduce, *11th International Conference on Parallel Problem Solving from Nature*, PPSN, trimisă spre publicare, Aprilie 2010.

Logofătu, D.: On the Compaction of DNA Sequence Vectors, *Proceedings Bio-Inspired Computational Methods Used for Difficult Problems Solving. Development of Intelligent and Complex Systems* (BICS 2008), pp. 25-36, Târgu Mureş, România, 2008.

Logofătu, D., Gruber, M.: Efficient Approaches for DNA Sequences Ordering, *Proceedings Bio-Inspired Computational Methods Used for Difficult Problems Solving. Development of Intelligent and Complex Systems* (BICS 2008), pp. 59-69, Târgu Mureş, România, 2008.

Logofătu, D.: Efficient Evolutionary Approach for the Test Compaction Problem, *Proceedings 9th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS*, pp. 144-148, Suceava, Romania, May 22-24, 2008.

Logofătu, D., Drechsler, R.: Comparative Study by Solving the Test Compaction Problem, *Proceedings 38th International Symposium on Multiple-Valued Logic (ISMVL '08)*, Dallas, USA, pp. 44-49, 2008.

Logofătu, D.: Greedy Approaches for the Data Ordering Problem with Inversion, *Proceedings of ROSYCS* - Romanian Symposium on Computer Science, pp. 65-80, Iaşi, 2006.

Logofătu, D., Drechsler, R..: Efficient Evolutionary Approaches for the Data Ordering Problem with Inversion, *3rd European Workshop on Hardware Optimisation Techniques (EvoHOT)*, LNCS 3907, pp. 320-331, Springer, Berlin/Heidelberg, 2006.

Drechsler, R., Drechsler, N.: Minimization of Transitions by Complementation and Resequencing using Evolutionary Algorithms, In *Proceedings of 21st IASTED International Multi-Conference Applied Informatics (AI 2003)*, Innsbruck, 2003.

Drechsler, N., Drechsler, R.: Exploiting don't cares during data sequencing using genetic algorithms, *ASP Design Automation Conf.*, pp. 303-306, 1999.

Murgai, R., Fujita, M., Oliveira, A.: Using complementation and resequencing to minimize transitions, *Design Automation Conf.,* pp. 694-697, 1998.

Murgai, R., Fujita, M., Krishnan, S. C.: Data sequencing for minimum-transition transmission, *IFIP Int'l Conf. on VLSI,* 1997.

Stan, M., Burleson, W.: Limited-weight codes for low-power I/O, *Int'l Workshop on Low Power Design,* 1994.

Jurnale, reviste:

Logofătu, D.: Static Test Compaction for VLSI Tests: an Evolutionary Approach, *Advances in Electrical and Computer Engineering*, Vol. 8, Nr. 2, pp. 49-53, Romanian Academy of Technical Sciences, 2008.

Logofătu, D.: DNA Sequences and their Compaction, BICS 2008: Proceedings of the 1st International Conference on Bio-Inspired Computational Methods Used for Difficult Problems Solving: Development of Intelligent and Complex Systems. AIP Conference Proceedings, Vol. 1117, Nr. 1, pp. 29-39, *American Institute of Physis*, 2009.

Logofătu, D., Gruber, M.: DNA Sequences and their Ordering, BICS 2008: Proceedings of the 1st International Conference on Bio-Inspired Computational Methods Used for Difficult Problems Solving: Development of Intelligent and Complex Systems. AIP Conference Proceedings, Vol. 1117, Nr. 1, pp. 3-11, *American Institute of Physis*, 2009.

Pentru rezolvarea problemelor DOP/DOPI şi DCP au fost realizate proiecte *Open Source* de cercetare, disponibile la adresele http://dopiSolver.sourceforge.net şi http://dcpSolver.sourceforge.net.

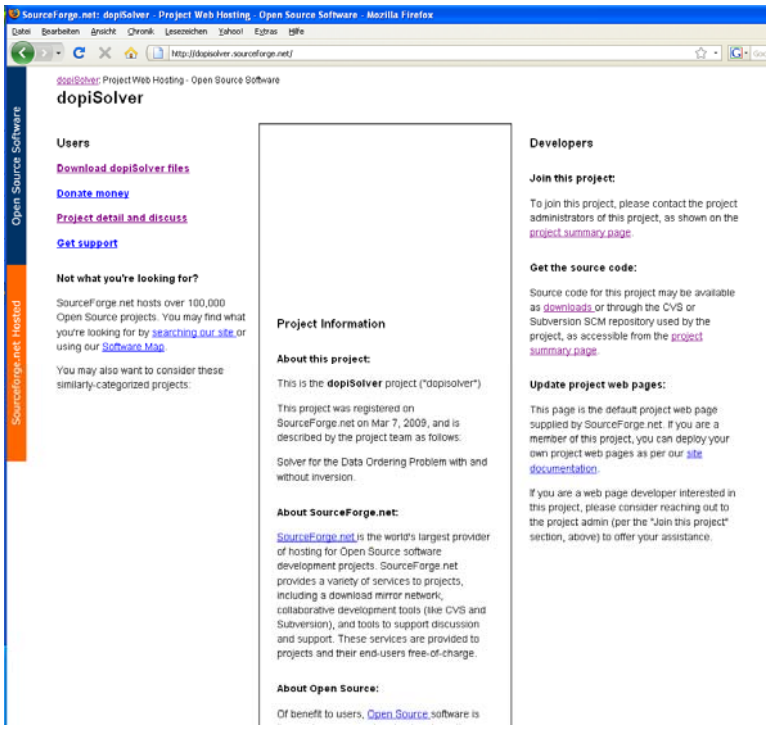


Fig. 1.1. http://dopisolver.sourceforge.net/

Aceste *framework*-uri sunt flexibile şi pot fi folosite ca atare, pentru a rula algoritmii DOP/DOPI/DCP pe diferite instanţe de date. Oferă şi facilităţi de extindere cu noi abordări ale problemei.

Anexele conţin modul de utilizare a proiectelor *dopiSolver* şi *dcpSolver* de pe serverul *sourgeforce.net* şi seturi detaliate de experimente.

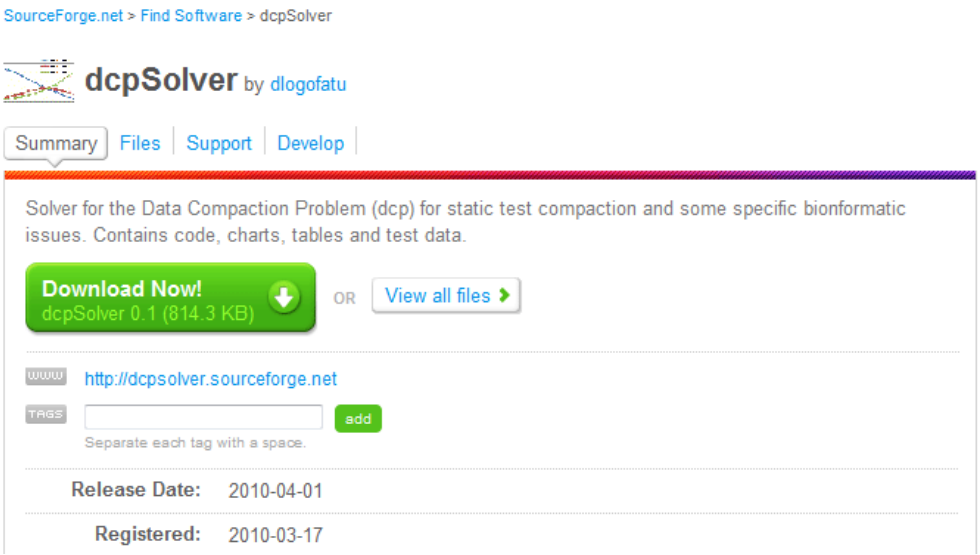


Fig. 1.2. http://sourceforge.net/projects/dcpsolver/

2

MapReduce

2.1 Paradigma MapReduce

Tehnologia MapReduce este un model distribuit de programare introdus de Google. În cadrul acestui model utilizatorul specifică calculul cu ajutorul a două funcții, *Map* şi *Reduce* [Dea08, Eka08, Jin08, Llo10].

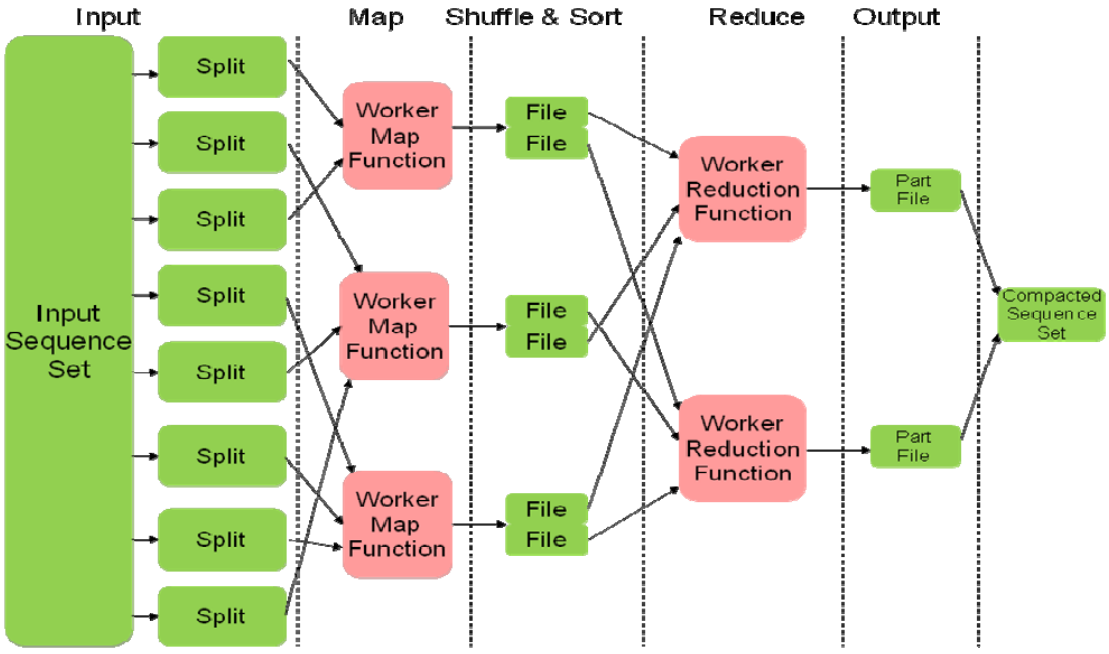


Fig. 2.1. Dataflow MapReduce

Tehnologia MapReduce se foloseşte de aceste două abstractizări, *Map* şi *Reduce*, pentru a permite dezvoltarea unor aplicaţii distribuite pe scală largă, atât timp cât aceste aplicaţii o permit. Conceptual, funcţiile *Map* şi *Reduce* oferite de utilizator trebuie să aibă următoarea formă:

$$Map(k_1, v_1) \rightarrow List(k_2, v_2)$$

$$Reduce(k_2, List(v_2)) \rightarrow List(v_3)$$

Apelurile *Map* sunt distribuite de-a lungul mai multor maşini prin partiţionarea automată a datelor de intrare. Submulţimi ale datelor de intrare sunt procesate în paralel pe maşini diferite. Apelurile *Reduce* sunt distribuite deasemenea folosind o funcţie de partiţionare.

