Universitatea Babeş-Bolyai

Cristina Mihăilă

Teză de doctorat

Calculul evolutiv în probleme de alocare

Coordonator
D. Dumitrescu

Cluj-Napoca, 2011

Abstract

Conceptul de “alocare de sarcini şi resurse” nu este unul nou: acum 5000 de ani, Sun Tzu a scris despre alocare şi strategii din persepctivă militară, piramidele sunt vechi de peste 3000 de ani iar căile ferate transcontinentale se construiesc de vreo 200 de ani în coace. Nici una dintre aceste activităţi nu s-ar fi putut realiza fără o formă de alocare, adică fără înţelegerea sarcinilor şi a secvenţialităţii acestora [Weaver2006].

Astăzi, alocarea de sarcini şi resurse este o formă de luare de decizii care joacă un rol important în multe domenii: de la decizii ce ţin de viaţa personală (stabilirea unei agende zilnice sau planificarea itinerariului de dezvoltare personală) până la decizii la nivel guvernamental (stabilirea unei strategii de creştere economică, sau de asigurare a calităţii în educaţie), de la decizii de ordin cultural (organizarea unei expoziţii sau producerea unui film) până la decizii de ordin economic (introducerea unui nou produs pe piaţă sau relocalizarea unei fabrici).

Alocarea de sarcini şi resurse este acum studiată de către cercetători în domeniul managementului, al ingineriei industriale, al cercetării operaţionale şi în domeniul informaticii. Dată fiind importanţa şi complexitatea problemelor de programare, lucrarea de faţă investighează dacă paradigma calculului evolutiv, în esenţă de tip euristic, reprezintă un bun candidat pentru a rezolva mai bine şi/sau mai repede anumite instanţe ale diferitelor probleme de alocare de sarcini şi resurse.

Cuvinte cheie: alocare de sarcini şi resurse, algoritmi evolutivi

Lista publicațiilor

Dumitrescu, D., Iantovics, B., **Florea, C.**, Multi-Agent Systems: a new allocation protocol and evolutionary search for equilibrium; , in *Proceedings of Symposium "Zilele academice clujene" - Computer Science Section*, 119-133, Cluj-Napoca, Romania, 2002.

Dumitrescu, D., **Florea, C.**, Patranjan; P., Evolutionary Reorganization in MAS; , in *Proceedings of the European Conference on Information Technology* (ECIT02), 1-5, Iasi, Romania, 2002.

Dumitrescu, D., **Florea, C.**, Patranjan; P., A New Evolutionary Model for Multi Agent Systems, in *Proceedings of* the *4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC02), 137-143, Timisoara, Romania, 2002.

Groşan, C., Oltean, M., **Florea, C.**, NP-complete problems using Evolutionary Algorithms, *Lucrarile Seminarului de Didactica Matematicii al Universitatii Babes-Bolyai*, Vadu-Crisului, Romania, 2003.

Florea, C., Dumitrescu, D., Negotiation in Multiagent Systems; in *Proceedings of Conference on Applied and Industrial Mathematics* (CAIM03), Oradea, Romania, 2003.

Mihaila, C., Dumitrescu, D., Quantum Computing and Multiagent Systems, in *Proceedings of the Symposium "Colocviul Academic Clujean de Informatica",* Cluj-Napoca, Romania, 2005.

Mihiş, A., Crețu, C., **Mihăilă, C.**, Şerban, C., Code Simplification using Boolean Functions Simplification, in *Proceedings of the International Conference of Mathematics & Informatics*, Supplement of "Studii şi cercetări ştiinţifice. Seria Matematică", no.16, University of Bacau, 493-502, Bacău, România, 2006.

Nițchi, I.Ş., Mihăilă, A., **Mihăilă, C.**, About Project Management Planning Optimization using Genetic Algorithms, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 79-82 ,Cluj-Napoca, România, 2007.

Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A., **Mihăilă, C.**, About the Logical Model for Intelligent Agents, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 83-90, Cluj-Napoca, România, 2007.

Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A**.**, **Mihăilă, C.**, On the collaborative systems for e-business, in *Proceedings of the International Conference on Competitiveness and European Integration*, 266-272, Cluj-Napoca, România, 2007.

Mihăilă C., Cobârzan C., Evolutionary approach for multimedia caching, in *IEEE Proceedings of the Evolutionary Techniques in Data Processing Workshop,* International Conference on Database and Expert Systems Application (DEXA), 531-536, Torino, Italia, 2008

Mihăilă C., Mihăilă A., An Evolutionary Algorithm for Uniform Parallel Machines Scheduling, in *IEEE Proceedings of the European Modelling Symposium*, 76-80, Liverpool, United Kingdom, 2008.

Cobârzan C., **Mihăilă C.**, A Genetic Algorithm for Utility Based Video Proxy-Caching, in *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 231-238, Timişoara, România, 2008

Mihăilă A., Mihiş A., **Mihăilă C.,** Genetic Algorithm for Logical Topic Text Segmentation, in *IEEE Proceedings of the International Conference on Digital Information Management*, 500-505, London, United Kingdom, 2008

Mihăilă A., **Mihăilă C.**, Uniform Parallel Machines Scheduling using an Evolutionary Algorithm, in *IEEE Proceedings of the International Workshop on Evolutionary Multiobjective Optimization Design and Applications*, International Conference on Intelligent Systems Design and Applications (ISDA), 401-406, Kaohsiung, Taiwan, 2008

Mihăilă C., Nițchi, I.Ş., R., Mihăilă, A., Coroş R., A genetic algorithm for permutation flow shop scheduling problem, in Annals of Tiberiu Popoviciu Seminar of Functional Equation, Approximation and Convexity, p. 241-250, Cluj-Napoca, România, 2008

Oltean M., Groşan C., Dioşan L., **Mihăilă C.,** *Genetic Programming with Linear Representation a Survey,* International Journal on Artificial Intelligence Tools, 197-238, 2009

Cuprins

Introducere

În ultimii ani, cercetarea în domeniul alocării de sarcini şi resurse a avut un impact tot mai mare asupra problemelor practice, o serie de tehnici de alocare făcându-şi cu adevărat loc în lumea dezvoltării aplicațiilor. Modele bazate pe constrângeri îmbină acum o înaltă flexibilitate reprezentațională cu managementul constrângerilor şi cu procedurile de căutare foarte scalabile. In mod similar, instrumentele de alocare matematică sunt acum capabile să abordeze probleme la o scară fără precedent, în timp ce meta-euristicile oferă capacități solide pentru optimizarea alocărilor. Având în vedere aceste succese precum şi progresele realizate, ar putea fi tentant să concluzionăm că principalele piedici tehnice care stau la baza problemelor de alocare de sarcini şi resurse au fost înlăturate. Totuşi, o astfel de concluzie (în cel mai bun caz) presupune o interpretare mai degrabă îngustă şi specializată a alocării şi (în cel mai rău caz) ignoră o mare parte a procesului şi a contextul mai larg al alocării din majoritatea mediilor cu aplicație practică [Smith2005].

Sumarizând starea de fapt actuală, putem identifica o serie de puncte tari din perspectivă tehnologică [Smith2005]:

- scalabilitatea – tehnicile actuale de alocare de sarcini şi resurse sunt capabile să rezolve probleme mari (adică zeci de mii de activități, sute de resurse) într-un timp rezonabil.
- flexibilitatea modelării – tehnicile actuale sunt capabile să genereze alocări sub constrângeri mari şi diversificate de natură temporală şi de capacitate a resurselor.
- optimizarea – cercetarea în domeniul aplicării structurilor de căutare globale, locale şi meta-euristice asupra problemelor de alocare a dus la o serie de abordări generale asupra optimizării alocărilor. În acelaşi timp, integrarea tehnicilor de cercetare bazate pe inteligența artificială cu instrumentele matematice de cercetare produce abilități de optimizare destul de puternice.

În ciuda avantajelor prezentate de tehnicile curente, problemele abordate sunt în general grele din punct de vedere al timpului nedeterminist polinomial şi sunt rezolvate doar aproximativ. Există mult spațiu pentru îmbunătățirea acestor tehnici, pentru acomodarea diferitelor clase de constrângeri precum şi pentru optimizarea în funcție de diferite seturi de criterii obiective [Smith2005].

Scopul lucrării de față este să investigheze utilizarea tehnicilor de calcul evolutiv în vederea rezolvării diferitelor clase de probleme de alocare de sarcini şi resurse. În acest sens, prima parte a lucrării (capitolele 1-3) prezintă unele aspecte teoretice, din literatura de specialitate, legate de teoria alocării de sarcini şi resurse şi a calculului evolutiv, în timp ce partea a doua (capitolele 4-6) introduce o serie de rezultate personale obținute în urma aplicării tehnicilor de calcul evolutiv unor clase de probleme de alocare.

Capitolul 1 introduce unele noțiuni de bază (sarcini, resurse, funcții obiectiv), concepte utilizate în teoria alocării de sarcini şi resurse precum şi o schemă de clasificare utilizată în codarea problemelor de alocare. Sunt prezentate de asemenea unele aspecte legate de problemele de alocare cum ar fi întârzierea în comunicare şi sarcinile multiprocesor, alocarea cu disponibilitate limitată a procesorului, alocarea cu resurse limitate şi alocarea multicriterială.

Capitolul 2 ilustrează complexitatea problemelor de programare. Reducția în timp polinomial şi ierarhia complexității sunt de asemenea prezentate.

Capitolul 3 este dedicat tehnicilor de calcul evolutive subliniind mecanismele de lucru ale acestora. Totodată, sunt prezentate şi o serie de aspecte legate de principalele elemente (reprezentarea, funcția fitness, selecția, încrucişarea, mutația, parametrii) care influențează performanța unui algoritm evolutiv.

Capitolul 4 prezintă un algoritm genetic pentru rezolvarea unei probleme de alocare de sarcni şi resurse pe maşini/procesoare paralele şi uniforma (uniform parallel machines scheduling problem). Maşinile/procesoarele uniforme reprezintă clase speciale de resurse în care maşinile/procesoarele au diferite viteze însă viteza este constantă şi nu depinde de sarcină.

Capitolul 5 prezintă o serie de rezultate obținute în urma aplicării unui algoritm genetic hibrid pentru determinarea unei soluții a unei probleme de alocare de tip permutation flow shop (permutation flow shop scheduling problem). Obiectivul unei astfel de probleme de alocare este să găsească o secvență pentru procesarea unui set de sarcini utilizând un set de maşini/procesoare astfel încât un anumit criteriu să fie optimizat, luându-se în calcul faptul că fiecare maşină/procesor procesează sarcina în aceeaşi ordine.

Capitolul 6 prezintă o problemă de alocare a unui video proxy-cache precum şi o serie de rezultate obținute în vederea determinării coeficienților unor funcții de utilitate care stau la baza mecanismului de înlocuire a cache-ului.

Principalele contribuții ale lucrării de față constau în:

- un nou algoritm genetic pentru o problemă de alocare pe maşini paralele şi uniforme [Mihăilă&Mihăilă2008a]. Algoritmul propus nu doar că obține rezultate mai bune decât alți algoritmi dar şi calculează rezultatul mai repede [Mihăilă&Mihăilă2008b].
- un nou algoritm genetic hibrid pentru o problemă de alocare de tip permutation flow shop [Mihăilă et.al.2008b]. Noutatea adusă de algoritmul propus constă în utilizarea unei combinații între o procedură de inițializare aleatorie şi o procedură de inițializare bazată pe euristica constructivă NEH, în definirea unui nou operator de încrucişare şi în utilizarea unui operator de mutație definit ca o combinație între un operator bazat pe euristica constructivă NEH şi mutația prin translatare (shift mutation). Rezultatele obținute de către algoritmul propus sunt comparabile cu rezultatele obținute de un algoritm de tip greedy iterativ.