2.2 Avantajele utilizării Apache Hadoop

Apache Hadoop este o platformă Java *Open Source* ce permite rularea de aplicaţii pentru prelucrarea paralelă şi distribuită de volume mari de date, de preferinţă pe clustere cu multe noduri ce rulează pe configuraţii hardware obişnuite (*commodity hardware*). Platforma implementează paradigma MapReduce, oferind paralelizarea şi distribuirea automată a celor doi paşi (Map şi Reduce) pe care aplicaţiile rulate trebuie să îi implementeze.

Căderile unor noduri din cluster sunt "tolerate" de sistem prin redistribuirea *task*-urilor asignate acestora către alte noduri active. Supravegherea resurselor şi operaţiilor active se poate realiza cu ajutorul aplicaţiilor de monitorizare furnizate. Un *Job* MapReduce împarte datele de intrare în secţiuni logice independente (*splits*), care sunt trimise la unul sau mai multe noduri pentru executarea funcţiei *map*. Perechile de chei/valori rezultate sunt trimise nodurilor care execută funcţia *reduce*.

Hadoop vine cu un sistem distribuit de fişiere, *Hadoop Distributed File System* (*HDFS*), care crează replici multiple ale blocurilor de date şi le distribuie pe nodurile din cluster. Nodurile care prelucrează datele şi cele care le stochează sunt în mod normal aceleaşi. Acest fapt permite sistemului să repartizeze operaţiile pe nodurile unde datele există deja, ceea ce măreşte semnificativ viteza de agregare şi sortare a datelor. Pe platforma Hadoop rulează un singur process *master* `JobTracker` şi câte un process `JobTracker` *slave* pe fiecare nod din cluster. *Master*-ul e responsabil cu distribuţia de operaţii *map* şi *reduce* pe nodurile *slave*. Aplicaţiile Hadoop construiesc *job*-uri prin specificarea locaţiilor HDFS pentru datele de intrare şi ieşire, furnizarea implementărilor pentru operaţiile *map* şi *reduce*, precum şi alte informaţii contextuale (invarianţi). *Job client*-ul Hadoop trimte informaţiile serializate (archiva *jar* cu byte code-ul java şi configuraţia job-ului) la `JobTracker`, care apoi le distribuie la nodurile *slave*, după care planifică, distribuie şi monitorizează executarea operaţiilor *map* şi *reduce*, oferind informaţii privind progresul lor şi starea sistemului *Job client*-ului.

3

Problema Ordonării Datelor

Pe lângă o serie de abordări deja existente, autoarea propune o serie de noi algoritmi pentru ambele probleme DOP şi DOPI, printre care şi trei evolutivi, precum şi noi operatori genetici. Se propun un algoritm de tip greedy eficient – *Greedy Min Simplified* – GMS_DOPI [Log06c, Log08d, Log09a], doi algoritmi evolutivi nedistribuiți MUT_DOPI (algoritm evolutiv cu mutație) și GA_DOPI (algoritm genetic hibrid) [Log06a], un algoritm genetic paralel folosind noua paradigmă MapReduce – *Map Reduce Parallel Genetic Algorithm* – MRPGA_DOPI [Log10c]. Noii operatori genetici propuși – *Simple Cycle Inversion Operator* (SCIM), *Cycle PMX* (CPMX) și *Cycle OX* (COX) [Log06a] – sunt folosiți în cadrul algoritmilor MUT_DOPI și GA_DOPI. A treia abordare, cea distribuită folosind paradigma MapReduce, este cea mai eficientă din punct de vedere al rezultatelor. În plus, este și capabilă sa manipuleze date de dimensiuni foarte mari. Se oferă deasemenea rezultatele unor serii de experimente variate, în formă de tabele, grafice și teste statistice, care compară metodele propuse cu cele prezente în literatură și certifică eficiența lor. Autoarea oferă și un proiect științific de cercetare *Open Source*, disponibil la adresa: http://dopisolver.sourceforge.net/.

3.1 Descrierea problemelor DOP și DOPI

Definiția 3.1. Numim *tranziție* poziția unui bit în care două secvențe binare diferă pe această poziție.

Definiția 3.2. Numim *inversarea* (*complementarea*) unui cuvânt binar operația prin care fiecare bit al cuvântului este înlocuit cu negatul său ($0 \to 1$, $1 \to 0$). Notăm cu $\overline{w}$ inversul (complementul) cuvântului w.

Exemple: $w = 10110001$, $\overline{w} = 01001110$ şi $\overline{10110} = 01001$.

Pentru problema DOPI (*DOP with Inversion*) permitem ca unele cuvinte să fie inversate la destinatar.

Definiția 3.3. Dacă un cuvânt w_r este transmis urmat imediat de w_s, atunci *numărul de tranziții* este dat de numărul de biți care se schimbă. Acesta este:

$$d(w_r, w_s) = \sum_{j=1}^{k} w_{rj} \oplus w_{sj}, \tag{3.1}$$

unde k este lungimea cuvintelor și w_{rj} este al j-lea bit din reprezentarea lui w_r și $\oplus$ este operația *SAU-EXCLUSIV*.

Definiția 3.4. *Asignare.* Se numește asignare o funcție δ cu domeniul {1, 2, ..., n} și codomeniul {0,1}.

Definiția 3.5. *Numărul total de tranziții.* Se consideră secvențele w_1, w_2, ..., w_n, o permutare σ a lor și o asignare δ care specifică pentru fiecare cuvânt dacă se va considera inversat sau neinversat. *Numărul total de tranziții* se definește ca fiind:

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}^{\delta(j)}, w_{\sigma(j+1)}^{\delta(j+1)}) \tag{3.2}$$

unde d este distanța Hamming.

Definiția 3.6. *Problema Ordonării Datelor(DOP*) se poate formula ca fiind căutarea unei permutări σ a șirului de cuvinte $w_1, w_2, ..., w_n$, astfel încât numărul total de tranziții

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}, w_{\sigma(j+1)}) \tag{3.3}$$

să fie minimizat.

Definiția 3.7. *Problema ordonării datelor cu inversiune (DOPI)* se poate formula ca fiind găsirea unei permutări σ a șirului de cuvinte $w_1, w_2, ..., w_n$ și a unei asignări δ astfel încât numărul total de tranziții

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}^{\delta(j)}, w_{\sigma(j+1)}^{\delta(j+1)}) \tag{3.4}$$

să fie minimizat.

Problemele DOP și DOPI sunt NP-dificile. Vom nota generalizarea lor tot cu DOP și DOPI. Se consideră date o mulțime n de cuvinte de aceeași lungime k peste un alfabet $\mathcal{A}$. În cazul DOPI se dă și o permutare de lungime k ce reprezintă funcția de inversiune (complementare). Se caută o permutare a cuvintelor (în cazul DOPI și o asignare a acestora) care să minimizeze numărul total de tranziții.

3.2 Cercetare precedentă

În [Mur95] se demonstrează că forma decizională a problemei *DOP* este *NP*-completă şi sunt propuse o serie de scheme pentru găsirea unor soluţii acceptabile: arbore parţial dublu (*Double Spanning Tree – DST*), arbore parţial minim cu potrivire maximă (*Minimum Spanning Tree Maximum Matching – ST-MM*) şi *Greedy* (care din punct de vedere empiric a fost găsită drept cea mai bună). Atât în cazul DOP/DOPI, cât şi problema comisului voiajor – *Travelling Salesman Problem* (*TSP*)− se cere o reordonare a elementelor relativ la costul dintre două elemente alăturate. Deoarece *DOP* şi *DOPI* sunt *NP*-complete, algoritmii exacţi pot manipula doar instanţe ale problemei de dimensiune foarte mică. În anii precedenţi au fost prezentate o serie de euristici pentru *DOP* şi *DOPI* (cele mai multe în strânsă legătură cu TSP):

1. Arbore dublu de acoperiere – Double Spanning Tree (DST) [Gar79]
2. Arbore de acoperire/Potrivire Minimă – Spanning Tree/Minimum Matching (ST-MM) [Gar79]
3. Greedy Min (GM) [Mur97]
4. Greedy Simple (GS) [Dre97]
5. Euristici evolutive [Dre03]

Cel mai eficient algoritm polinomial cunoscut până acum este *Greedy Min*. Acesta poate fi aplicat atât pentru *DOP* cât şi pentru *DOPI*:

1. Se calculează distanţa Hamming pentru toate perechile de cuvinte şi selectează o pereche cu cost minim.
2. Se alege cea mai potrivită pereche de cuvinte – aceasta denotă secvenţa iniţială.
3. Se construieşte secvenţa progresiv, adăugând la fiecare pas cel mai potrivit cuvânt care nu a fost încă adăugat. Acest cuvânt poate fi adăugat fie la începutul, fie la sfârşitul secvenţei, depinzând de poziţia unde distanţa obţinută este minimală.

Cum am precizat şi mai sus, algoritmii evolutivi sunt cei mai performaţi relativ la calitatea rezultatelor. Astfel de abordări evolutive conduc la rezultate mai bune decât Greedy Min, dar necesită mai multe resurse timp şi spaţiu. Algoritmi evolutivi pentru optimizări de calitate superioră sunt prezentaţi in [Dre97, Dre03]. În [Dre03] se găsesc algoritmi evolutivi pentru *DOP* şi *DOPI*. Pentru *DOPI* operatorii de mutaţie şi încrucişare sunt aplicaţi in paralel pentru a crea noi indivizi, iar agoritmul este unul hibrid deoarece indivizii iniţiali sunt generaţi cu ajutorul metodelor Greedy. Rezultatele furnizate de algoritmii evolutivi sunt mai bune decât cele furnizate de Greedy Min, iar numărul maxim de cuvinte este 100. În [Mur98] se introduce un model din teoria grafurilor, împreună cu noţiunile de bază necesare. Pentru o instanţă a problemei cu n cuvinte, fiecare de lungime k, se construieşte corespunzător un multigraf. Vârfurile sunt cuvintele, iar muchiile sunt etichetate cu distanţa dintre cuvinte. Dacă mesajele au aceeşi asignare, atunci distanţa este aceeaşi. La fel se comportă şi când au asignări diferite. Vor fi deci maximum două muchii între două vârfuri: una etichetată cu distanţa

pentru cazul acceeaşi asignare şi una etichetată cu distanţa pentru cazul asignărilor complementare. În această manieră, DOPI se transformă în problemă echivalentă NP-completă de găsire a drumului hamiltonian minimal. Aşa cum se arată şi în [Mur98], un algoritm de mărginire inferioară este oferit de costul arborelui minimal de acoperire a acestui multigraf.

3.3 Algoritmii RAN, EX și LB pentru DOP și DOPI

RAN este cea mai simplă euristică pentru a genera soluții de start pentru astfel de probleme. Algoritmul *Random (RAN)* pentru *DOP* va genera o permutare aleatorie şi o va returna, împreună cu costul ataşat. Cel pentru *DOPI* va genera o permutare şi un şir de biți reprezentând asignarea [Log06a, Log07c].

Cel mai simplu algoritm optimal pentru DOP/DOPI se bazează pe căutarea exhaustivă, el parcurge toate permutările (și asignările) și reține o permutare (pereche) de cost minim. Acest algoritm se poate îmbunătăți folosind variații de programare dinamică, *branch and bound* sau programare liniară. Pentru a testa o serie de cazuri de bază, implementăm variante bazate pe *brute-force*. Algoritmul *Exact* pentru *DOP* generează toate permutările şi o reține pe prima care conduce la un cost minim. Cel pentru *DOPI* generează toate perechile posibile permutare-asignare şi o reţin pe prima cu costul minim [Log06a, Log07c].

Algoritmul de mărginire inferioară (engl. *lower-bound, LB*) foloseşte arborele parţial minim de acoperire şi se dovedeşte foarte util în evaluarea eficienței algoritmilor greedy şi evolutivi propuşi. Acesta este un algoritm de tip greedy.

3.4 Algoritmul Greedy Min Simplified (GMS)

Algoritmul *Greedy Min Simplified* (GMS) [Log06c, Log08d, Log09a] funcționează, ca și *Greedy Min* [Mur97], pentru ambele probleme, *DOP* şi *DOPI*. Diferența este că, la fiecare pas, adăugarea noului cuvânt (cu posibilitatea de a se seta şi inversa, în cazul *DOPI*) este permisă numai la un capăt al secvenței. Explicația este că se economiseşte timp prin testarea numai la un capăt şi performanțele în practică pentru seturi mari de date sunt foarte apropiate de *Greedy Min* și chiar mai bune decât acesta. Pentru operația de transmitere a cuvintelor (la testarea dinamică sau prin intermediul internetului, de exemplu), timpul este un factor important.

3.5 Algoritmii MUT și GA pentru DOP și DOPI

MUT_DOPI [Log06a] este o abordare simplă care operează pe un singur individ (inițial stabilit cu *Greedy Min*). Se propune un nou operator de mutație, *Simple Cycle Inversion Mutation* (SCIM)

[Log06a], care se aplică împreună cu *Simple Inversion Mutation* (SIM) [Hol75]. Acest algoritm conduce la îmbunătățiri ale soluției Greedy într-un timp de execuție relativ scurt.

GA_DOPI [Log06a] conduce la îmbunătățiri mai bune şi se bazează pe modelul clasic al algoritmilor genetici hibrizi. Operatorii genetici sunt aplicați sincronizat pentru permutare şi pentru asignare, cu scopul de a păstra subsecvențele performante. Asupra indivizilor din populația curentă se aplică operatorii de mutație şi de încrucişare cu o probabilitate dată, şi cei mai buni *populationSize* indivizi sunt păstrați în faza de selecție. Numărul total de tranziții dat de formula (3.2) este folosit ca funcție obiectiv ce măsoară calitatea unui individ:

$$eval(\sigma,\delta) = \sum_{j=1}^{n-1} d(w_{\sigma(j)}^{\delta(j)}, w_{\sigma(j+1)}^{\delta(j+1)}) \tag{3.5}$$

Operatorii de încrucişare utilizați de algoritmul evolutiv şi care conduc la rezultate calitative sunt operatorii clasici *Partially-Mapped Crossover* – PMX [Gol85], *Ordered Crossover* – OX [Dav85] şi doi operatori derivați propuşi de autoare: *Cycle PMX* (CPMX) [Log06a] şi *Cycle OX* (COX) [Log06a]. Ei sunt asemănători cu mutarea 2-opt, cu deosebirea că se inversează şi secvența dintre punctele de tăiere. Ca operatori de mutație folosim operatorii SIM şi SCIM. Setările parametrilor sunt alese pe baza experimentelor. Deoarece algoritmul genetic se aplică tipurilor diferite de date, de la foarte mici la extinse, devine necesar să adaptăm aceste setări la dimensiunea problemei.

3.6 Algoritmul paralel distribuit MRPGA pentru DOP şi DOPI

Se propune un algoritm distribuit paralel MRPGA_DOPI (*MapReduce Parallel Genetic Algorithm*) [Log10c] care foloseşte tehnologia MapReduce, modelul de programare paralelă introdus de Google şi prezentat în capitolul anterior. Folosim MapReduce pentru a aplica metoda ruletei în paralel pe diferite mulțimi de indivizi. Fiecare individ reprezintă o pereche permutare/asignare a secvențelor de intrare. Numărul de mulțimi de indivizi peste care se face selecția este dat de numărul total de indivizi din generație împărțit la un factor de reducere dat (*reductionFactor*). Acest număr va constitui numărul de *task*-uri *Reduce* ce vor fi pornite în paralel de către Hadoop. În funcția *Map*, pentru fiecare secvență se alege aleator mulțimea de indivizi unde se va face selecția. În funcția *Reduce*, cheia de intrare reprezintă numărul mulțimii selecție şi valorile de intrare sunt perechile permutare/asignare ce alcătuiesc mulțimea. Se aplică o selecție prin metoda ruletei şi se scriu la ieşire indivizii aparținând următoarei generații. După terminarea *job*-ului se adună indivizii noii generații din fişierele scrise de fiecare nod reducător şi urmează un nou pas de ordonare. Implementarea MRPGA_DOPI este o aplicație Hadoop scrisă în Java. Versiunea binară şi librăriile dependente sunt arhivate în fişierul *dopiSolver.jar*. *Driver*-ul ataşat arhivei înregistrează aplicația Hadoop cu numele **dopi**.

```
ALGORITMUL_ MRPGA_DOPI
    Initialize(populationSize)
    Initialize(crossoverRate)
    Initialize(mutationRate)
    Initialize(reductionFactor)
    Initialise_GreedyMin_individuals()
    Initialise_Greedy1_individuals()
    Initialise_Random_individuals()
    def MRPGA_MAP( initialIndex,pair<permutation, assignation>) = {
        reductionSetIndex = random(0.. reductonFactor-1)
        context.write(reductionSetIndex, pair<permutation, assignation>)
    }
    def MRPGA_REDUCE
        (reductionSetIndex,Iterable< pair<permutation, assignation>>)= {
        Apply_Crossover_operators(numCrossovers);
        Apply_Mutation_operators(numMutatios);
        Calculate_fitness(allNewIndividuals);
        Remove_WorstInviduals (numCrossovers+numMutations);
        for (pair<new_perm, new_assign> : allNewIndividuals)
            context.write
                (reductionSetIndex,pair<new_perm, new_assign>)
    }
    for( i ← 1;i ≤ numGenerations; step 1) execute
        job← NewJob(MRPGA_MAP, MRPGA_REDUCE)
        job.configuration.set(populationHDSFPath)
        job.configuration.set_crossoverRate)
        job.configuration.set_mutationRate)
        job.configuration.setNumReduceTasks(reductonFactor)
        job.submit_and_wait
        collect_new_population
    end_for
END_ALGORITHM_ MRPGA_DOPI
```

Fig. 3.1. Pseudocode pentru MRPGA_DOPI.

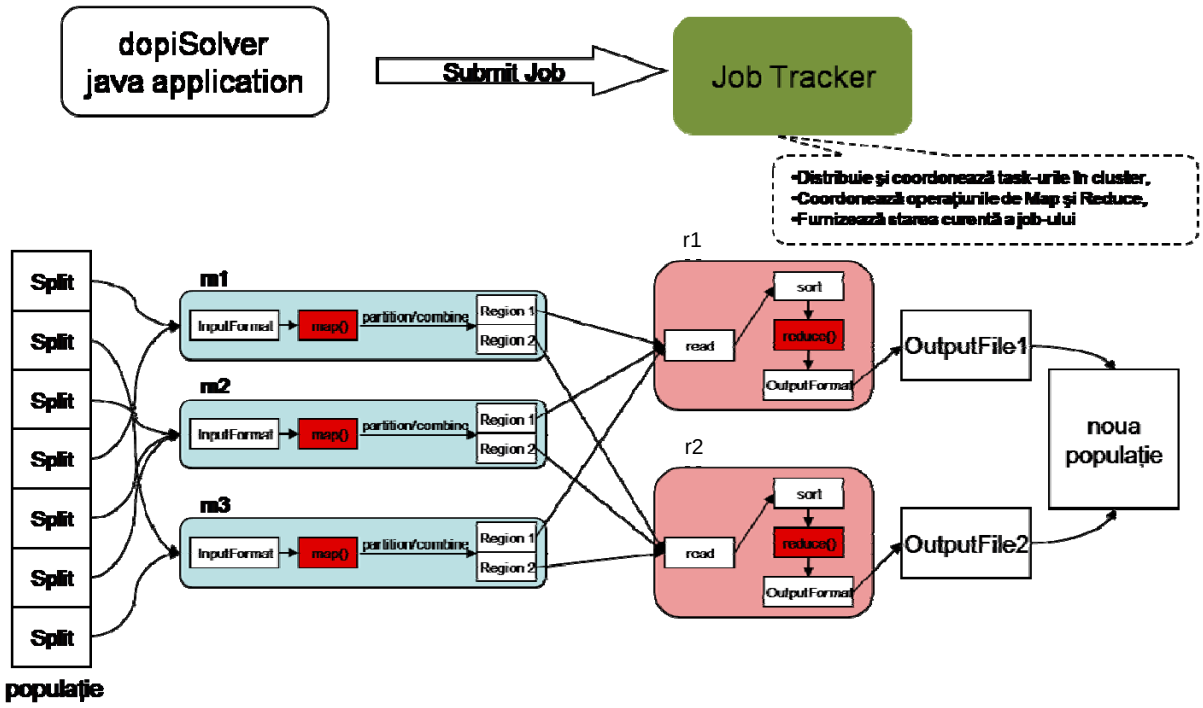


Fig. 3.2. Implementare MRPGA_DOPI cu Java şi Hadoop.

3.7 Experimente numerice şi teste statistice

Rezultatele experimetale sunt concentrate pe instanţe ale problemei DOPI, care furnizează rezultate mai bune decât DOP (datorită gradului sporit de libertate în alegerea asignării). Acest grad de libertate măreşte considerabil şi complexitatea problemei. Programul test iniţializează instanţe aleatoare cu distribuţie uniformă ale problemei, pentru o pereche dată (n, k): n cuvinte binare, fiecare de lungime k. Pentru aceste instanţe se aplică algoritmii prezentaţi mai sus.