- două noi metode de definire a utilității obiectelor stocate într-un video proxy-cache şi un nou algoritm genetic utilizat pentru determinarea coeficienților care apar în aceste două definiții cu scopul de a maximiza byte hit rate [Mihăilă&Cobârzan2008]. Rezultatele obținute de algoritmul propus şi cu una din cele două funcție de utilitate sunt similare sau chiar mai bune decât cele obținute prin intermediul altei metrici [Cobârzan&Mihăilă2008].

Partea A – Contextul

1. Alocarea de sarcini şi resurse

Scopul acestui capitol este să prezinte elementele de bază ale unei probleme de alocare (sarcini, resurse şi funcţii obiectiv) precum şi o serie de aspecte legate de aceste elemente. Este prezentată de asemenea şi o schemă de clasificare a problemelor de alocare de sarcini şi resurse

Alocarea de sarcini şi resurse se ocupă de alocarea de resurse limitate unor sarcni cu scopul de a optimiza una sau mai multe criterii (măsuri) de performanță [Leung2004].

1.1. Alocarea deterministă de sarcini şi resurse

Problemele de alocare de sarcini şi resurse sunt caracterizate de trei seturi [Blazewicz2007]:

- setul T= $\{T_1, T_2, \ldots, T_n\}$ a n sarcini,
- setul P = $\{P_1, P_2, \ldots, P_m\}$ a m procesoare sau maşini şi
- setul R = $\{R_1, R_2, \ldots, R_s\}$ a s tipuri de resurse adiționale.

Alocarea, în general vorbind, înseamnă atribuirea unui procesor/maşină din P şi (dacă este necesar) resurse din R unor sarcini din T pentru a îndeplini toate sarcinile sub constrângerile impuse.

In teoria alocării clasice există două constrângeri generale:

- fiecare sarcină poate fi procesată de către cel mult un/o procesor/maşină odată (plus posibilele cantități de resurse adiționale specificate) şi
- fiecare procesor/maşină este capabil să proceseze cel mult o sarcină odată (această constrângere poate fi relaxată).

Procesoarele pot fi fie paralele, adică procesoare care execută aceleaşi funcții, sau dedicate, adică specializate pe executarea anumitor sarcini [Blazewicz2007]. În funcție de viteză lor există trei tipuri de procesoarele paralele [Blazewicz2007]: identice (au viteze egale de

procesare a tuturor sarcinilor), uniforme (au viteze diferite de procesare însă viteza fiecărui procesor este constantă şi nu depinde de tipul sarcinii procesate) şi nerelaționate (au viteze diferite şi aceste viteze depind de tipul sarcinilor). În cazul procesoarelor dedicate există trei modele de procesare a seturilor de sarcini (un set de sarcini formează o activitate): [Blazewicz2007]: flow shop, open shop şi job shop.

În general o sarcină $T_j \in T$ este caracterizată de următoarele date [Blazewicz2007]:

- vectorul timpilor de procesare $p_j = [p_{1j}, p_{2j},\ldots, p_{mj}]^T$, unde p_{ij} este timpul necesar procesorului P_j pentru procesarea sarcinii T_j.
- momentul sosirii (sau momentul pregătirii) r_j, care reprezintă momentul în care sarcina T_j este pregătită pentru procesare. Dacă momentul sosirii este aceleaşi pentru toate sarcinile din T, atunci se presupune că $r_j = 0$ pentru orice j.
- data termen $\tilde{}d_j$, care specifică timpul limită până la care T_j trebuie să fie îndeplinită; de obicei sunt definite funcții de penalizare în concordanță cu acest timp limită.
- termenul limită $\tilde{}d_j$, care este un termen limită concret până la care sarcina T_j trebuie îndeplinită.
- prioritatea w_j, care exprimă gradul de urgență a sarcinii T_j.
- cererea de resurse suplimentare (dacă este cazul).

O alocare se numeşte întreruptibilă (preemptiv) dacă fiecare sarcină poate fi întreruptă în orice moment şi repornită ulterior fără nicio pierdere, probabil pe un alt procesor. Dacă nu este permisă întreruperea tuturor sarcinilor atunci alocarea se numeşte neîntreruptibilă [Blazewicz2007].

În setul T pot fi definite, printre sarcini, constrângeri de precedență. $T_i \prec T_j$ semnifică faptul că procesarea sarcinii T_i trebuie să fie terminată înainte ca sarcina T_j să poată începe să fie procesată. Cu alte cuvinte, în setul T se defineşte o relație de precedență $\prec$.

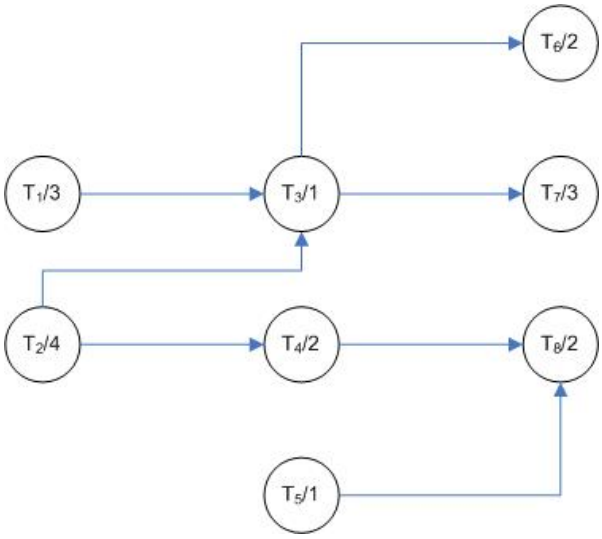


Figura 1.1 Un exemplu de precedență a sarcinilor [Blazewicz2007]

O alocare este o atribuire de procesoare din setul P (şi posibil resurse din setul R) unor sarcini din setul T, astfel încât următoarele condiții să fie satisfăcute [Blazewicz2007]:

- în fiecare moment fiecărui procesor îi este atribuită cel mult o sarcină şi fiecare sarcină este procesată de cel mult un procesor,
- sarcina T_j este procesată în intervalul de timp $[r_j, \infty)$,
- toate sarcinile sunt finalizate,
- dacă sarcinile T_i, T_j se află în relația $T_i \prec T_j$, procesarea sarcinii T_j, nu începe până când nu este finalizată sarcina T_i,
- în cazul unei alocări neîntreruptibile nicio sarcină nu se întrerupe altfel numărul întreruperilor fiecărei sarcini este finit,
- constrângerile legate de resurse, dacă există, sunt satisfăcute.

Programele pot fi reprezentate prin diagrama Gantt ca în Figura 1.2.

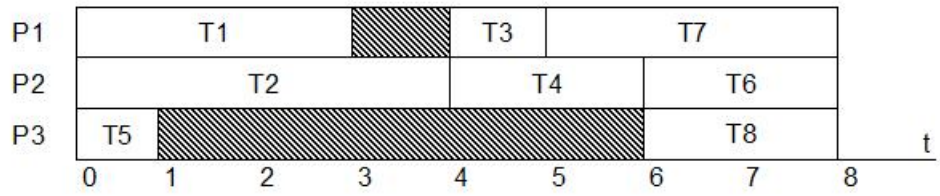


Figura 1.2 Exemplul unei diagrame Gantt [Blazewicz2007]

Pentru fiecare sarcină T_j, $j = 1, 2, ..., n$, procesată pot fi calculați următorii parametri [Blazewicz2007]:

- completion time C_j,
- flow time $F_j = C_j - r_j$, reprezentând suma timpurilor de aşteptare şi de procesare;
- lateness $L_j = C_j - d_j$,;
- tardiness $D_j = max\{C_j - d_j, 0\}$;
- earliness $E_j = max\{d_j - C_j, 0\}$.

Timpul de completare a unei sarcini este momentul la care procesarea ultimei operațiuni a sarcinii a fost finalizată. [Conway et. al. 1967]. Flow-time al unei sarcini este timpul total petrecut de către sarcină în shop [Conway et. al. 1967].

Pentru evaluarea alocărilor se folosesc următoarele criterii (măsurător) de performanță sau criterii de optimalitate [Blazewicz2007]:

- schedule length (makespan)

$$C_{max} = max\{C_j\},$$

- mean flow time

$$\overline{F} = \frac{1}{n}\sum_{j=1}^{n} F_j,$$

or mean weighted flow time

$$\overline{F_w} = \sum w_j Fj / \sum_{j=1}^{n} w_j,$$

- maximum lateness

$$L_{\max} = \max\{L_j\}$$

sau alte criterii înrudite.

O alocare pentru care valoarea unui anumit criteriu de performanță γ este la valaorea minimă se va numi optimal, iar valoarea corespunzătoare lui γ va fi reprezentată de γ^* [Blazewicz2007].

O problemă de alocare Π este definită ca un set de parametri dintre care nu toți au valori numerice, precum şi de un criteriu de optimalitate. O instanță I a problemei Π se obține precizând valori specifice pentru toți parametrii problemei [Blazewicz2007].

Un algoritm de alocare este un algoritm care construieşte un program pentru o problemă Π dată.

Teoria alocării de sarcini şi resurse se caracterizează printr-un număr nelimitat de tipuri de probleme [Brucker2007]. Pentru a face față acestei varietăți a problemelor de programare, s-a introdus un sistem de notare alcătuit din trei câmpuri $\alpha|\beta|\gamma$ [Brucker2007]:

- α descrie caracteristicile procesorului/maşini,
- β descrie caracteristicile sarcinii şi a resurselor,
- γ denotă un criteriu de optimalitate (măsură a performanței).

1.2. Întârzierea în comunicare şi sarcinile multiprocesor

În ultimii ani, având în vedere dezvoltarea rapidă a sistemelor paralele şi distribuite, constrângerea care impune ca fiecare sarcină să fie executată de un singur procesor odată, poate fi relaxată. În acest context, întârzierile cauzate de comunicarea între sarcini nu pot fi ignorate. Există trei modele care descriu problemele de comunicare în contextul problemelor de alocare. În acest sens, modelul sarcinilor din teoria aloării clasice a fost îmbogățit pentru a încorpora întârzierile cauzate de comunicare. Aceste întârzieri pot fi gestionate implicit sau explicit. În primul caz, intervalele de comunicare sunt deja incluse în timpul alocat procesării sarcinii. De obicei, o sarcină necesită mai mult de un procesor odată. O astfel de sarcină se numeşte sarcină multiprocesor. Sarcinile multiprocesor pot specifica cerințele procesorului fie în termeni de procesoare solicitate simultan, fie în termenii unei specificări explicite a unui set de procesore (sau seturi de procesoare subsets) care este/sunt necesară/necesare pentru procesare. În primul caz vom vorbi despre cerințe ale procesoarelor paralele, în timp ce în al doilea caz vom vorbi despre cerințe ale procesoarelor dedicate [Blazewicz2007].

1.3. Alocarea cu disponibilitate limitată a procesorului

Un sistem de procesoare/maşini cu disponibilitate limitată este un set de procesoare/maşini care nu operează continuu; fiecare procesor/maşină fiind pregătit/pregătită pentru procesare doar în anumite intervale temporale de disponibilitate [Blazewicz2007]. În acest caz se doreşte determinarea unei alocării fezabile, dacă există, astfel încât toate sarcinile să poată fi procesate în intervalele de disponibilitate date ale procesoarelor/maşinilor, optimizând anumite criterii de performanță.

Termenul întreruptibil este utilizat conform definiției anterior menționate. Adesea, în locul acestui termen se foloseşte termenul de reluare (resumability). În acest scenariu în care activitatea se reia, o sarcină poate fi întreruptă atunci când o maşină nu mai este disponibilă şi reluată în momentul în care maşina redevine disponibilă, fără a se aplica vreo penalizare. În situația în care activitatea nu se reia, întreruperea este de obicei interzisă. Cel mai general scenariu este cel de semi-reluare (semi-resumability) [Blazewicz2007].

1.4. Alocarea cu constrângeri de resurse

Modelul de alocare cu constrăngeri de resurse este mai complicat decât cele precedente, deoarece orice sarcină, pe lângă procesoare/maşini, mai poate solicita pentru procesare şi anumite resurse adiționale, limitate.

Resursele, în funcție de natura lor, pot fi clasificate în tipuri şi categorii [Blazewicz2007]. Clasificarea pe tipuri ia în considerare doar funcțiile pe care le îndeplinesc resursele: resursele de acelaşi tip se presupune că îndeplinesc aceleaşi funcții [Blazewicz2007]. Clasificarea pe categorii are în vedere două aspecte. Mai întâi, se disting trei categorii de resurse din punct de vedere al constrângerilor resurselor. Vom numi o resursă regenerabilă numai dacă întreaga sa utilizare (adică disponibilitatea temporală în fiecare moment) se află sub constrângeri. O resursă se numeşte neregenerabilă doar dacă întreg consumul său (adică disponibilitatea integrală până la orice moment dat) se află sub constrângeri (cu alte cuvinte, odată ce această resursă a fost utilizată de către o sarcină nu mai poate fi utilizată de către altă sarcină). O resursă se numeşte dublu-constrânsă dacă atât utilizarea totală şi consumul total sunt constrânse. În al doilea rând, se disting două categorii de resurse din punct de vedere al divizibilității resurselor: resurse discrete (adică discret-divizibile) şi resurse continue (adică continuu-divizibile). U alte cuvinte, printr-o resursă discretă vom înțelege o resursă care poate fi alocată sarcinilor în cantități discrete dintr-un set finit de alocări posibile, care, în mod special, poate să consiste într-un singur element. Resursele continue, pe de altă parte, pot fi alocate în cantități arbitrare, ne prestabilite, din intervale date [Blazewicz2007].

1.5. Probleme de alocare multicriterială

Multe probleme de alocare din domeniul producției sau al serviciilor implică mai multe criterii. Ca regulă generală, luarea în calcul a unei serii de criterii facilitează oferirea unei soluții mai realiste factorului de decizie. O problemă de alocare multicriterială este o problemă care constă în calcularea unui optim Pareto pentru mai multe criterii conflictuale. Această problemă poate fi divizată în trei sub-probleme [T'kindt&Billaut2006]:

- modelarea problemei, a cărei rezolvare duce la determinarea naturii problemei de alocare date, precum şi la determinarea definiției criteriului care se ia în calcul,
- luarea în calcul a criteriului, a cărei rezolvare duce la indicarea contextului de rezolvare şi la modul în care vrem să se ia în calcul criteriul. Analistul finalizează un modul de ajutor de decizie pentru problema multicriterială, de asemenea numit modul pentru luarea în calcul a criteriului.
- alocarea, a cărei rezolvare ne duce la găsirea soluției pentru problemă. Analistul finalizează un algoritm pentru rezolvarea problemei de programare, numit şi modul de rezolvare a problemei de programare.

În etapa de luare în calcul a criteriului şi ținând cont de informațiile pe care acesta le stabileşte, analistul alege o abordare a rezolvării problemei de alocare şi astfel se defineşte o problemă de alocare. Luând în calcul diversitatea de metode de determinare a optimului Pareto, funcțiile de optimizat pentru problema de alocare pot lua diferite forme. Fiecare se traduce printr-o metodă de determinare a optimului Pareto. Criteriile nu se schimbă şi corespund acelor criterii definite în cadrul etapei de modelare a problemei [T'kindt&Billaut2006].

O problemă de programare multicriterială, după etapa de modelare, poate fi notată într-un mod general folosind sistemul de notare cu trei câmpuri, unde câmpul γ conține lista de criterii: $\alpha \mid \beta \mid Z_1, Z_2, \ldots, Z_k$.[T'kindt&Billaut2006].

2. Complexitatea problemelor de alocare

Scopul acestui capitol este să arate complexitatea problemelor de alocare de sarcini şi resurse în general şi să prezinte ierarhia problemelor descriind relaţiile dintre diferite probleme de alocare.

Teoria complexităţii reprezintă un instrument important în cercetarea domeniului de alocare de sarcini şi resurse [Leung2004].

Aceasta oferă un cadru matematic în care problemele de calcul sunt studiate astfel încât să poată fi clasificate drept "uşoare" sau „grele" [Brucker2007]. Această clasificare este utilă pentru a vedea dacă există un algoritm eficient, în special din punct de vedere al timpului, pentru rezolvarea unei anumite probleme. O problemă ține de o clasă de complexitate care ne oferă informaţii despre complexitatea "celui mai bun algoritm" capabil să o rezolve. Astfel, dacă o anumită problemă se dovedeşte a aparține clasei problemelor "uşoare", înseamnă că putem să găsim un algoritm în timp polinomial pentru a o rezolva. De obicei aceasta este o veste bună, dar din păcate acest lucru nu se întâmplă foarte des în cazul problemelor complexe. Prin urmare, dacă o problemă aparține clasei problemelor grele nu poate fi rezolvată în timp polinomial ceea ce, cu alte cuvinte, implică faptul că pentru unele instanțe timpul CPU necesar rezolvării devine „exponențial" [T'kindt&Billaut2006].

2.1. Probleme, algoritmi şi complexitate

O problemă Π este descrisă dacă se dă [Garey&Johnson1979]:

- o descriere generală a tuturor parametrilor săi şi
- un enunț despre ce proprietăți trebuie să satisfacă răspunsul sau soluția.

O instanță *I* a problemei Π se obține precizând valori specifice pentru toți parametrii problemei [Garey&Johnson1979]. Fiecărei instanțe i se atribuie o „mărime". Mărimea unei instanțe se referă la lungimea şirului de date necesar pentru a preciza instanța şi ea depinde de magnitudinea celui mai mare element [T'kindt&Billaut2006]. Aceasta se mai numeşte şi lungime (mărime) a schemei de codare [Pinedo2008]. O schemă de codare cartografiază instanțele problemei sub formă de şiruri care le descriu [Garey&Johnson1979].

Algoritmii sunt proceduri generale de rezolvare a problemelor pas cu pas. Un algoritm rezolvă o problemă Π dacă găseşte o soluție pentru orice instanță I a problemei Π [Blazewicz et. al. 2007].

În general, ne interesează să găsim cel mai „eficient" algoritm pentru rezolvarea unei probleme. În sensul său cel mai larg, noțiunea de eficiență implică toate diferitele resurse necesare pentru rezolvarea algoritmului. Totuşi, de obicei, prin „cel mai eficient " se înțelege cel mai rapid. De vreme ce cerințele referitoare la timp sunt adesea factorul dominant care determină dacă un anumit algoritm este sau nu suficient de eficient în practică, acesta este singura resursă luată în calcul în contextul analizei complexității unui algoritm [Garey&Johnson1979].

Timpul de funcționare a unui algoritm este măsurat în funcție de numărul etapelor de calcul de bază pe care le efectuează [Leung2004]. Pentru a defini o etapă de calcul se foloseşte un model standard de calcul, maşina Turing. Orice text standard referitor la complexitatea calculului conține presupunerile făcute de maşina Turing [Pinedo2008].

Teoretic, funcția de complexitate a timpului unui algoritm A car rezolvă o problemă Π este o funcție care cartografiază fiecare lungime intrării (input) a unei instanțe I a Π reprezentând un număr maxim de etape elementare (sau unități de timp) ale unui computer, necesare pentru rezolvarea unei instanțe a mărimii respective prin algoritmul A [Blazewicz et. al. 2007]. Această funcție nu este bine definită până când nu se realizează [Garey&Johnson1979]:

- schema de codare de folosit la determinarea lungimii (mărimii) input şi
- computerul sau modelul de computer de folosit pentru determinarea timpului de execuție a etapelor de bază.

Diferiți algoritmi au o varietate mare de funcții de complexitate a timpului şi definirea celor care sunt „suficient de eficiente" şi a celor care sunt „prea ineficiente" va depinde întotdeauna de situație. Totuşi, informaticienii recunosc o mică diferență care oferă informații aprofundate legate de aceste aspecte. Este vorba de diferența dintre algoritmii în timp polinomial şi algoritmi în timp exponențial [Garey&Johnson1979].

Aşa cum sa menționat anterior, eficiența unui algoritm se măsoară în comparație cu limita superioară (upper bound) $T(n)$ la numărul etapelor de calcul efectuate de algoritm pentru a rezolva instanța I a unei probleme Π [Brucker2007]. Cu alte cuvinte eficiența unui algoritm pentru o anumită problemă se măsoară în funcție de numărul maxim (cel mai rău caz) de etape de calcul necesare pentru a obține o soluție optimă ca funcție a mărimii instanței [Pinedo2008].

În cele mai multe cazuri va fi greu să se calculeze forma exactă a lui T. De aceea, forma exactă a lui T este înlocuită cu ordinul său asimptotic. De aceea, spunem că $T(n) \in O(g(n))$ dacă există constante $c > 0$ şi un număr întreg nenegativ n_0 astfel încât $T(n) \leq cg(n)$ pentru toate numerele întregi $n \geq n_0$ [Brucker2007].

Un algoritm în timp polinomial este definit ca fiind un algoritm a cărui funcție de complexitate a timpului este $O(g(n))$ pentru o funcție polinomială g, unde n este folosit pentru a reprezenta lungimea intrării (input) [Garey&Johnson1979]. Orice algoritm a cărui funcție de complexitate a timpului nu se poate limita în acest fel, se numeşte algoritm în timp exponențial (totuşi, ar trebui notat faptul că această definiție include anumite funcții de complexitate a timpului non-polinomial, cum ar fi $n^{\log n}$, care nu sunt în mod normal considerate funcții exponențiale) [Garey&Johnson1979].

Diferențierea între aceste două tipuri de algoritmi are o importanță semnificativă atunci când luăm în considerare soluția unor instanțe mari ale unor probleme.

Este în majoritate acceptat faptul că o problemă nu a fost „bine rezolvată" până când nu se cunoaşte un algoritm în timp polinomial pentru aceasta. De aceea, o problemă se numeşte greu de rezolvat(intractable) dacă este atât de grea încât nici un algoritm în timp polinomial nu o poate rezolva [Garey&Johnson1979]. Definiția termenului "intractable" oferă un cadru teoretic cu o generalitate şi o putere considerabile. Caracterul de "intractability" al unei probleme se dovedeşte a fi fundamental independentă de schema de codare respectivă şi de modelul de computer utilizat pentru determinarea complexității timpului [Garey&Johnson1979] dacă sunt utilizate scheme de codare şi modele de computer „rezonabile ".

O schemă de codare „rezonabilă" este una care satisface următoarele două condiții:

- codarea unei instanțe *I* trebuie să fie concisă iar nu „ticsită" cu informații sau simboluri inutile şi
- cifrele care apar în *I* trebuie să fie reprezentate în baza doi (sau zece sau opt sau în orice altă bază fixă, alta decât 1),

în timp ce un model de computer "rezonabil" este unul în care există o limită polinomială asupra cantității de lucru care se poate efectua într-o singură unitate de timp (astfel, de exemplu, un model care are capacitatea de a efectua în mod arbitrar mai multe operații în paralel nu ar fi considerat „rezonabil" şi, într-adevăr, nici un calculator existent (sau în stare de proiect) nu are această capacitate [Garey&Johnson1979].

Definiția termenului intractability a permis distingerea între două cauze diferite. Prima, care este de obicei prima la care ne gândim, este că problema este atât de dificilă încât este nevoie de o cantitate exponențială de timp pentru a-i găsi o soluție. A doua cauză este că soluția în sine se necesită a fi atât de extinsă încât nu poate fi descrisă printr-o expresie cu o lungime limitată de o funcție polinomială a lungimii intrării [Garey&Johnson1979]. În cele ce urmează ne vom îndrepta atenția asupra primului tip de intractability (se vor lua în considerare doar problemele pentru care lungimea soluției este limitată de o funcție polinomială a lungimii inputului).