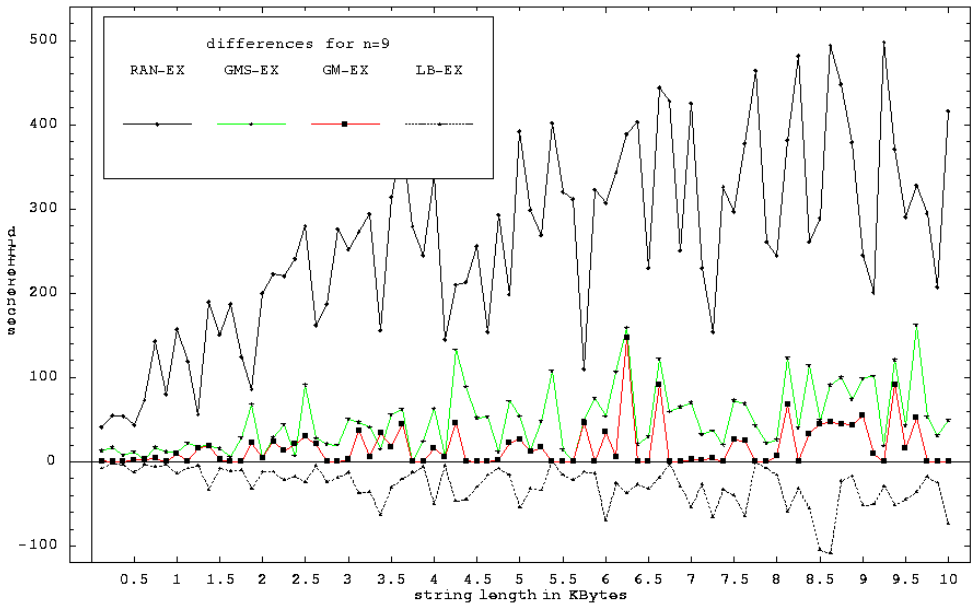
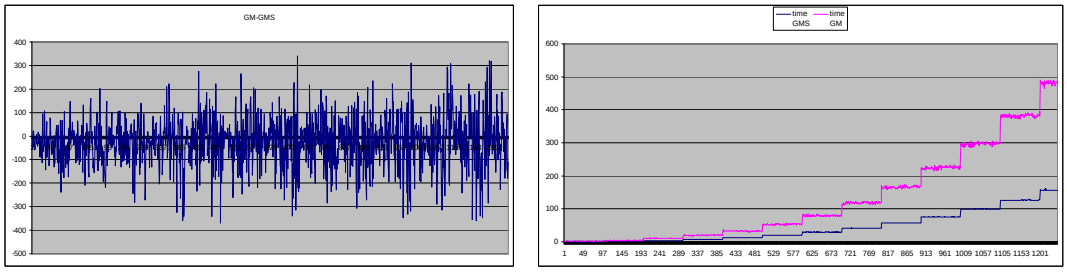


Fig. 3.3. Valorile RAN–EX, GMS–EX, GM–EX şi LB–EX pentru parametrii n=9, k=500, 1000, ..., 10000.



3.4. Valorile GM−GMS (stânga) şi timpii de execuţie (sec., dreapta) pentru 1000 de experimente, n = 1000, 1500..7000, k = 50, 100..1000. Valorile pozitive ale graficului reprezintă rezultate GMS mai eficiente. Se observă că timpul de execuţie GMS este considerabil mic decât cel GM.

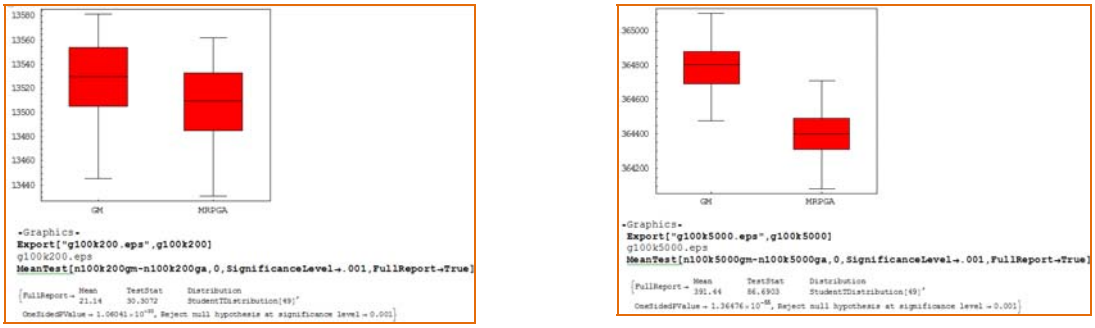


Fig. 3.5. Medianele valorilor GM şi MRPGA şi teste statistice student-t care certifică superioritatea metodei MRPGA pentru DOPI cu o probabilitate de 0.999. Parametri utilizaţi: 50 rulări, n=100, k=200 (stânga), 5000 (dreapta).

Teza conţine şi numeroase alte experimente şi teste statistice pentru a compara algoritmii propuşi în perechi, atât din punct de vedere al calităţii rezultatelor, cât şi al timpului de execuţie.

3.8 Concluzii şi direcţii de cercetare

Pe lângă alţi algoritmi, am prezentat în acest capitol două abordări Greedy şi trei abordări evolutive. Algoritmii random, exact şi de mărginire inferioară au ajutat la stabilirea calităţii rezultatelor. Una dintre abordările Greedy (GMS) este nouă şi ea se dovedeşte a fi foarte eficientă, atât din punct de vedere al calităţii rezulatatelor cât şi al timpului de execuţie, mai ales pentru dimensiuni mari ale datelor de intrare. Abordările evolutive sunt recomandate atunci când focusul este calitatea rezultatelor şi resursele spaţiu/timp de execuţie sunt irelevante. Una dintre direcţiile de studiu ar putea fi combinarea profitabilă a acestor tehnici şi testarea pe dimensiuni extinse ale datelor de intrare, variate şi din punct de vedere al distribuţiei elementelor atom. Algoritmul evolutiv MRPGA_DOPI este cel mai performant din punct de vedere al rezultatelor, şi datorită tehnologiei MapReduce, se poate aplica datelor de dimensiuni foarte mari, ceea ce constituie un real avantaj pentru aplicaţiile practice din domeniul industrial.

4

Problema Compactării Datelor

În acest capitol se propun o serie de algoritmi originali pentru rezolvarea problemei compactării datelor, *Test Compaction Problem* (TCP). Această problemă importantă este NP-dificilă. Ea se referă la activitatea de testare a circuitelor integrate şi joacă un rol important în proiectarea acestora. Algoritmii propuşi sunt doi de tip Greedy, *Greedy Naive Algorithm* (GRNV_TCP) şi *Greedy Binary Tree Algorithm* (GRBT_TCP) [Log08a, Log08e], un algoritm evolutiv eficient *Genetic Algorithm Test Compaction* (GA_TCP) [Log08b, Log09c] şi un algoritm evolutiv distribuit folosind noua tehnologie MapReduce, *MapReduce Parallel Evolutionary Algorithm* (MRPEA_TCP) [Log10b]. Deasemenea, se oferă şi un proiect ştiinţific *Open Source* aflat la adresa: http://dcpsolver.sourceforge.net/.

4.1 Descrierea problemei TCP

Definiţia 4.1. *Mulţimea de compactare (Compaction Set -CS).* Fiecare test este reprezentat ca un şir ce conţine caractere din mulţimea *CS* = {‘0’, ‘1’, ‘U’, ‘Z’, ‘X’}. ‘X’ este caracterul “Don’t Care”.

Definiţia 4.2. *Caractere compatibile şi operaţia de fuzionare (merge).* Două caractere c_1 şi c_2 sunt compatibile dacă ele coincid sau dacă unul dintre ele este ‘X’. Notăm cu ‘≅’ relaţia de compatibilitate şi negaţia ei cu ‘≇’.

Definiţia 4.3. *Teste compatibile şi operaţia de fuzionare.* Două teste date sunt compatibile dacă au aceeşi lungime şi toate caracterele de pe aceeaşi poziţie sunt compatibile.

Două teste compatibile pot fuziona. Testul lor *fuzionat* este obţinut prin substituirea fiecărui caracter cu caracterul lor fuzionat (*merge*). Vom nota această relaţie pentru teste tot cu ‘≅’ şi negaţia ei cu ‘≇’.

Definiția 4.4. *Mulțimea de acoperire (Coverage Set)*. Fie două mulțimi de teste $S_1 = \{t_{11}, t_{12}, .., t_{1i}\}$ şi $S_2 = \{t_{21}, t_{22}, \ldots, t_{2j}\}$. S_2 este un set de acoperire al lui S_1 dacă pentru fiecare test t în S_1 există unul compatibil în S_2 astfel încât pe pozițiile cu 0, 1, U şi Z în testul din S_1 să conțină exact acelaşi caracter şi în S_2.

Definiția 4.5. *Test Compaction Problem* (*TCP*). Fie $S_1 = \{t_{11}, t_{12}, \ldots, t_{1i}\}$ o mulțime de test unde pentru toți k=1, .., i, t_{ik} au aceeaşi lungime. Se caută o mulțime de acoperire $S_2 = \{t_{21}, t_{22}, \ldots, t_{2j}\}$ a lui S_1, astfel încât cardinalul lui S_2 să fie minimal (peste cardinalii mulțimilor de acoperire ale lui S_1).

Problema compactării secvețelor ADN este de fapt aceeaşi cu cea a testelor VLSI, cu diferența că apar alte simboluri. Vom numi generalizarea problemei *Data Compaction Problem* (DCP): se dau n secvențe de lungime k, fiecare conținând simboluri dintr-un alfabet dat $\mathcal{A}$, şi un simbol '*X*' cu semnificația „Don't Care". Se cere găsirea unei mulțimi compactate de secvențe, care să acopere mulțimea inițială.

4.2 Algoritmul optimal pentru TCP

Algoritmul optimal se bazează pe metoda backtracking recursiv. Operația de fuzionare este aplicată asupra perechilor compatibile de teste şi algoritmul se apelează recursiv asupra tuturor vectorilor astfel obținuți. Variabila booleană *compact* verifică dacă există perechi compatibile, în caz contrar rămâne setată pe *true*. Datorită complexității sale exponențiale, acest algoritm poate fi aplicat în practică doar pentru dimensiuni mici ale problemei.