2.2. Reducerea polinomială

În timp ce teoreticienii continuă să caute metode mai puternice prin care să dovedească că unele probleme sunt intractable, în paralel se desfăşoară eforturi de a afla mai multe despre modurile în care diverse probleme sunt înrudite în ceea ce priveşte dificultatea. Principala tehnică utilizată pentru a demonstra că două probleme sunt înrudite este aceea a „reducerii" una la cealaltă, oferind o transformare constructivă care cartografiază fiecare instanță din prima problemă cuplând-o cu o instanță echivalentă din cea de-a doua. O astfel de transformare oferă mijloacele pentru transformarea oricărui algoritm care rezolvă a doua problemă într-un algoritm corespondent pentru rezolvarea celei de-a doua probleme [Garey&Johnson1979].

Spunem că problema P se reduce la problema P' dacă pentru fiecare instanță a P se poate construi o instanță echivalentă a P' can be constructed. În teoria complexității se foloseşte de obicei un termen mai riguros. Problema P se reduce în timp polinomial la problema P' dacă un

algoritm în timp polinomial pentru P' implică un algoritm în timp polinomial pentru P. reducerea polinomială a lui P la P' este denotată de $P \propto P'$. Dacă se ştie că dacă nu există un algoritm în timp polinomial pentru problema P, atunci nu există un algoritm în timp polinomial nici pentru problema P'[Pinedo2008].

Noțiunea de reducție în timp polinomial stă la baza teoriei dificultății timpului nedeterminist polinomial. Această teorie se aplică doar la problemele de decizie [Leung2004]. O problemă de decizie este o problemă pentru care răspunsul este „da" sau „nu". Având în vedere că majoritatea problemelor de programare sunt probleme de optimizare, se pare că teoria dificultății timpului nedeterminist polinomial nu este foarte folositoare în teoria programării. Dar orice problemă de optimizare (maximizare sau minimizare) poate fi convertită într-o problemă de decizie corespondentă, adăugând un parametru ω şi pur şi simplu punând întrebarea dacă există o soluție fezabilă astfel încât costul soluției să fie mai mic sau egal (sau mai mare sau egal în cazul problemelor de maximizare) decât ω [Leung 2004].

O problemă se numeşte soluționabilă din punct de vedere al timpului polinomial dacă există un p polinomial astfel încât $T(n) \in O(p(n))$ unde n este lungimea intrării cu privire la o schemă de codare „rezonabilă", adică dacă există un k astfel încât $T(n) \in O(n^k)$. Dacă pentru o problemă T(n) este polinomial cu privire la o codare unară atunci problema se numeşte pseudo-polinomială [Brucker2007].

Clasa în care se încadrează toate problemele de decizie soluționabile din punct de vedere polinomial este denumită P [Brucker 2007].

Timpul nedeterminist polinomial se referă la clasa problemelor de decizie care au certificate „succinte" care pot fi verificate în timp polinomial. Certificatele "succincte" sunt acelea ale căror mărime este limitată de o funcție polinomială a mărimii intrării [Leung2004].

Despre o problemă de decizie Q spunem că este completă din punct de vedere al timpului nedeterminist polinomial [Leung2004] dacă:

- Q face parte din clasa timpului nedeterminist polinomial şi
- Toate problemele din clasa timpului nedeterminist polinomial se pot reduce la Q.

Spunem despre o problemă că este dificilă din punct de vedere al timpului nedeterminist polinomial dacă satisface doar a doua condiție din definiția de mai sus [Leung2004]. Nu toate problemele din clasa celor dificile din punct de vedere al timpului nedeterminist polinomial sunt la fel de dificile. Unele sunt mai dificile decât altele. De exemplu, se poate ca o problemă să poată fi soluționată în timp polinomial ca o funcție a mărimii problemei în codare unară, dar să nu poată fi soluționată în timp polinomial ca funcție a mărimii problemei în codare binară. Pentru alte probleme se poate să nu existe algoritmi în timp polinomial nici în codare unară nici binară. În prima clasă problemele sunt mai dificile decât în cea de-a doua. De obicei problemele din prima clasă sunt numite dificile din punct de vedere al timpului nedeterminist polinomial în sensul clasic (ordinary sense) sau pur şi simplu dificile din punct de vedere al timpului nedeterminist polinomial. Algoritmii utilizați pentru această clasă de probleme se numesc pseudo-polinomiali. A doua clasă de probleme este denumită de obicei ca fiind foarte dificilă din punct de vedere al timpului nedeterminist polinomial [Pinedo2008].

Pentru a demonstra că o problemă este completă din punct de vedere al timpului nedeterminist polinomial, trebuie să dovedim că toate problemele din clasa timpului

nedeterminist polinomial se pot reduce la acea problemă. Deoarece în clasa timpului nedeterminist polinomial există un număr infinit de probleme nu este clar cum ar putea cineva demonstra că o problemă este completă din punct de vedere al timpului nedeterminist polinomial. Din fericire, Cook [Cook1971] a adus o dovadă că o problemă de satisfiabilitate este completă din punct de vedere al timpului nedeterminist polinomial, realizând o reducție generică de la maşinile Turing la satisfiabilitate [Leung2004]. Pornind de la problema de satisfiabilitate, se poate dovedi că şi alte probleme sunt complete din punct de vedere al timpului nedeterminist polinomial reducându-le la problemele țintă. Deoarece reductibilitatea este tranzitivă, acest lucru este echivalent cu a dovedi că toate problemele din clasa timpului nedeterminist polinomial sunt reductibile la probleme țintă. Pornind de la satisfiabilitate, Karp [Karp1972] a demonstrat că un mare număr de probleme de combinatorică sunt complete din punct de vedere al timpului nedeterminist polinomial [Leung2004].

Problema de satisfiabilitate este definită în felul următor: dat fiind un set de variabile şi o serie de condiții definite în funcție de variabile, există o atribuire a valorilor la variabile pentru care fiecare din condiții să fie adevărată?[Pinedo2008]. Problema în care fiecare condiție conține exact 3 literali se numeşte problemă 3-satisfiabilitate (3-SAT) [Brucker2007].

Diagrama din Figura 2.1 arată câteva transformări polinomiale de bază între câteva probleme. Un arc de la P la Q în Figura 2.1 indică faptul că $P \propto Q$. Deoarece toate problemele din Figura 2.1 țin de clasa celor complete din punct de vedere al timpului nedeterminist polinomial toate aceste probleme sunt complete din punct de vedere al timpului nedeterminist polinomial [Brucker2007].

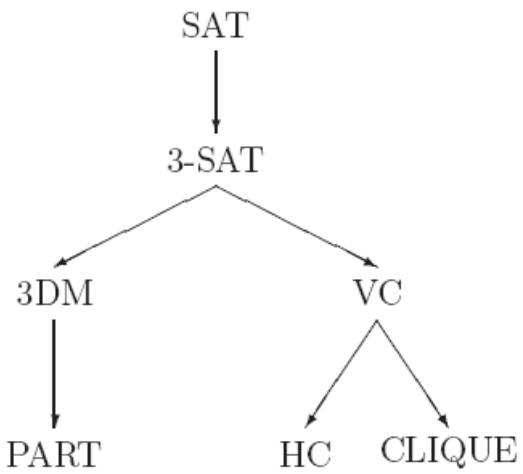


Figura 2.1 Transformări polinomiale de bază [Brucker2007].

Problemele de partiționare (PART), circuit Hamiltonian (HC) şi clică (CLIQUE) sunt deosebit de importante în teoria alocării de sarcini şi resurse deoarece problemele a căror complexitate este stabilită printr-o reducție dintr-o problemă de partiție de obicei permit algoritmi în timp psudo-polinomial şi de aceea sunt dificile din punct de vedere al timpului nedeterminist polinomial în sensul clasic (ordinary sense). Problemele dificile din punct de vedere al timpului nedeterminist polinomial la care complexitatea este stabilită prin reducerea de

la satisfiabilitate, 3-partiție, circuit hamiltonian sau clică sunt foarte dificile din punct de vedere al timpului nedeterminist polinomial [Pinedo2008].

2.3. Ierarhia complexității

Există în literatura de specialitate o serie de rezultate tradiționale pentru calcularea complexității problemelor de alocare. Aceste rezultate demonstrează legătura dintre diferite probleme de alocare deterministe cu un singur criteriu. Dacă o problemă de alocare de sarcini şi resurse se reduce la o altă problemă de alocare, algoritmul pentru una din probleme poate fi aplicat şi la cealaltă [T'kindt&Billaut2006].

S-au făcut eforturi deosebite pentru a stabili o ierarhie a problemelor care să descrie relațiile dintre sute de probleme de alocare. Din comparația dintre complexitățile diferitelor probleme de alocare ne interesează să aflăm cum o schimbare la un singur element din clasificarea unei probleme îi afectează complexitatea. În Figurile 2.2 - 2.4 sunt exemplificate o serie de grafice care ajută la determinarea ierarhiei complexității problemelor de alocare deterministe [Pinedo2008]. Aceste grafice ilustrează reducțiile polinomiale între problemele de alocare. Astfel de grafice există pentru tipuri de probleme (figura 2.2), tipuri de constrângeri (figura 2.3) şi criterii (figura 2.4) [T'kindt&Billaut2006]:

- în figura 2.2, prezența unui arc de la A către B înseamnă că există o reducție polinomială de la o problemă $A|\beta|\gamma$ la o problemă corespunzătoare $B|\beta|\gamma$.
- în figura 2.3, prezența unui arc de la A către B înseamnă că există o reducție polinomială de la o problemă $\alpha|A|\gamma$ la o problemă corespunzătoare $\alpha|B|\gamma$.
- în figura 2.4 prezența unui arc de la A către B înseamnă că există o reducție polinomială de la o problemă $\alpha|\beta|A$ la o problemă corespunzătoare $\alpha|\beta|B$ problem.

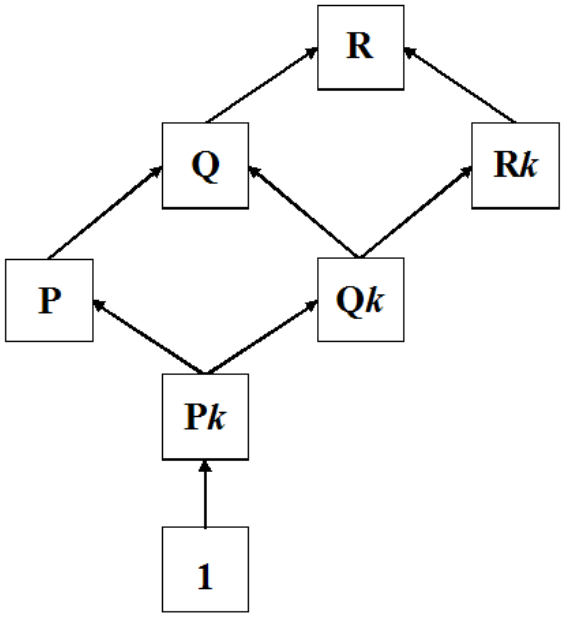
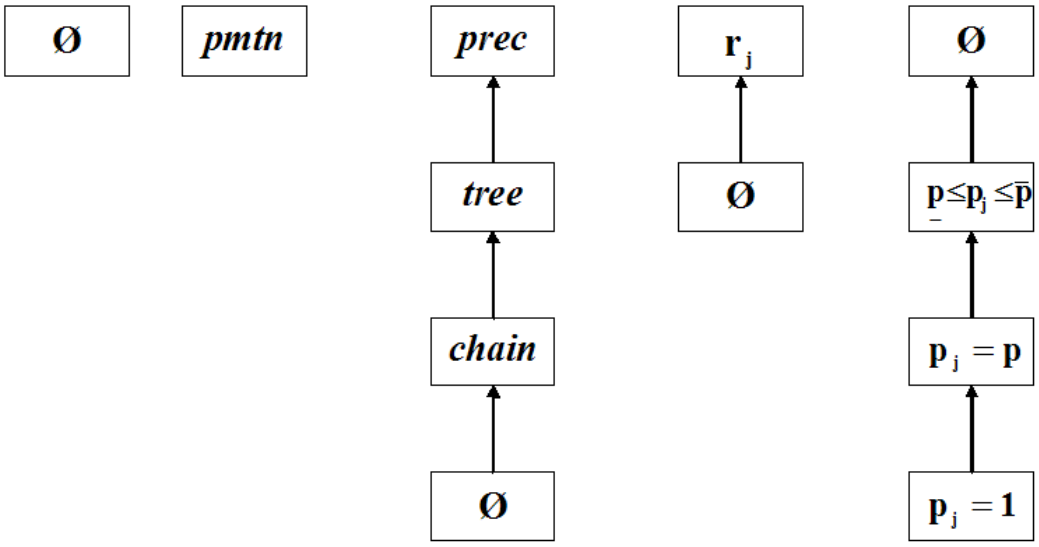
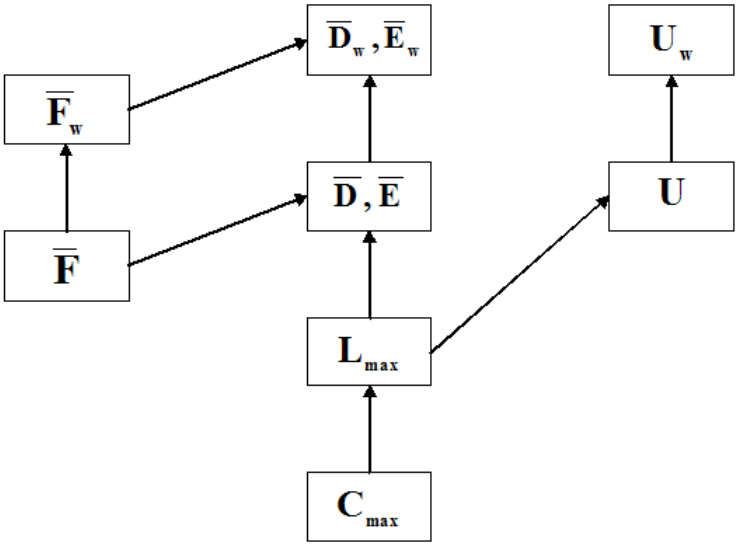


Figura 2.2 Grafic al reducției pentru caracteristicile procesorului [Blazewicz et.al. 2007]



Fagure 2.3 Grafic al reducției pentru caracteristicile sarcinilor şi a resurselor [Blazewicz et.al. 2007]



Fagure 2.4 Grafic al reducției pentru criteriile de optimalitate [Blazewicz et.al. 2007]

Graficele de reducție prezentate sunt utilizabile doar atunci când ştim deja care este complexitatea problemelor de alocare respective. S-au depus eforturi considerabile pentru stabilirea rezultatelor de complexitate (polinomial soluționabile, pseudo-polinomial soluționabile şi dificile din punct de vedere al timpului nedeterminist polinomial) al unei serii de probleme de alocare [Brucker2007].

3. Calculul evolutiv

Scopul acestui capitol este să ofere o privire de ansamblu asupra elementelor de bază ale unui algoritm evolutiv (reprezentare, funcția fitness, mutație şi parametri) precum şi să ofere o serie de reprezentări ale unor programări fezabile.

Problemele de alocare de sarcini şi resurse sunt probleme de optimizare. Când ne ocupăm de o problemă de programare trebuie întotdeauna să îi aflăm complexitatea, deoarece aceasta determină natura algoritmului care trebuie implementat. Dacă problema respectivă aparține clasei P, ştim că există un algoritm în timp polinomial exact pentru a o rezolva. În acest caz este convenabil să folosim sau să perfecționăm un astfel de algoritm. În schimb, dacă o problemă este dificilă din punct de vedere al timpului nedeterminist polinomial există două alternative. Prima este să se propună un algoritm aproximat, atică unul euristic, care calculează în timp polinomial o soluție care este cât de apropiată posibil de soluția optimă. A doua este să se propună un algoritm care calculează soluția optimă pentru problemă dar a cărui complexitate maximală este exponențială. În acest caz, provocarea constă în a crea un algoritm care să poată rezolva probleme de cele mai mari dimensiuni posibile [T'kindt&Billaut2006]. În această lucrare abordăm prima alternativă utilizând tehnici ale calculului evolutiv pentru a propune un algoritm aproximat pentru problemele de programare.

3.1. Algoritmi evolutivi

Calculul evolutiv (CE) se referă la sisteme de rezolvare a problemelor bazate pe calcul, sisteme ce utilizează modele de calculare a proceselor evolutive, cum ar fi selecția naturală, supraviețuirea celui mai adaptat şi reproducerea, ca şi componente fundamentale ale unui astfel de sistem de calcul [Engelbrecht2002]. Evoluția pe calea selecției naturale a unei populații de indivizi selectați arbitrar poate fi considerată ca o căutare în spațiu a unor posibile valori cromozomiale. În acest sens un algoritm evolutiv (AE) reprezintă o căutare stocastică a unei soluții optime pentru o problemă dată [Engelbrecht2002].

Figura 3.1 reprezintă o schiță a unui algoritm evolutiv simplu [Ahn2006].

Pasul 1. Inițializare
    Generează populația inițială *P* în mod arbitrar sau pe baza unor cunoştințe avute în prealabil
Pasul 2. Evaluarea adaptării (fitness)
    Evaluează calitatea (adaptarea) fiecărui individ din *P*
Pasul 3. Selecția
    Selectează un set de candidați promițători *S* din *P*
Pasul 4. Reproducerea
    Step 4.1. Încrucişarea (opțional)
        Aplică încrucişarea în bazinul (pool) de împerechere *S* pentru a genera un set de fii *O*
    Step 4.2. Mutația (probabilistic)
        Aplică mutația asupra setului de fii *O* pentru a obține setul modificat *O'*
Pasul 5. Înlocuirea
    Înlocuieşte populația actuală *P*
Pasul 6. Terminarea
    Dacă criteriile de terminare nu sunt îndeplinite, mergi la pasul 2.

Figura 3.1 Pseudocod pentru algoritmi evolutivi

Acest algoritm evolutiv simplu este mai complex decât pare la prima vedere. Există cinci decizii importante care acționează ca factori asupra designul algoritmului [Ashlock2006]: Ce structură a datelor veți folosi? Ce funcție fitness veți folosi? Ce operații de reproducere (încrucişare, mutație) veți folosi? Cum veți selecta părinții din populație şi cum veți introduce copiii în populație? Ce condiție de terminare va pune capăt algoritmului?

Există mai multe paradigme CE [Engelbrecht2002]: Algoritmi Genetici (AG), Programare Evolutivă (PE), Strategii de Evoluție (SE), Programare Genetică (PG), Evoluție Difirențiată (ED), Evoluție Culturală (EC), Co-evoluție (CoE). În cele ce urmează ne vom referi doar la algoritmi genetici (AG), Programare Evolutivă (PE) şi Strategii de Evoluție (SE).

Reprezentare

Deoarece structura soluției variază de la problemă la problemă, o soluție a unei probleme anume poate fi reprezentată în mai multe moduri. De obicei, o metodă de căutare este cea mai eficientă în cazul abordării unei anumite reprezentări şi este mai puțin eficientă în abordarea altor reprezentări. Astfel, alegerea unei scheme de reprezentare eficiente depinde nu doar de problema de bază ci şi de metoda de căutare aleasă. Eficiența şi complexitatea unui algoritm de căutare depinde în mare măsură de modul în care soluțiile au fost reprezentate şi de cât de potrivită este reprezentarea în contextul operatorilor de căutare de bază. În unele cazuri, o problemă dificilă poate fi simplificată prin alegerea unei reprezentări potrivite care funcționează eficient cu un anumit algoritm [DeJong1997].

Populația inițială

Algoritmii evolutivi sunt algoritmi stohastici de căutare pe baza populației. Fiecare AE menține aşadar o populație de soluții – candidat. Primul pas în aplicarea EA pentru rezolvarea

unei probleme de optimizare este generarea populației inițiale. Modalitatea standard de a genera o populație inițială este să se atribuie o valoare arbitrară din domeniul permis fiecărei gene din fiecare cromozom. Scopul selecției arbitrare este să ne asigurăm că populația inițială este o reprezentare uniformă a întregului spațiu de căutare. Dacă unele regiuni din spațiu nu sunt acoperite de către populația inițială, există şanse ca acele părți să fie neglijate de către procesul de căutare. Mărimea populației inițiale are consecințe în termeni de complexitate a calculului şi de abilități de explorare [Engelbrecht2002].

Funcția Fitness

În modelul evoluționist al lui Darwin, indivizii cu cele mai bune caracteristici au cele mai mari şanse să supraviețuiască şi să se reproducă. Pentru a determina abilitatea unui individ dintr-un AE de a supraviețui, este folosită o funcție matematică, numită funcția fitness, pentru a cuantifica cât de bună este soluția reprezentată de un cromozom. Funcția fitness are un rol important într-un algoritm evolutiv deoarece operatorii evolutivi de obicei se folosesc de funcția fitness a cromozomilor [Engelbrecht2002].

Selecția

Selecția reprezintă unul dintre principalii operatori utilizați în algoritmii evolutivi. Obiectivul principal al operatorului de selecție este să evidențieze soluțiile mai bune dintr-o populație. Acest operator nu creează noi soluții, ci selectează soluțiile relativ bune dintr-o populație ştergând restul soluțiilor care nu sunt aşa de bune [DeJong1997]. Identificarea unei soluții bune sau rele în cadrul unei populații se realizează de obicei ținând cont de funcția fitness. Ideea de bază este că o soluție care are un fitness mai bun (care este mai bine adaptată) trebuie să aibă o probabilitate mai mare de selecție. Totuşi, operatorii de selecție diferă în modul în care atribuie copiile soluțiilor mai bune. Unii operatori sortează populația în funcție de fitness şi aleg în mod determinist cele mai bune câteva soluții, în timp ce alți operatori atribuie o probabilitate de selecție fiecărei soluții în funcție de fitness şi fac o copie folosindu-se de acea distribuție de probabilitate [DeJong1997].

Există diverse tipuri de operatori de selecție cum ar fi selecția proporțională, selecția turnir, selecția în funcție de rang etc. Operatorii de selecție sunt caracterizați de presiunea lor de selecție numită şi timpul de preluare (takeover time), care se raportează la timpul necesar pentru a produce o populație uniformă. Acesta este definit ca fiind viteza cu care cea mai bună soluție ocupă întreaga populație prin aplicarea repetată doar a operatorului de selecție. Un operator cu presiune de selecție mare face să scadă diversitatea în cadrul populației mai repede decât operatorii cu o presiune de selecție scăzută, lucru care poate duce la convergența prematură a soluțiilor suboptime. O presiune de selecție mare limitează abilitățile de explorare ale populației [Engelbrecht2002].

Reproducerea (încrucişarea şi mutația)

Reproducerea este procesul de producere de noi candidați din părinți selectați, aplicând operatori de încrucişare şi/sau mutație.

Încrucişarea este procesul de creare a unuia sau a mai multor indivizi prin combinarea materialului genetic selectat în mod arbitrar de la doi sau mai mulți părinți. Dacă selecția se

axează pe indivizii cei mai adaptați, presiunea selecției poate cauza convergența prematură datorată diversității reduse a noilor populații [Engelbrecht2002].

Mutația este procesul prin care se schimbă în mod arbitrar valorile genelor într-un cromozom. Principalul scop al mutației este să introducă material genetic nou în populație, crescând astfel diversitatea genetică [Engelbrecht2002].