4.3 Algoritmii Greedy GRNV și GRBT pentru TCP

Algoritmii GRNV şi GRBT au fost propuşi de către autoare în [Log08a, Log08e, Log09b]. Transformarea se face asupra mulțimii de teste şi se operează doar cu mulțimea procesată. La fiecare pas, primele două teste compatibile sunt substituite de către fuzionatul lor. Dacă astfel de teste nu mai sunt găsite atunci ne oprim şi se livrează mulțimea de teste obținută. Complexitatea este polinomială $O(n^3)$. Un factor important în cadrul execuției algoritmului Greedy este modul de a alege la fiecare pas cele două teste compatibile ce trebuie fuzionate. Aceasta poate să influențeze calitatea rezultatelor. Vom face o diagnoză folosind două posibilități: prima stochează testele într-o structură de tip tablou unidimensional, de fiecare dată primele două teste compatibile găsite în parcurgerea tabloului sunt înlocuite de fuzionatul lor. Numim acest algoritm GRNV – **Gr**eedy **N**ai**v**e. O altă posibilitate este utilizarea unui arbore binar de căutare în care comparatorul este dat de ordinea descrescătoare a numărului de simboluri X pe care le conțin testele. Numim acest algoritm GRBT - **Gr**eedy **B**inary Search **T**ree.

4.4 Algoritmul evolutiv GATC pentru TCP

Acest algoritm se bazează pe schema clasică a algoritmilor genetici, cu deosebirea că populația inițială conține clone ale secvenței de start. Mutațiile se aplică asupra populației curente şi cei mai buni indivizi sunt păstrați pentru iterația succesoare. După un număr de iterații aplicăm pentru fiecare individ algoritmul Greedy GRBT descris mai sus pentru a furniza mulțimi de acoperire compactate ale testelor. Particularitatea algoritmului este faza de inițializare cu clone ale secvenței, dar şi faza finală, care foloseşte metoda Greedy. O soluție potențială este o mulțime de acoperire pentru secvența de start, adică o mulțime care conține cel puțin un test compatibil pentru fiecare test din mulțimea de start. O populație este o mulțime de astfel de soluții potențiale. Pe aceşti indivizi se aplică succesiv operatorii de mutație, care înlocuiesc două teste compatibile aleatorii cu fuzionatul acestora. Ca şi funcție de selecție vom utiliza numărul total de simboluri *X*:

$$N_X(t_1,t_2,...,t_n)=\sum_{i=1}^{n}\#X(test_i) \qquad (4.\mathbf{1})$$

în care *#X*() este o funcție care returnează numărul de caractere 'X' din testul dat ca parametru.

4.5 Algoritmul evolutiv distribuit MRPEA pentru TCP

Implementarea algoritmului evolutiv paralel este o aplicație Hadoop (vezi capitolul 2) bazată pe paradigma MapReduce. Toate secvențele din fişierul de intrare au aceeaşi lungime (acelaşi număr de coloane). La început aplicația parcurge întreg fişierul de intrare şi calculează frecvența fiecărui simbol din alfabetul de intrare (inclusiv Don't Care) pentru fiecare coloană. Prin excluderea *coloanelor uniforme* (coloane ce conțin un anumit simbol cu o frecvență mai mare decât limita de uniformitate dată, de ex. 90%) se obține mulțimea *coloanelor permutabile*. În cadrul algoritmului se generează aleator un număr dat (*maxp*) de aranjamente ale coloanelor permutabile, luate câte un factor de permutare dat (*pf* – permutation factor). Pentru fiecare aranjament generat, aplicația rulează un *job* Hadoop ce crează o compactare a secvențelor inițiale. Cea mai scurtă compactare constituie rezultatul final. Pentru *job*-ul Hadoop aplicația furnizează câte o implementare pentru *InputReader*, *Map*, *Partitioner*, *Compare*, *Reduce*, *OutputWriter* [Dea08] [http://hadoop.apache.org]. *InputReader*-ul divide fişierul de intrare în secțiuni logice independente (*splits*) de lungime 64M. Fiecare *split* este asignat unui *nod Mapper* din cluster. Un *split* este o partiție logică a fişierului de intrare: calea fişierului de intrare din *HDFS (Hadoop Distributed File System)* împreună cu poziția de start şi *offset*-ul. *InputReader*-ul trebuie să furnizeze şi un *record reader* (cititor de înregistrări) care va fi folosit de *nodul Mapper* pentru parsarea split-urilor în perechi chei/valori, ce vor servi drept input pentru funcția *Map*. În cazul nostru, cheile sunt indicii secvențelor în fişierul de intrare şi valorile sunt secvențele. Funcția *Map* extrage din secvențele primite simbolurile din coloanele date de aranjamentul aleator

stocat in contextul *job*-ului, crează chei pe baza lor şi împreună cu secvenţele sunt trimise mai departe pentru reducere. Funcţia de partiţie (*Partitioner*) alege pe baza cheii un nod din cluster (o partiţie) care va prelucra mai departe (*Reduce*) rezultatele funcţiei *Map*: *key.hashCode()%NumarulNodurilorDeReducere*.
Pe nodurile de reducere, cu ajutorul funcţiei *Compare*, se grupează valorile având aceeaşi cheie. Funcţia de reducere (*Reduce*) iterează valorile având aceeaşi cheie şi generează perechi chei/valori rezultat, unde valorile rezultate sunt obţinute prin compactarea GATC_TCP a valorilor de intrare. Cheia de intrare este folosită mai departe în perechile rezultat. Perechile rezultat ale funcţiei de reducere sunt trimise *OutputWriter*-ului pentru a fi scrise în sistemul distribuit de fişiere, în unul sau mai multe fişiere create într-un director specificat în contextul job-ului (fiecare nod reducător scrie într-un fişier separat). La sfârşit *job*-ul citeşte toate secvenţele scrise de *OutputWriter* şi le compactează din nou cu algoritmul GRBT. Compactarea având cardinalitatea minimă este salvată în sistemul local de fişiere.

```
ALGORITHM_MRPEA_DCP
    initialize(currentSequenceSet)
    initialize(mutationRate)
    initialize(reductionFactor)
    for (i ← 1; i ≤ numGenerations; step 1)
        population ← GenerateRandomPermutations(currentSequenceSet)
        def MRPGA_TCP_MAP(seqIndex, sequence) =
            context.write(random(0.. reductionFactor−1), sequence)
        def MRPGA_TCP_REDUCE(subsetIndex, Iterable<sequence>)= {
            // run GATC on subset
            numMutations ← populationSize*mutationRate
            apply_Mutation_Operators(numMutatios);
            calculate_Fitness(allNewIndividuals);
            remove_Worst_Inviduals (populationSize/2);
            complete_With_Copy_Individuals(populationSize/2)
            context.write(subsetIndex, best_element(individuals))
        }
        job← NewJob(MRPGA_TCP_MAP, MRPGA_TCP_REDUCE)
        job.configuration.setNumReduceTasks (reductionFactor)
        job.submit_and_wait
        currentSequenceSet = concatenate_reducers_parts(job)
    endfor
    return currentSequenceSet
END_ ALGORITHM_MRPEA_DCP
```

4.6 Rezultate experimentale și teste statistice

Referitor la algoritmii propuși pentru TCP se oferă o serie de experimente variate, care compară eficiența și timpul de execuție pentru algoritmii propuși în perechi. De exemplu, în peste 600 de cazuri testate, cu un n între 5 şi 30 şi k între 20 şi 40 doar în 10 cazuri algoritmul OPT a condus la rezultate mai bune decât acest algoritm Greedy GRNV. Acest lucru atestă calitatea algoritmului GRNV.

Un alt experiment executat este unul de comparare a algoritmilor GRNV şi GRBT, $100 \le n \le 1100$, $25 \le k \le 1025$ şi factori diferiți de compactare (13, 25, 38, 50, 62, 75, 88).

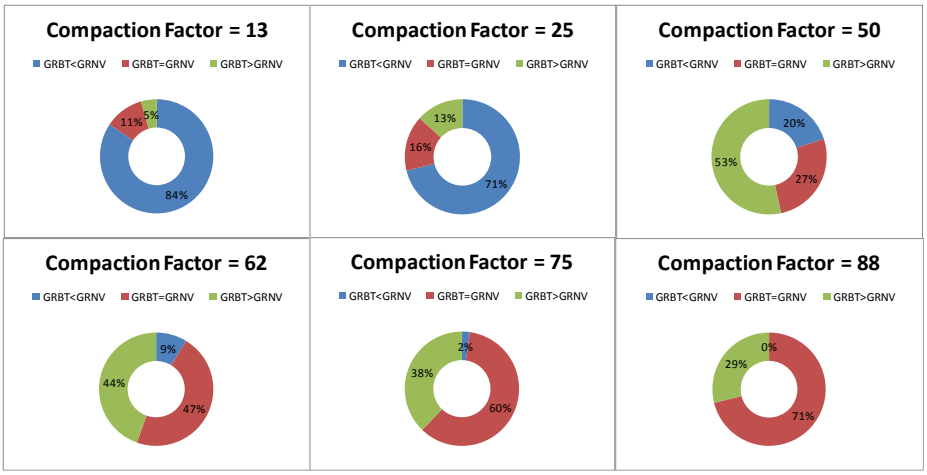


Fig. 4.1. Numărul de cazuri GRBT<GRNV, GRBT=GRNV şi GRBT>GRNV cu $100 \le n \le 1100$, $25 \le k \le 1025$, factor de compactare = 13, 25, 50, 62, 75 şi 88 – 1000 de rulări pentru fiecare factor de compactare.

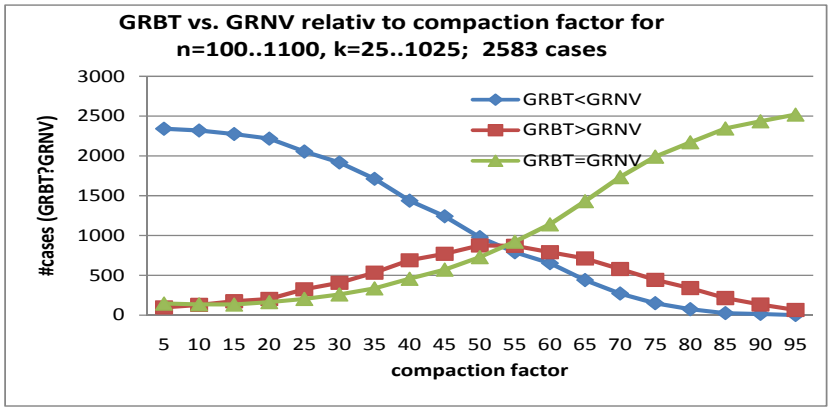


Fig. 4.2. Evoluția diferenței GRBT−GRNV în cadrul a 2583×19 experimente cu $100 \le n \le 1100$, $25 \le k \le 1025$, factor de compactare = 5, 10, 15... 95.

Cele 49077 experimente anterioare au fost global evaluate statistic: media diferenței (GRBT−GRNV) este −0.79, iar media timpilor de execuție (timp_GRBT−timp_GRNV) este −0.52, ceea ce atestă eficiența atât din punct de vedere al calității rezultatelor, cât şi timp de execuție, a metodei GRBT.