Criteriul de oprire

Operatorii evolutivi sunt aplicați în mod iterativ într-un AE până când condiția de oprire este satisfăcută. Cea mai simplă condiție de oprire este să se limiteze numărul de generații pe care AE are voie să îl producă , sau se stabileşte o limită a numărului de evaluări ale funcției fitness. Această limită nu trebuie să fie prea mică, altfel AE nu va avea suficient timp să exploreze spațiul de căutare [Engelbrecht2002]. Pe lângă o limită a timpului de execuție, se foloseşte de obicei şi un criteriu de convergență pentru a detecta dacă populația a convers. Convergența este vag definită ca fiind momentul în care populația stagnează. Cu alte cuvinte, atunci când nu mai are loc nicio schimbare genotipică sau fenotipică în cadrul populației [Engelbrecht2002]:

3.2. Clasificarea tehnicilor de control al parametrilor

Problema stabilirii valorilor diferiților parametri ai unui algoritm evolutiv (AE) este crucială pentru obținerea unei bune performanțe. În clasificarea tehnicilor de control al parametrilor unui algoritm evolutiv pot fi luate în considerare mai multe aspecte [Siarry&Michalewicz2008]:

- Ce se schimbă (ex. reprezentarea, funcția de evaluare, operatorii, procesul de selecție, rata mutației, mărimea populației şi aşa mai departe)?
- Cum se realizează schimbarea (adică în mod euristic determinist, euristic bazat pe feedback sau auto-adaptativ)?
- Dovada pe baza căreia se realizează schimbarea (ex. monitorizarea performanței operatorilor, diversitatea populației şi aşa mai departe)?

Pentru a clasifica tehnicile de control al parametrilor, din perspectiva a ce componente sau parametri se schimbă [Siarry&Michalewicz2008], este necesar să se convină asupra unei liste cu toate componentele principale ale algoritmului evolutiv lucru care este dificil în sine: reprezentarea indivizilor, evaluarea funcțiilor, variația operatorilor şi a probabilităților lor, selecția operatorilor (selecția părinților sau selecția reproducerii), operatorul de înlocuire (selectare în funcție de supraviețuire sau de mediu), populația (mărime, tipologie etc.).

Metodele de schimbare a valorii unui parametru (adică aspectul Cum) se pot clasifica]n [Siarry&Michalewicz2008]: tuning al parametrului şi control al parametrului. Prin tuning înțelegem abordarea practicată în mod obişnuit prin care se evaluează valorile bune pentru un parametru *înainte* de aplicarea algoritmului şi apoi se aplică algoritmul folosind acele valori, care rămân fixe în timpul aplicării. Controlul parametrilor formează o alternativă, care permite să se înceapă cu o aplicare cu valorile inițiale ale parametrului, care se schimbă în timpul aplicării. Controlul parametrilor poate fi mai departe încadrat în una din următoarele trei categorii

[Siarry&Michalewicz2008]: determinist, adaptativ şi auto-adaptativ. Această terminologie duce la taxonomia ilustrată în Figura 3.1.

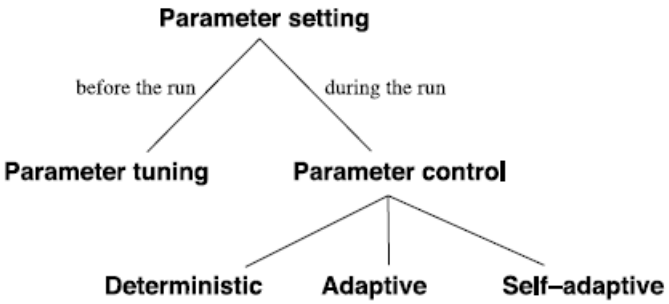


Figura 3.1 Taxonomie globală a stabilirii parametrilor în AE [Siarry&Michalewicz2008]

3.3. Calculul evolutiv vs. optimizarea clasică

Algoritmii de optimizare clasică s-au dovedit a fi de mare succes (şi mai eficienți decât AE) în probleme liniare, quadratice, puternic convexe, unimodale şi alte probleme specializate, însă algoritmii evolutivi s-au dovedit a fi mai eficienți pentru problemele discontinue, nediferențiabile, multimodale şi noisy. CE şi optimizarea clasică (OC) se diferențiază mai ales în procesul de căutare şi în ceea ce priveşte informația despre spațiul de căutare folosit pentru a ghida procesul de căutare [Engelbrecht2002]:

- Procesul de căutare: OC utilizează reguli deterministe pentru a se deplasa de la un punct din spațiul de căutare la următorul punct. CE, pe de altă parte, utilizează reguli de tranziție probabilistice. De asemenea, CE aplică o căutare paralelă a spațiului de căutare, în timp ce OC utilizează căutarea secvențială. O căutare a AE porneşte de la un set de puncte inițiale diverse, ceea ce permite căutare paralelă într-o zonă mare a spațiului de căutare. OC porneşte dintr-un punct ajustând succesiv acest punct pentru a se deplasa către punctul optim.
- Informații despre suprafața de căutare: OC utilizează informații despre derivate, de obicei de primul sau al doilea ordin, din spațiul de căutare pentru a-şi ghida calea către punctul optim. CE, pe de altă parte, nu foloseşte informația despre derivate. Valorile fitness ale indivizilor sunt utilizate pentru a ghida căutarea.

Conform teoremei no-free-lunch (NFL) [Wolpert&Macready 1996] nu poate exista nici un algoritm care să rezolve toate problemele (de optimizare) şi care să fie în general (în medie) superior oricărui competitor, aşadar întrebarea dacă AE sunt inferiori sau superiori altor abordări este lipsită de sens. Singurul lucru care se poate afirma este că AE se comportă mai bine decât alte metode în ceea ce priveşte rezolvarea anumitor clase de probleme – cu consecința că se comportă mai rău în cazul altor clase de probleme [DeJong1997, Baeck. et. al 2000].

Partea B – Contribuții

4. Problemă de alocare pe maşini paralele şi uniforme

Scopul acestui capitol este să prezinte un nou algoritm genetic pentru o prblema de alocare pe maşini paralele şi uniforma. Algoritmul propus nu doar că obține rezultate mai bune dar şi calculează rezultatul mai rapid..

Maşinile paralele şi uniforme reprezintă o clasă specială de resurse [Blazewicz et. al. 2007] în care maşinile au viteze diferite însă viteza este constantă şi nu depinde de sarcină. Deoarece problema s-a dovedit a fi dificilă din punct de vedere al timpului nedeterminist polinomial [Garey&Johnson1979] am propus un nou algoritm genetic (GASP) pentru o găsi o soluție la această problemă [Mihăilă&Mihăilă2008a]. Se raportează o serie de rezultate iar performanța abordării prezentate este comparată cu alte tehnici de optimizare. Rezultatele empirice indică faptul că GASP este mai eficient [Mihăilă&Mihăilă2008b].

4.1. Introducere

Se ştie că alocarea de sarcini şi resurse este dificilă din punct de vedere al timpului nedeterminist polinomial. De aceea utilizarea euristicii este abordarea de-facto pentru a face față acestei dificultăți în practică. Pe lângă abordările euristice precum căutarea locală [Ritchie&Levine2003], călirea simulată [Abraham et. al. 2000] [Yarkhan&Dongarra2002], căutarea tabu [Abraham et. al. 2000] şi algoritmii genetici [Abraham et. al. 2000][Zomaya&The2001] au fost utilizați în problemele de alocare. Ritchie şi Levine [Ritchie&Levine2004] au combinat un algoritm de optimizare al unei colonii de furnici cu un algoritm de căutare tabu pentru o problemă de alocare în timp ce Ye et al. [Guangchang et. al. 2006] au formulat o abordare de optimizare multi-obiectiv pentru a optimiza simultan timpul de terminare şi costul total al execuției. Alte abordări ale acestei probleme includ optimizare folosind particule swarm [Abraham et. al. 2006], alocare bazată pe teoria fuzzy [Kumar et. al. 2004] şi abordări economice [Buyya et. al. 2000].

4.2. O problema de alocare pe maşini paralele şi uniforme

Teoretic, problema de alocare poate fi descrisă în felul următor: *n* sarcini independente $T = \{T_1, T_2, ..., T_n\}$ trebuie să fie alocate unor *m* maşini paralele şi uniforma $M = \{M_1, M_2, ..., M_m\}$ având în vedere obiectivul de a minimiza timpul de terminare utilizând resursele în mod eficient. Viteza fiecărei maşini este exprimată în numărul de cicluri pe unitate de timp iar lungimea fiecărei sarcini este exprimată în n umărul de cicluri. Fiecare sarcină T_i are nişte cerinţe de procesare de P_i cicluri iar maşinile M_k au viteza de S_k cicluri/secundă. Fiecare sarcină T_i trebuie să fie procesată de maşina M_k, până la completare [Grosan et. al. 2007].

Obiectivul problemei noastre de programare este să minimizeze timpul de terminare al alocarii (makespan).

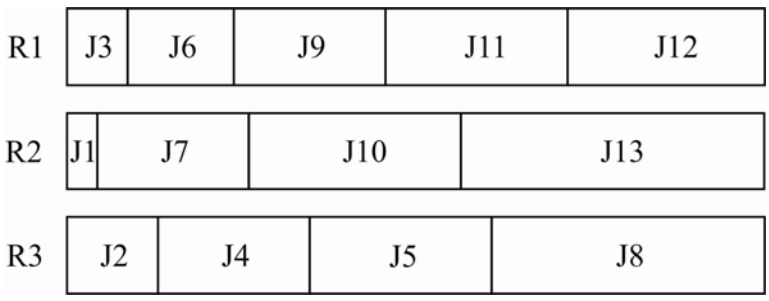
4.3. Algoritmul genetic propus

Algoritmul începe cu o populație de cromozomi generați în mod arbitrar (potențiale soluții). Pentru a genera o nouă populație aplicăm următorii operatori genetici [Baeck et. al. 2000]: selecție turnir bina, pentru selectarea părinților şi încrucişare într-un punct de tăietura şi mutație a genelor pentru generarea noilor cromozomi fii care vor constitui noua populație. Procesul de evoluție este similar cu schema evolutivă a unui algoritm genetic standard. Am utilizat în plus o selecție elitistă.

Soluția problemei de alocare este reprezentată sub forma unui şir de lungime egală cu numărul de sarcini. Valoarea care corespunde fiecărei poziții *i* din şir reprezintă maşina căreia sarcina *i* a fost alocată. Dacă luăm cazul a 13 sarcini şi 3 maşini atunci un cromozom al atribuirii de sarcini poate fi reprezentat după cum urmează:

2	3	1	3	3	1	2	3	1	2	1	1	2

Diagrama Gantt care corespunde acestei codificări este:



4.4. Rezultate experimentale

Pentru a testa algoritmul propus (GASP) s-au efectuat o serie de experimente folosind patru instanțe de testare şi s-aucomparat rezultatele obținute cu alte tehnici de optimizare: Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Multi-Objective Evolutionary Algorithm (MOEA).

Fiecare experiment a fost repetat de 10 ori şi fiecare testare a fost realizată având următorii parametrii stabiliți: dimensiunea populației 20; numărul de iterări 50 * *m* * *n*; probabilitatea de încrucişare 0.45 (pentru instanța 1) şi 0.18 (pentru instanțele 2, 3, 4 şi 5); probabilitatea de mutației 0.35 (pentru instanța 1) şi 0.02 (pentru instanțele 2, 3, 4 and 5).

Makespan-ul mediu şi deviațiile standard raportate pentru 10 încercări sunt ilustrate în Tabelul 4.1.

Tabelul 4.1. Makespan-ul mediu şi deviația standard

Instanța	Rezultatul optim	Makespan-ul mediu	Deviația standard
1	46	46	0
2	85.5279	85.5431	0.009
3	41.5788	41.7395	0.0856
4	35.1303	35.3785	0.0477
5	59.1658	59.3041	0.0461

Figura 4.1 ilustrează, pentru instanțele 1, 2, 3, 4 şi 5, rezultatele (makespan) optime, cele mai bune şi cele medii obținute de GASP.