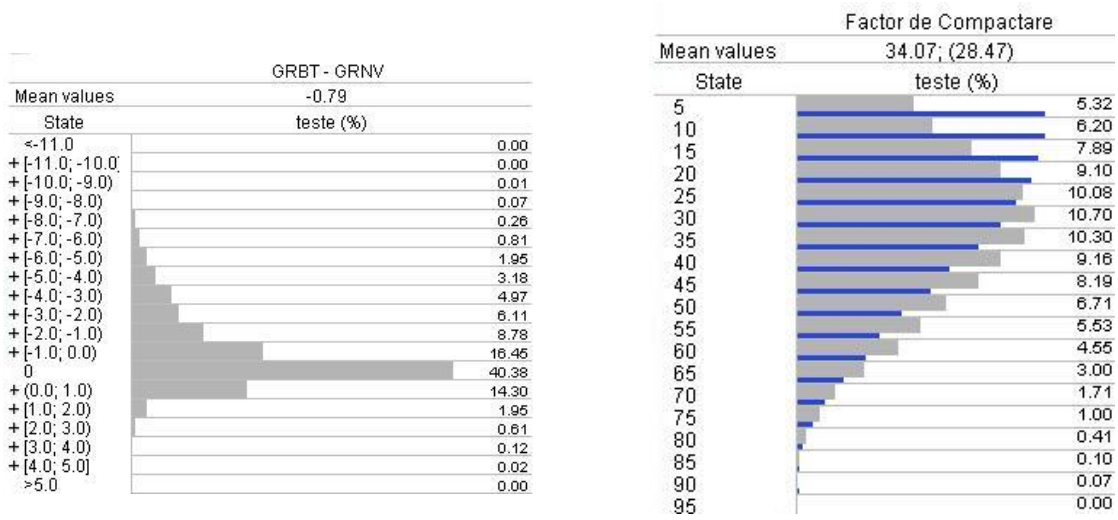


Fig. 4.3. Stânga: Distribuția pe intervale a diferenței GRBT−GRNV în cadrul a 49077 experimente cu $100 \le n \le 1100$, $25 \le k \le 1025$, factor de compactare = 5, 10, 15... 95. Dreapta: Distribuția diferenței de timp în cadrul celor 9915 teste pentru care GRBT<GRNV şi timpul GRBT este mai mic decât GRNV, referința sunt toate testele în care GRBT este mai bună decât GRNV.

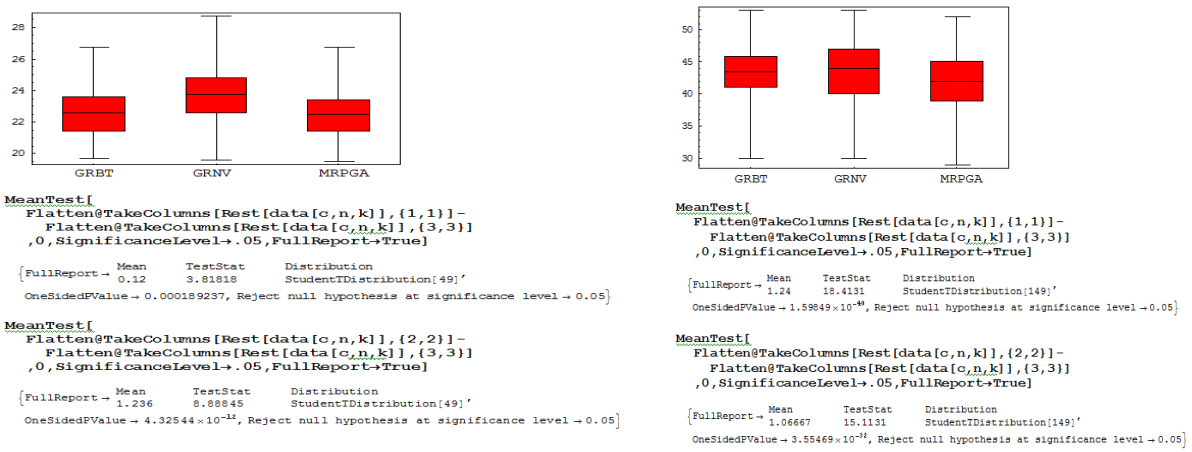


Fig. 4.4. Medianele şi teste statistice de tip student-t care atestă superioritatea metodei MRPEA pentru TCP cu o probabilitate de 0.95. Stânga: 50 rulări, *cf*=20, *n*=1000, *k*=2000. Dreapta: 150 rulări, *cf*=40, *n*=100, *k*=200.

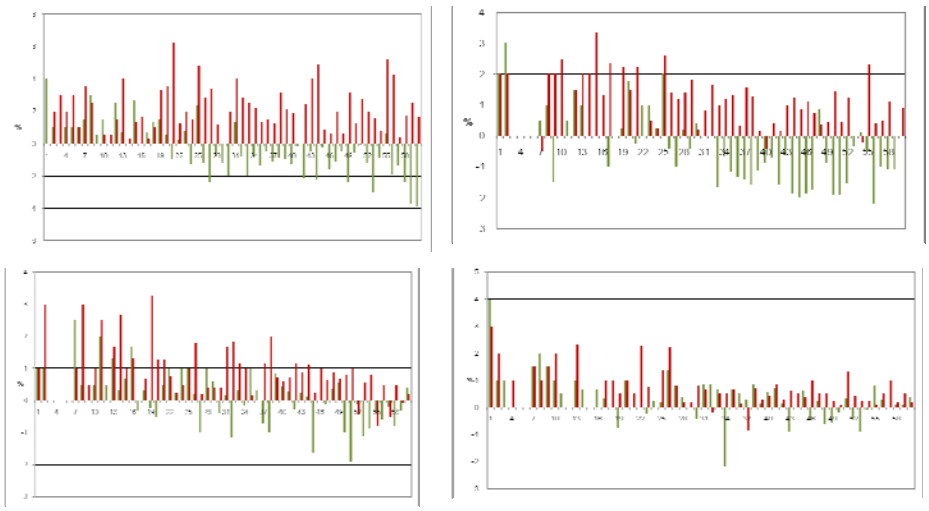


Fig. 4.5. Rezultate experimentale pentru seturi de secvențe cu n = 100 to 1000, pas 100, k = 50 to 550, pas 100, *pf*=3, (*a*) *cr*= 30, (*b*) *cr* = 40, (*c*) *cr* = 50, (*d*) *cr* = 60. Verde (GRBT−MRPEA), roşu (GRNV−MRPEA).

4.7 Concluzii și direcții de cercetare

După o descriere formală a problemei TCP, prezentăm o soluție optimală bazată pe metoda backtracking recursiv. Complexitatea este exponențială, de aceea algoritmul poate fi utilizat doar pentru seturi de date cu dimensiuni mici. Adițional, prezentăm două metode greedy şi una evolutivă, aplicate pe seturi reprezentative de date. Acestea arată calitatea foarte bună a rezultatelor furnizate de metodele prezentate, dar şi diferențele (calitate, timp de execuție) dintre ele relativ la diferiți parametri pentru dimensiunea datelor şi factorul de compactare. Noțiunea de factor de compactare este introdusă aici şi se dovedește a fi un element de bază în proiectarea şi alegerea algoritmului corespunzător unui set dat de date. În acest capitol sunt propuşi o serie de algoritmi originali pentru rezolvarea problemei compactării datelor, *Test Compaction Problem* (TCP). Această problemă importantă este NP-dificilă. Ea se referă la activitatea de testare a circuitelor integrate şi joacă un rol important în proiectarea acestora. Algoritmii propuşi sunt doi de tip Greedy, *Greedy Naive Algorithm* (GRNV_TCP) şi *Greedy Binary Tree Algorithm* (GRBT_TCP) [Log08a, Log08e], un algoritm evolutiv eficient *Genetic Algorithm Test Compaction* (GA_TCP) [Log08b, Log09c] şi un algoritm evolutiv distribuit folosind noua tehnologie MapReduce, *MapReduce Parallel Evolutionary Algorithm* (MRPEA_TCP) [Log10c]. Deasemenea, se oferă şi un proiect ştiințific *Open Source* aflat la adresa:

http://dcpsolver.sourceforge.net/.

Acesta conține implementările algoritmilor prezentați şi seturi de date pentru testare şi poate fi folosit în scopuri ştiințifice pentru extindere/modificare. Experimentele efectuate arată faptul că structura de date de timp arbore binar este mai eficientă pentru algoritmul greedy şi o rată de compactare mai mică de 50%. Comportarea algoritmilor variază o dată cu diferiți parametri: numărul de teste, lungimea unui test, rata de compactare. Calitatea rezultatelor furnizate de algoritmii greedy poate fi îmbunătățită prin adaptarea structurii de date la diferiți parametri sau atribute ale datelor de intrare. Îmbunătățiri pot aduce şi metodele de implementare; de exemplu, experimentele au arătat că pentru o reprezentare cu *std::string* din *STL*, timpul de execuție a fost mai bun decât una binară pentru anumite tipuri de date de intrare, de ex. dimensiune între 100 şi 500, lungimea testelor între 5000 şi 10000, factorul de compactare 20%. De asemenea, o analiză detaliată a factorului de compactare aşteptat poate duce la alegerea eficientă a algoritmului folosit, experimentele au condus la diferite calități ale rezultatelor furnizate de diverşi algoritmi pentru diferiți factori de compactare. O altă direcție ar fi clasificarea secvențelor cu mai mulți *Don't Care* pe aceleaşi poziții (sau alte criterii de clasificare) şi împărțirea datelor de intrare în seturi independente, rezolvarea acestora cu diferiți algoritmi şi combinarea rezultatelor. Algoritmii prezentați mai sus pot fi rafinați prin adaugarea unor tehnici mai sofisticate, de exemplu adaugarea de Tabu-lists sau îmbunătățirea arborelui de căutare cu ajutorul unui algoritm de tip A*. Stabilirea unui algoritm de mărginire inferioară eficient poate fi un factor important în testarea calității algoritmilor prezentați. Transformarea problemei într-o instanță a unei probleme cunoscute, de exemplu SAT, şi rezolvarea cu un *solver* specific poate aduce cu sine o serie de îmbunătățiri.

Bibliografie

[Adl94] Adleman, L. M.: Molecular Computation of Solutions to Combinatorial Problems, *Science*, vol. 266, pp. 1021-1024, 1994.

[Adl83] Adleman, L.M., Promerance, C., Rumely, R.S.: On distinguishing prime numbers from composite numbers, *Annals of Mathematics* 117 (1983), pp. 173-206, 1983.

[Agr04] Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in *P*, *Annals of Mathematics* 160 (2004), pp. 781-793, 2004.

[Alo06] Alon, N., Moshkovitz, D., Safra, M.: Algorithmic construction of sets for *k*-restrictions, *ACM Transactions on Algorithms (TALG)*, v. 2, n. 2, pp. 153-177, 2006.

[App03] Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problem, *INFORMS Journal of Computing* 15 (2003), pp. 82-92, 2003.

[App07] Applegate, D.L., Bixby, R., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, 2007.