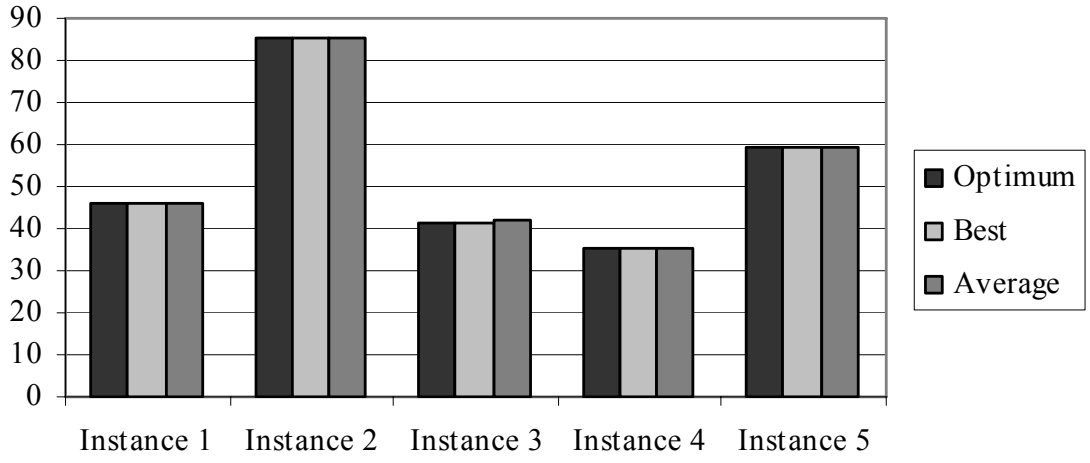


Figura 4.1. Rezultatele cele mai bune şi cele medii obținute de GASP pentru instanțele 1, 2, 3, 4 şi 5.

Am comparat rezultatele obținute de algoritmul propus pentru problema de alocare (GASP) cu alte tehnici utilizate pentru optimizarea alocărilor de sarcini şi resurse: Genetic

Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

Valorile makespan medii [Abraham et. al. 2008] şi deviațiile standard raportate pentru cele 10 încercări sunt ilustrate în Tabelul 4.2, unde *am* reprezintă makespan mediu iar *sd* reprezintă deviația standard (pentru instanța 2 a PSO considerăm că este vorba de o eroare de scriere). Rezultatele obținute de algoritmii luați în considerare pentru comparație au fost preluate din [Abraham et. al. 2006][Abraham et. al. 2008]. După cum putem observa din Tabelul 4.2 algoritmul GASP a oferit cele mai bune rezultate la toate instanțele alese. Am realizat un alt experiment pentru a vedea dacă performanța rămâne valabilă şi în cazul reducerii numărului de iterații. Am testat algoritmul pentru 25**m***n*, GASP (2), şi pentru *m***n*, GASP (3), iterații. Celelalte setări ale parametrilor au rămas neschimbate.

Tabelul 4.2. Comparație a performanței.

Instanța		1	2	3	4	5
Optimum		**46**	**85.5279**	**41.5788**	**35.1303**	**59.1658**
GA	*Am*	47.1167	85.7431	42.927	38.0428	
	Sd	±0.7700	±0.6217	±0.415	±0.6613	
SA	*Am*	46.6	90.7338	55.4594	41.7889	
	Sd	±0.4856	±6.3833	±2.0605	±8.0773	
PSO	*Am*	46.2667	84.0544	41.9489	37.6668	
	Sd	±0.2854	±0.5030	±0.6944	±0.6068	
ACO	*Am*	46.2667	88.1575			
	Sd	±0.2854	±0.6423			
GASP	*Am*	**46**	**85.5431**	**41.7395**	**35.3785**	**59.3041**
(1)	*Sd*	±0	±0.0090	±0.0856	±0.0477	±0.0461
GASP	*Am*	**46.05**	**85.5595**	**41.8042**	**35.6098**	59.3625
(2)	*Sd*	±0.15	±0.0092	±0.0775	±0.1398	±0.0716
GASP	*Am*	46.5667	**85.681**	42.3229	**36.455**	59.7058
(3)	*Sd*	±0.3512	±0.0803	±0.3036	±0.6077	±0.1683

După cum se poate observa în Tabelul 4.2, performanța algoritmului GASP rămâne valabilă chiar dacă înjumătățim numărul de iterații. Aceasta înseamnă că algoritmul GASP a obținut rezultate foarte bune mult mai repede decât ceilalți algoritmi luați în considerare. În cazul unui număr de *m***n* iterații, algoritmul GASP a dat rezultate foarte bune în comparație cu alte tehnici, luând în considerare faptul că alte tehnici au folosit un număr de 50**m***n* iterații.

De asemenea am comparat algoritmul nostru (GASP) cu Multi-Objective Evolutionary Algorithm (MOEA) [Abraham et. al. 2008]. Rezultatul mediu (makespan) pentru zece încercări este prezentat în Tabelul 4.3. Rezultatul pentru MOEA a fost preluat din [Abraham et. al. 2008].

Tabelul 4.3. Comparație a performanței cu MOEA

Instanța		1	2	3	4	5
Optimum		**46**	**85.53**	**41.58**	**35.13**	**59.17**
MOEA	*am*	**46**			36.68	
GASP (1)	*am*	**46**	85.55	41.74	**35.38**	**59.37**
	sd	0	0.0084	0.055	0.067	0.0659
GASP (2)	*am*	46.15	85.57	41.8	**35.67**	**59.51**
	sd	0.2291	0.0108	0.074	0.2011	0.1328
GASP (3)	*am*	46.48	85.65	42.27	**35.94**	**59.84**
	sd	0.3609	0.0484	0.29	0.3032	0.2112

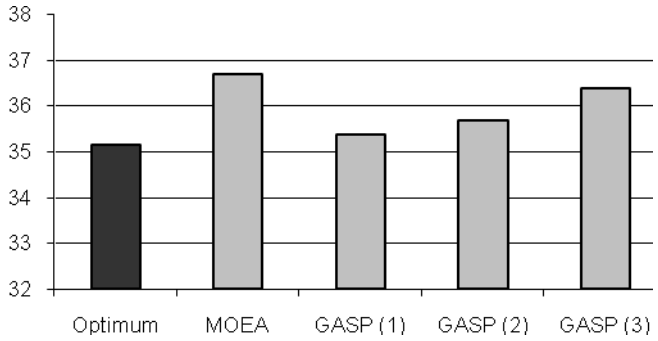


Figura 4.2 Makespan-ul mediu al MOEA şi GASP pentru instanța 4.

După cum se poate observa din Tabelul 4.3 GASP a dat rezultate bune chiar şi atunci când mărimea populației şi numărul de iterații s-au înjumătățit. GASP (3) a dat de asemenea rezultate bune ținând cont de valoarea mărimii populației şi cea a numărului de iterații. Menționăm că atât GASP (2) cât şi GASP (3) au găsit valoarea optimă pentru instanța 1. Figura 4.2 ilustrează makespan mediu, pentru instanța 4, a MOEA şi a GASP (cu 3 setări de parametri) la 10 testări.

4.5. Concluzii şi direcții viitoare de cercetare

Din datele raportate mai sus putem concluziona că GASP a dat rezultate excelente în comparația cu alte tehnici. Chiar dacă abordarea GASP a obținut rezultate mai bune pentru problemele de testare luate în calcul, în comparație cu rezultatele obținute de alte tehnici de optimizare, se va putea ajunge la mai multe concluzii doar după o validare amplă utilizând probleme mai mari.

Planul nostru viitor de cercetare este să extindem abordarea problemelor de alocare implicând maşini/procesoare paralele nerelaționate şi mani/procesoare dedicate şi să testăm algoritmii propuşi cu date reale.

5. Problemă de alocare de tip permutation flow shop

Scopul acestui capitol este prezinte un nou algoritm genetic hibrid pentru o problemă de alocare de tip permutation flow shop. Noutatea algoritmului propus constă în utilizarea unei combinații între o procedură de inițializare arbitrară şi o procedură de inițializare bazată pe euristica constructivă NEH, în definirea unui nou operator de încrucişare şi în utilizarea unui operator de mutație definit ca o combinație între un operator bazat pe euristica constructivă NEH şi mutația prin translatare. Rezultatele obținute de către algoritmul propus sunt comparabile cu rezultatele obținute de un algoritm greedy iterativ.

Obiectivul unei probleme de alocare de tip permutation flow shop este să găsească o secvență de procesare a sarcinilor care să minimizeze un criteriu dat ştiind că toate sarcinile au aceeaşi ordine de procesare de către maşini. Deoarece problema este dificilă din punct de vedere al timpului nedeterminist polinomial s-au propus multe metode euristice şi meta-euristice. Algoritmul propus [Mihăilă et.al.2008b] utilizează euristica constructivă NEH pentru a genera un procent predefinit de cromozomi ai populației inițiale cu scopul de a creşte şi de a grăbi şansele de găsire a unei soluții bune. Euristica constructivă NEH este de asemenea utilizată de către operatorul de mutație pentru a îmbunătăți cromozomii obținuți. Rezultatele obținute de către algoritmul genetic propus sunt comparate cu cele mai bune rezultate raportate de un algoritm greedy iterativ.

5.1. Introducere

Problema de alocare de tip permutation flow shop a fost studiată pentru prima oară de Johnson în 1954 [Johnson1954] iar de atunci s-au propus multe metode euristice şi meta-euristice [Ruiz&Stützle2007]. Metodele euristice variază de la euristica constructivă cum ar fi Rapid Access [Dannenbring1977] sau NEH [Nawaz et. al. 1983] până la euristica îmbunătățită precum RACS şi RAES [Dannenbring1977] sau cea propusă de Suliman [Suliman2000]. Metodele meta-euristice pot fi de asemenea privite ca metode de euristică îmbunătățită. Aceste metode variază de la acela care îmbunătățesc un program dat, cum ar fi căutarea tabu

[Grabowski&Wodecki2004], [Nowicki&Smutnicki1996], [Taillard1990] călirea simulată [Osman&Potts1989] sau greedy iterativ [Ruiz&Stützle2007], până la cele care lucrează cu o colecție de alocări precum algoritmii genetici [Chen et. al. 1995] [Murata et. al. 1996], [Reeves&Yamada1998], [Ruiz et. al. 2004] sau coloniile de furnici [Rajendran&Ziegler2004].

5.2. O problemă de alocare de tip permutation flow shop

După cum am menționat deja, obiectivul unei probleme de alocare de tip permutation flow shop este să găsească o secvență pentru procesarea unui set de n sarcini $J = \{J_1, ..., J_n\}$ pe un set de m procesoare sau mașini, $P = \{P_1, ... , P_m\}$ astfel încât un anumit criteriu să fie optimizat. Criteriul luat în calcul în acest capitol este timpul total de stagnare acumulat pe ultima mașină iar obiectivul este ca acest criteriu să se minimizeze.

Fiecare activitate J_i, $i = 1, ..., n$, este compusă dintr-un set de m sarcini și fiecare sarcină k, $k = 1, ..., m$, trebuie să fie executată de o altă mașină, adică, pentru a fi completă, o activitate trebuie să fie procesată de către fiecare mașină. Timpul de procesare al sarcinii k din activitatea J_i este descrisă de $p_{i,j}$.

În problema de alocare de tip permutation flow shop toate activitățile au aceeași ordine de procesare pe mașini și de aceea, odată ce o succesiune de activități este stabilită pe primele mașini, aceasta va fi menținută pe toate celelalte mașini [Blazewicz et. al. 2007]. În timp ce succesiunea activităților pentru toate mașinile este aceeași, problema este să găsim succesiune a de activități care va minimiza criteriul dat [Blazewicz et. al. 2007], în cazul nostru timpul total de stagnare acumulat pe ultima mașină.

Problema de alocare de tip permutation flow shop, ca un caz particular de problemă de alocare de tip flow shop, îndeplinește o serie de constrângeri precum [Ruiz&Maroto2004]: fiecare mașină poate gestiona doar o activitate odată [Blazewicz et. al. 2007]; fiecare activitate poate fi realizată doar pe o mașină odată [Blazewicz et. al. 2007]; nu există constrângeri legate de precedență în cadrul sarcinilor diferitelor activități [Blazewicz et. al. 2007]; toate activitățile sunt disponibile spre procesare la momentul 0 [Ruiz&Maroto2004]; momentele (intervalele) de set-up al activităților pe mașini sunt neglijabile și de aceea pot fi ignorate [Ruiz&Maroto2004]; nu se permite întreruperea adică procesarea unei acțiuni pe o mașină nu poate fi întreruptă [Ruiz&Maroto2004]; mașinile sunt disponibile permanent [Ruiz&Maroto2004]; este permis să se facă un inventar în timpul procesării adică dacă următoare mașină la rând este necesară unei activități iar ea nu este disponibilă, activitatea poate aștepta la rând pentru acea mașină [Ruiz&Maroto2004].