[Aro98] Arora, S.: Polynomial Time Approximation Schemes for Euclidian Travelling Salesman and Other Geometric Problems, *Journal of the ACM*, Vol. 45, No. 5, pp. 753-782, 1998.

[Bak87] Baker, J.E.: Reducing bias and inefficiency in the selection algorithm, *Proceedings of the International Conference on Genetic Algorithms*, vol. 2, pp. 14-21, 1987.

[Boo87] Booker, L.: Improving Search in Genetic Algorithms, *Genetic algoritms and simulated annealing*, Davis, L. (ed.), Morgan Kaufmann Publishers, pp. 61-73, 1987.

[Bli95] Blicke, T., Thiele, L.: A Comparison of Selection Schemes used in Genetic Algorithms, 2. Ed., *TIK Report* No. 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zürich, 1995.

[Car06] Cardei, M., Wu, J.: Energy-efficient coverage problems in wireless ad-hoc sensor networks, *Computer Communications*, v. 29, n. 4, pp. 415-420, 2006.

[Cha92] Chandrakasan, A. P., Potkonjak, M., Rabaey, J., Brodersen, R. W.: HYPER-LP: a system for power minimization using architectural transformations, *Int'l Conf on CAD*, pp. 300-303, 1992.

[Chr76] Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem, *Technical Report* 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[Coo71] Cook, S. A., The complexity of theorem-proving procedures, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151-158, Shaker Heights, Ohio, USA, 1971.

[Cor01] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Introduction to Algorithms*, 3rd Edition, MIT Press, Boston, 2003.

[Dav85] Davis, L.: Applying adaptive algorithms to epistatic domains, *Proceedings of IJCAI*, pp. 162-164, 1985.

[Dav91] Davis, L.: *Handbook of Genetic Algorithms*, van Nostrand Reinhold, New York, 1991.

[Dav60] Davis, M., Putnam, H., A Computing Procedure for Quantification Theory, *Journal of the ACM 7* (1)*,* pp. 201-215, 1960.

[Dav62] Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving, *Communications of ACM* 5(7), pp. 394-397, 1962.

[Dea08] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters, *Commun. ACM, 51*(1), pp. 107-113, 2008.

[Dev95] Devadas, S., Malik, S.: A survey of optimization techniques targeting low power VLSI circuits, *Design Automation Conf.*, pp. 242-247, 1995.

[Dor96] Dorigo, M., Gambardella, L.M.: Ant Colonies for the Traveling Salesman Problem, Universitet Libre de Bruxelles, Belgium, 1996.

[Dre02] Drechsler, R., Drechsler, N.: *Evolutionary Algorithms for Embedded System Design*, Kluwer Acadmeic Publisher, 2002.

[Dre03] Drechsler, R., Drechsler, N.: Minimization of Transitions by Complementation and Resequencing using Evolutionary Algorithms, In *Proceedings of 21st IASTED International Multi-Conference Applied Informatics (AI 2003)*, IASTED International Conference on Artificial Intelligence and Applications (AIA 2003), Innsbruck, 2003.

[Dre97] Drechsler, R., Göckel, N.: A genetic algorithm for data sequencing. *Electronic Letters*, 33(10), pp. 843-845, 1997.

[Dre98] Drechsler, R.: *Evolutionary Algorithms for VLSI CAD*, Kluwer Academis Publisher, 1998.

[Dre99] Drechsler, N., Drechsler, R.: Exploiting don't cares during data sequencing using genetic algorithms. In *ASP Design Automation Conf.*, pp. 303-306, 1999.

[Dum00] Dumitrescu, D., Lazzerrini, B., Jain, L.C.: *Evolutionary Computation*, CRC Press, 2000.

[Dum06] Dumitrescu, D.: *Algoritmi genetici şi strategii evolutive - aplicaţii în Inteligenţa Artificială şi în domenii conexe*, reeditare, Editura Albastră, 2006.

[Edm65] Edmonds, J.: Minimum partition of a matroid into independent subsets, *Journal of Research of the National Bureau of Standards* B 69 (1965), pp. 67-72, 1965.

[Eka08] Ekanayake, J., Pallickara, S., Fox, G.: Mapreduce for data intensive scientific analyses, *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, IEEE Computer Society, pp. 277-284, Washington, DC, USA, 2008.

[Fei98] Feige, U.: A Threshold of Ln N for Approximating Set Cover, *Journal of the ACM (JACM)*, v. 45, n. 4, pp. 634-652, 1998.

[Fed95] Fredman, M. L., Johnson, D. S., McGeoch, L. A., Ostheimer, G.: Data Structures for Travelling Salesmen, *Journal of Algorithms 18*, pp. 432-479, 1995.

[Fog66] Fogel, L.J., Owens, A.J., Walsh, M.J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

[Gar79] Garey, M. R., Johnson, D. S.: *Computers and Intractability – A Guide to NP-Completeness*, Freeman, San Francisco, 1979.

[Gar97] Garzon, M., Deaton, R., Neathery, P., Murphy, R. C., Stevens Jr., S. E., Franceschetti, D. R.: A new metric for DNA computing, *Genetic Programming 1997*, *Proceedings of the Second Annual Conference*, Stanford University, pp. 479-490, AAAI, 1997.

[Gen00] Gen, M., Cheng, R.: *Genetic Algorithms and Engineering Optimization*, John Wiley Sons Inc., USA, 2000.

[Ghe03] Ghemawat, S., Gobioff, H., Leung, S., -T: The google file system, *SIGOPS Oper. Syst. Rev.*, 37(5), pp. 29-43, 2003.

[Gol85] Goldberg, D. E., Lingle, R.: Alleles, loci, and the travelling salesman problem, *Int'l Conference on Genetic Algorithms*, pp. 154-159, 1985.

[Guo01] Guo, R., Pomeranz, I., Reddy, S. M., On improving static test compaction for sequential circuits, *VLSI Design*, Fourteenth International Conference, pp. 111-116, 2001.

[Hel70] Held, M., Karp, R. M.: The traveling-salesman problem and minimum spanning trees, *Operations Research* 18, pp. 1138-1162, 1970.

[Hel71] Held, M., Karp, R.M.: The travelling-salesman problem and minimum spanning trees, part II. *Mathematical Programming* 1, pp. 6-25, 1971.

[Hel98] Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic/Roskilde University, *Writings on Computer Science*, 1998.

[Hel00] Helsgaun, K.: An effective implementation of the Lin-Kernighan Traveling Salesman Heuristic, *European Journal of Operation Research* 126, pp. 106-130, 2000.

[Hel06] Helsgaun, K.: An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic / Roskilde University, *Writings on Computer Science*, 2006.

[Hig06] Higami, Y., Kajihara, S., Pomeranz, I., Kobayashi, S., Takamatsu, Y.: On Finding Don't Cares in Test Sequences for Sequential Circuits, *IEICE Transactions on Information and Systems*, v. E89, n. 11, 2006.

[Hoc98] Hochbaum, D., S., Pathria, A.: Analysis of the greedy approach in problems of maximum *k*-coverage, *Naval Research Logistics*, v. 45, n. 6, pp. 615-627, 1998.

[Han95] Hansen, N., Ostermeier, A., Gawelczyk, A.: On the Adaptation of Arbitrary Mutation Distributions in Evolution Strategies: The Generating Set Adaptation*, Proceedings of*

the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 57-64, 1995.

[Hol75] Holland, J. H.: *Adaption in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

[Hur04] Hurkens, C.A.J., Woeginger, G.J.: On the nearest neighbour rule for the traveling salesman problem, *Operations Research Letters* 32 (2004), pp. 1-4, 2004.

[Ima94] Iman, S., Pedram, M.: Multilevel network optimization for low power, *Int'l Conf. On CAD*, pp. 372-377, 1994.

[Jin08] Jin, C., Vecchiola, C., Buyya, R.: MRPGA: An extension of mapreduce for parallelizing genetic algorithms, eScience '08, Proceedings of the 2008 Fourth IEEE International Conference on eScience, pp. 214-221, 2008.

[Joh95] Johnson, D. S., McGeoch, L. A.: The Traveling Salesman Problem: A Case Study in Local Optimization, 1995.

[Joh96] Johnson, D. S., McGeoch, L. A., Rothberg, E. E.: Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound, *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms* (1996), pp. 341-350, 1996.

[Kar72] Karp, M. R.: Reductibility Among Combinatorial Problems, *Complexity of Computer Computations* (Symposium Proceedings), Plenum Press, 1972.

[Knu97] Donald E. Knuth, *The Art of Computer Programming, Volume I: Fundamental Algorithms*, Addison-Wesley Longman, Amsterdam; Ed. 3, 1997.

[Kre99] Donald L. Kreher, Douglas R. Stinson, *Combinatorial algorithms. Generation, Enumeration, and Search*, CRC Press, 1999.

[Lad75] Ladner, R.E.: On the structure of polynomial time reducibility, *Journal of the ACM* 22 (1975), pp. 155-171, 1975.

[Lan60] Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems, *Econometrica* 28 (1960), pp. 497-520, 1960.

[Lin93] Lin, S., Kernighan, B. W.: An Effective Heuristic Algorithm for the Travelling-Salesman Problem, In Operations Research 21, pp. 498-516, 1973.

[Lit63] Little, J.D.C., Murty, K.G., Sweeny, D.W., Karel, C.: An algorithm for the traveling salesman problem, *Operations Research* 11 (1963), pp. 972-989, 1963.

[Llo10] Llora, X, Verma, A., Campbell, R. H., Goldberg, D., E.: When Huge is Routine: Scaling Genetic Algorithms and Estimation on Distribution Algorithms via Data-Intensive Computing, *Parallel and Distributed Computational Intelligence*, pp. 11-41, Springer, 2010.

[Log05] **Logofătu, D.**: Programare dinamică: sume de puteri , *GInfo* 15/2, pp. 40-43, 2005.

[Log06a] **Logofătu, D.**, Drechsler, R..: Efficient Evolutionary Approaches for the Data Ordering Problem with Inversion, *3rd European Workshop on Hardware Optimization Techniques (EvoHOT)*, LNCS 3907, pp. 320-331, Springer, Berlin/Heidelberg, 2006.

[Log06b] **Logofătu, D.**: *Algorithmen und Problemlösungen mit C++*, pp. 388-397, Vieweg-Verlag, 2006.

[Log06c] **Logofătu, D.**: Greedy Approaches for the Data Ordering Problem with Inversion, *Proceedings of ROSYCS* - Romanian Symposium on Computer Science, pp. 65-80, Iaşi, 2006.

[Log06d] **Logofătu, D.**: Problema ordonării datelor cu şi fără inversiune, GInfo 16/2, pp. 8-14, 2006.

[Log07a] **Logofătu, D.**: *Algoritmi fundamentali in C++. Aplicații*, pp. 127-154:265-273, Editura Polirom, Iaşi, 2007.

[Log07b] **Logofătu, D.**: *Algoritmi fundamentali in Java. Aplicații*, pp. 125-158:269-277, Editura Polirom, Iaşi, 2007.

[Log07c] **Logofătu, D.**: *Grundlegende Algorithmen mit Java*, pp. 65-98:205-214, Vieweg-Verlag, Germany, 2008.

[Log08a] **Logofătu, D.**, Drechsler, R.: Comparative Study by Solving the Test Compaction Problem, *Proceedings 38th International Symposium on Multiple-Valued Logic (ISMVL '08)*, Dallas, USA, pp. 44-49, 2008.

[Log08b] **Logofătu, D.**: Efficient Evolutionary Approach for the Test Compaction Problem, *Proceedings 9th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS*, pp. 144-148, Suceava, Romania, May 22-24, 2008.

[Log08c] **Logofătu, D.**: *Eine praktische Einführung in C*, pp. 207-208, entwickler.press, München, Germania, 2008.

[Log08d] **Logofătu, D.**, Gruber, M.: Efficient Approaches for DNA Sequences Ordering, *Proceedings Bio-Inspired Computational Methods Used for Difficult Problems Solving. Development of Intelligent and Complex Systems* (BICS 2008), pp. 59-69, Târgu Mureş, România, 2008.

[Log08e] **Logofătu, D.**: On the compaction of DNA Sequence Vectors, *Proceedings Bio-Inspired Computational Methods Used for Difficult Problems Solving. Development of Intelligent and Complex Systems* (BICS 2008), pp. 25-36, Târgu Mureş, România, 2008.

[Log09a] **Logofătu, D.**, Gruber, M.: DNA Sequences And Their Ordering, *American Institute of Physics*, Vol. 1117, pp. 3-11, 2009.

[Log09b] **Logofătu, D.**: DNA Sequences And Their Compaction, *American Institute of Physics*,

vol. 1117, pp. 29-39, 2009.

[Log09c] **Logofătu, D.**: Static Test Compaction for VLSI Tests: an Evolutionary Approach, *Advances in Electrical and Computer Engineering*, pp. 49-53, 2009.

[Log10a] **Logofătu, D.**: *Algorithmen und Problemlösungen mit C++*, pp. 402-411, Vieweg+Teubner-Verlag, 2010.

[Log10b] **Logofătu, D.**, Dumitrescu, D.: Parallel Evolutionary Approach of Compaction Problem Using MapReduce, 11th International Conference on Parallel Problem Solving from Nature, Cracovia, Polonia, 2010. (submited)

[Log10c] **Logofătu, D.**, Dumitrescu, D.: Distributed Genetic Algorithm for Data Ordering Problem with Inversion Using MapReduce, 11th International Conference on Parallel Problem Solving from Nature, Cracovia, Polonia, 2010. (submited)

[Lou03] Lourenço, H. R., Martin, O. C., Stützle, T.: Iterated Local Search, In Glover, F. (Ed.), Kochenberger, G. (Ed.): *Handbook of Methaeuristics*, Kluwer Academic Publishers, pp. 321-353, 2003.

[Lun94] Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems, *Journal of the ACM (JACM)*, v. 41 n. 5, pp. 960-981, 1994.

[Mat10] Matthews, S. J., Williams, T. L.: MrsRF: an efficient MapReduce algorithm for analyzing large collections of evolutionary trees, *BMC Bioinformatics*, 11(1), doi: 10.1186/1471-2105-11-S1-S15, 2010.

[Mal03] El-Maleh, A., Osais, Y.: Test vector decomposition based static compaction algorithms for combinatorial circuits, *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, pp. 430-459, 2003.

[Maz98] Mazumder, P., Rudnick, E.: *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Prentice Hall, 1998.

[Mic94] De Micheli, G.: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[Mic96] Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, Ed. 3rd, Springer-Verlag, Berlin Heidelberg New York, 1996.

[Mic07] Michiels, W., Aarts, E., Korst, J.: *Theoretical Aspects of Local Search*, Springer, Berlin, 2007.

[Mil05] Milenkovic, O., Kashyap, N.: DNA Codes that Avoid Secondary Structures, *Proceedings of IEEE International Symposium on Information Theory*, Adelaide, Australia, 2005.

[Mil06] Milenkovic, O., Kashyap, N.: On the Design of Codes for DNA Computing, *Coding and Cryptography (Lecture Notes in Computer Science 3969)*, pp. 100-119, Springer Verlag, Berlin-Heidelberg, Germany, 2006.

[Mil04] Miltersen, P. B., MILP, ILP and TSP, Course Notes for Search and Optimization, http://www.daimi.au.dk/dSoegOpt/ilp.pdf, Spring, 2004.

[Mur07] Murray, A., T., Kim, K., K., Davis, J., W., Machiraju, R., Parent, R.: Coverage optimization to support security monitoring, *Computers*, *Environment and Urban Systems*, vol. 31, n. 2, pp 133-147, 2007.

[Mur97] Murgai, R., Fujita, M., Krishnan, S. C.: Data sequencing for minimum-transition transmission, *IFIP Int'l Conf. on VLSI,* 1997.

[Mur98] Murgai, R., Fujita, M., Oliveira, A.: Using complementation and resequencing to minimize transitions, *Design Automation Conf.,* pp. 694-697, 1998.

[Mur94] T. Murata, H. Ishibuchi, Performance evaluation of genetic algorithms for flow shop scheduling problems, *International Conference on Evolutionary Computation*, pp. 812–817, 1994.

[Müh94] Mühlenbein, H.: The Breeder Genetic Algorithm - a provable optimal search algorithm and its application, *Colloquium on Applications of Genetic Algorithms*, IEEE 94/067, London, 1994.

[Müh93] Mühlenbein, H., Schilerkamp-Voosen, D.: Predictive Models for the Breeder Genetic Algorithm, *Continuous Parameter Optimization. Evolutionary Computation*, 1 (1), pp. 25-49, 1993.

[Oli87] Oliver, I. M., Smith, D. J., Holland, J. R. C.: A study of permutation crossover operators on the traveling salesman problem, *Int'l Conference on Genetic Algorithms*, pp. 224-230, 1987.

[Ost93] Ostermeier, A., Gawelczyk, A., Hansen, N.: A Derandomized Approach to Self Adaptation of Evolution Strategies, *Technical Report* TR-93-003, TU Berlin, 1993.

[Pap82] Papadimitriou, C.H., Steiglitz, K.: *Combinatorial Optimization; Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, cap. 17-19, 1998.

[Pap06] Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem, *Combinatorica* 26 (2006), pp. 101-120, 2006.

[Pra75] Pratt, V.: Every prime has a succinct certificate, *SIAM Journal on Computing* 4 (1975), pp. 214-220, 1975.

[Rag07] Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating mapreduce for multi-core and multiprocessor systems, *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007.

[Rec73] Rechenberg, I.: *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Fromman-Holzboog, Stuttgart, 1973.

[Rec73] Schwefel, H.-P.: *Numerical optimization of computer models*, Wiley&Sons, Chichester, 1981.

[Ros77] Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M.: An analysis of several heuristics for the traveling salesman problem, *SIAM Journal on Computing* 6 (1977), pp. 563-581, 1977.

[Rud96] Rudolph, G.: Convergence of Evolutionary Algorithms in General Search Spaces, *International Conference on Evolutionary Computation*, pp. 50-54, 1996.

[Sas07] Sastry, K., Goldberg, D., E., Llora, X.: Towards billion-bit optimization via a parallel estimation of distribution algorithm, *Proceedings of 9th annual conference on Genetic and evolutionary computation GECCO '07*, pp. 577-584, ACM, New York, 2007.

[Sel96] Selman, B., Kautz, H., Cohen, B.: Local Search Strategies for Satisfiability Testing, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* vol. 26, AMS, 1996.

[Sel93] Selman, B., Kautz, A.: Domain-Independent Extension to GSAT: Solving Large Structured Satisfiability Problems, In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI),* pp. 290-295, 1993.

[She95] Shen, W.-Z., Lin, J.-Y., Wang, F.-W.: Transistor reordering rules for power reduction in CMOS gates, *ASP Design Automation Conf.*, pp. 1-6, 1995.

[Shm90] Shmoys, D.B., Williamson, D.P.: Analyzing the Held-Karp TSP bound: a monotonicity property with application, *Information Processing Letters* 35 (1990), pp. 281-285, 1990.

[Sle95] Sleator, D. D., Tarjan, R. E.: Self-adjusting Binary Search Trees, *Journal of the Association for Computing Machinery*, Vol. 32, pp. 652-686, 1985.

[Sta94] Stan, M., Burleson, W.: Limited-weight codes for low-power I/O, *Int'l Workshop on Low Power Design,* 1994.

[Sys90] G. Syswerda, Schedule Optimization Using Genetic Algorithms, L. Davis, Editor, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1990.

[Tiw94] Tiwari, V., Malik, S., Wolfe, A., Lee, M.: Power analysis of embedded software: A first step towards software power minimization, *Int'l Conf. on CAD*, pp. 384-390, 1994.

[Tiw96] Tiwari, V., Malik, S., Wolfe, A., Lee, M.: Instruction level power analysis and optimization software, *VLSI Design Conf.*, 1996.

[Tom81] Ioan Tomescu, *Probleme de combinatorică şi teoria grafurilor,* Editura Didactică şi Pedagogică, Bucureşti, 1981.

[Tsu93] Tsui, C., Pedram, M., Despain, A. M.: Technology decomposition and mapping targeting low power dissipation, *Design Automation Conf.*, pp. 68-73, 1993.

[Tur36] Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the Londom Mathematical Society* (2) 42 (1936), pp. 230-265 şi 43 (1937), pp. 544-546, 1936-1937.

[Vai93] Vaishnav, H., Pedram, M.: PCUBE: A performance driven placement algorithm for low power design, *European Design Automation Conf.*, pp. 72-77, 1993.

[Whi89] Whitley, D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesman: The genetic edge recombination operator, *Int'l Conference on Genetic Algorithms*, pp. 133-140, 1989.

[Wol80] Wolsey, L.A.: Heuristic analysis, linear programming and branch and bound, *Mathematical Programming Study* 13, pp. 121-134, 1980.

Resurse Web:

http://scipy.org/scipy/scikits/wiki/MILP

[Mixed-Integer Linear Problems - MILP]

http://javailp.sourceforge.net/

[Java ILP – Java Interface to ILP solvers]

http://dopisolver.sourceforge.net/

[dopiSolver – *framework Open Source* pentru rezolvarea DOPI]

http://dcpsolver.sourceforge.net/

[dcpSolver – *framework Open Source* pentru rezolvarea DCP]

http://hadoop.apache.org/

[Apache Hadoop]

http://lsiwww.epfl.ch/LSI2001/teaching/webcourse/ch02/ch02.html#2.5

ftp://ftp.cs.cmu.edu/user/sleator/splaying/

[implementări C şi Java pentru *splay trees*]