În acest capitol s-a luat în considerare o problemă de alocare de tip permutation flow shop scheduling care respectă aceste presupuneri. Având în vedere că se știe că problema este grea din punct de vedere al timpului nedeterminist polinomial [Garey&Johnson1979] s-a ales pentru rezolvarea ei o metodă meta-euristică.

5.3. Algoritmul genetic propus

Pentru a găsi o soluție a problemei de alocare de tip permutation flow shop, descrisă în secțiunea precedentă, am utilizat un algoritm genetic generațional.

Algoritmii genetici utilizează o populație (colecție) de cromozomi (posibile soluții codate) care evoluează, prin operații genetice, într-o nouă populație. Acest proces de evoluție este ghidat de funcția fitness care măsoară cât de buni sunt cromozomii şi care se repetă de un număr predefinit de ori până când este satisfăcut un criteriu de oprire dat. Cel mai bun cromozom din ultima populație se raportează ca rezultat al algoritmului. De obicei populația inițială este generată în mod arbitrar.

Soluția pentru problema de alocare de tip permutation flow shop este codificată ca un şir de permutații ale activităților. Astfel, cromozomul are o lungime egală cu numărul de activități iar valoarea (gena) corespondentă fiecărei poziții din şir reprezintă o activitate. Ordinea relativă a activităților în permutare indică ordinea de procesare a activităților de către maşini [Ruiz et. al. 2004].

Pentru a evalua calitatea unui cromozom folosim ca funcție fitness timpul total de stagnare acumulat la ultima maşină.

Populația inițială a fost generată utilizând o combinație între o procedură de inițializare arbitrară şi o procedură de inițializare bazată pe euristica construcitvă NEH [Nawaz et. al. 1983]. Alocarea obținută cu ajutorul euristicii constructive NEH a fost introdusă în populația inițială şi s-a păstrat de asemenea şi ca bază, numit de acum în acolo *init,* pentru construirea de alte alocări. Procedura de inițializare bazată pe euristica construcitvă NEH utilizează ideea de faze de distrugere-construire introduse de [Ruiz&Stützle2007] pentru un algoritm iterated greedy.

Pentru a genera o nouă alocare (cromozom), un anumit procent de activități sunt extrase în mod arbitrar din baza *init* şi reinserate folosind euristica construcitvă NEH. Noua alocare obținută se introduce în populația inițială şi se reactualizează baza *init* la valoarea noii alocări construite. Procedura de inițializare arbitrară utilizează de asemenea baza *init* pentru a genera o nouă alocare dar nu îi reactualizează valoarea. Pentru a genera o nouă alocare (cromozom) secționăm mai întâi baza *init* conform unui şablon generat arbitrar şi schimbăm cele două segmente rezultate între ele, după care executăm un număr de *n* schimburi între cele două activități stabilite arbitrar. Populația inițială este formată în proporție de 70% din indivizi creați prin utilizarea procedurii de inițializare arbitrară în timp ce 30% din indivizi sunt generați folosind procedura de inițializare bazată pe NEH.

În vederea selectării cromozomilor pentru operatorul de încrucişare am utilizat selecția turnir binar [Back et. al. 2000] care constă în alegerea arbitrară a doi cromozomi din populația curentă şi selectarea celui mai bun.

În plus, am folosit o selecție elitistă [Back et. al. 2000] pentru a preveni pierderea celor mai buni cromozomi obținuți până acum. În acest sens 20% din cei mai buni cromozomi din populația curentă au fost copiați în următoarea populație (cea nouă).

Operatorul de încrucişare utilizat pentru a produce noi cromozomi poate fi descris în felul următor:

- se generează arbitrar un punct de tăietură

- se copiază, cu o probabilitate dată, primul sau ultimul segment din cromozomii părinților în cromozomii fii, adică dacă primul segment al cromozomului primului părinte este copiat ca prim segment în cromozomul primului fiu atunci primul segment al cromozomului celui de-al doilea părinte va fi copiat ca prim segment în cromozomul celui de-al doilea fiu (aceeaşi tactică se aplică şi în cazul ultimului segment)
- se completează părțile care rămân din cromozomii fii, începând de la dreapta după segmentul copiat anterior până la capăt, cu genele lipsă din cromozomul părintelui opus, adică cromozomul primului fiu va prelua genele lipsă din cromozomul celui de-al doilea părinte în timp ce cromozomul celui de-al doilea fiu va fi completat cu gene din cromozomul primului părinte. Dacă primul segment a fost copiat înainte, atunci cromozomii părintelui vor fi transferați de la punctul de secționare spre dreapta şi când se ajunge la capătul cromozomului, ordinea de transfer va începe cu prima genă şi va continua până când se găseşte punctul de secționare. Dacă ultimul segment a fost copiat înainte, atunci politica de transfer se inversează.

Ca operator de mutație am utilizat o combinație între un operator bazat pe NEH euristica constructivă NEH, numit de acum încolo mutație NEH, care este similar cu procedura de inițializare bazată pe euristica constructivă NEH şi mutația prin translatare (shift mutation) [Ruiz et. al. 2004] care constă în selectarea arbitrară a poziției cromozomilor şi relocarea genei (activității) corespondentă poziției alese într-o altă poziție aleasă arbitrar în timp ce genele (activitățile) dintre aceste două poziții se deplasează şi ele. Mutația NEH are ca scop îmbunătățirea cromozomilor obținuți de către operatorul de încrucişare în timp ce mutația prin translatare introduce noi cromozomi pentru a reduce pierderea în diversitate a populației.

Luând în considerare sugestiile din [Ruiz et. al. 2004] am modificat operatorul de supraviețuire al algoritmului genetic generațional în sensul că am inserat în populația următoare (nouă) doar cromozomi distincți pentru a limita efectul de convergență prematură şi pentru a creşte diversitatea populației.

5.4. Rezultate experimentale

Pentru a testa performanța algoritmului nostru am utilizat un set de date standard [Taillard1993] din care am ales 10 instanțe cu 50 de activități şi 20 de maşini şi 10 instanțe cu 100 de activități şi 20 maşini. Aceste instanțe au fost alese deoarece s-a dovedit că unele din aceste instanțe sunt foarte greu de rezolvat [Ruiz&Stützle2007].

Fiecare experiment a fost repetat de 10 ori. Setările specifice ale parametrilor pentru fiecare experiment sunt descrise în Tabelul 5.1.

Tabelul 5.1 Setările parametrilor

Parametru	**Valoare**
Dimensiunea populației	100
Numărul de generații	1000
Probabilitatea de încrucişare	0.5
Probabilitatea de mutație	0.1

Rezultatele cele mai bune, cele medii şi limitele inferioare şi superioare indică timpul total de realizare a unei program, numit şi makespan. Pentru a determina makespan-ul rezultatului nostru am adăugat la timpul total de stagnare acumulat la ultima maşină (calculat în funcție de funcția fitness) timpul total de execuție a ultimii maşini.

Figura 5.1 şi Figura 5.2 reprezintă grafic, pentru fiecare instanță luată în considerare, rezultatele cele mai bune şi cele medii comparate cu cea mai bună limită superioară cunoscută.

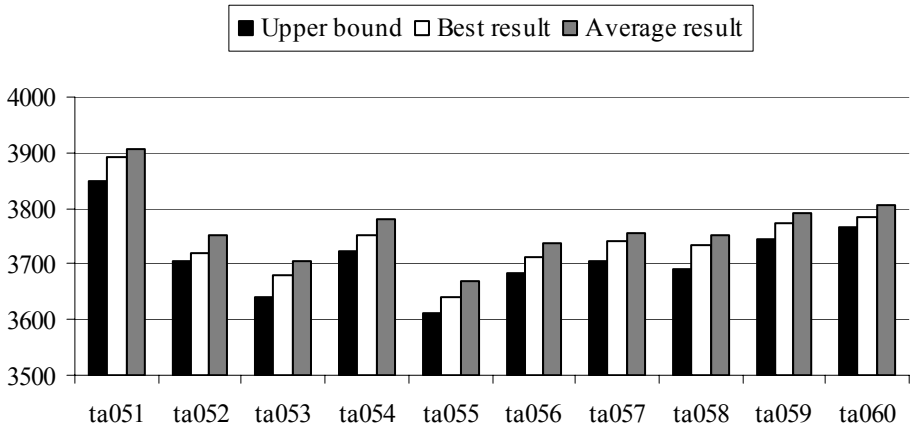


Figura 5.1 Rezultatele cele mai bune şi rezultatele medii pentru instanțele ta051-ta060

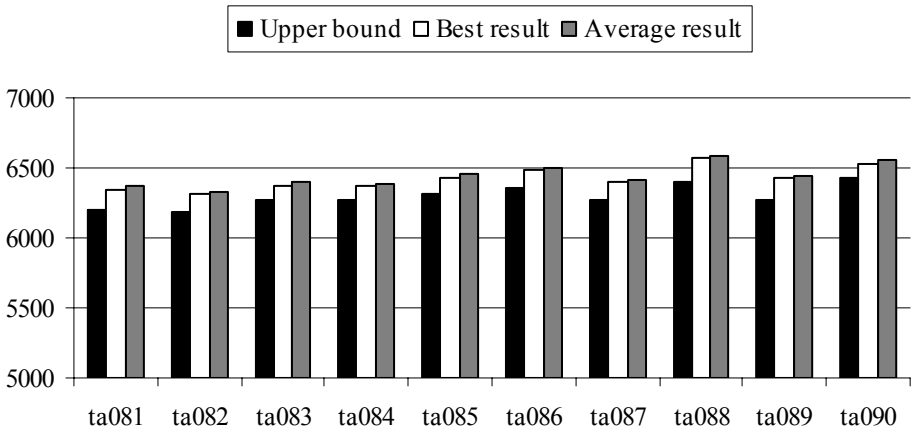


Figura 5.2 Rezultatele cele mai bune şi rezultatele medii pentru instanțele ta081-ta090

După cum se poate observa din Figura 5.1, rezultatele obținute au fost foarte aproape de limitele superioare. Diferența dintre rezultatele cele mai bune şi limitele superioare se încadrează între 0.43% şi 1.16% în timp ce diferența dintre rezultatele medii şi limita superioară se încadrează între 1.05% to 1.74% pentru instanțele ta051-ta060.

Rezultatele obținute au fost comparate cu rezultatele obținute de un algoritm greedy iterativ cu căutare locală [Ruiz&Stützle2007]. Cel mai bun rezultat al algoritmului greedy iterativa fost preluat din [Ruiz&Stützle2007]. Comparația dintre rezultatele cele mai bune este ilustrată în Figura 5.3 şi Figura 5.4.

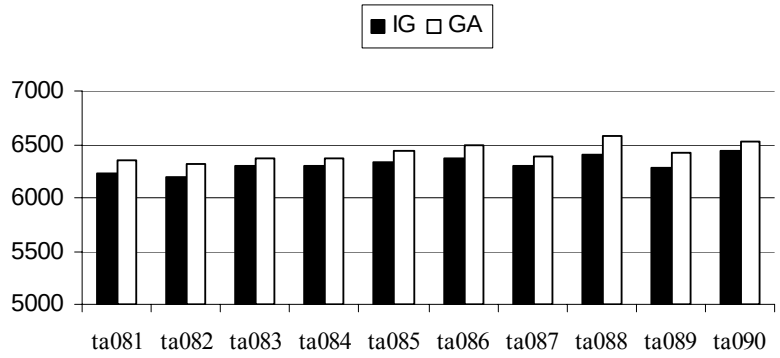


Figura 5.3 Comparație între rezultatele cele mai bune pentru instanțele ta051-ta060

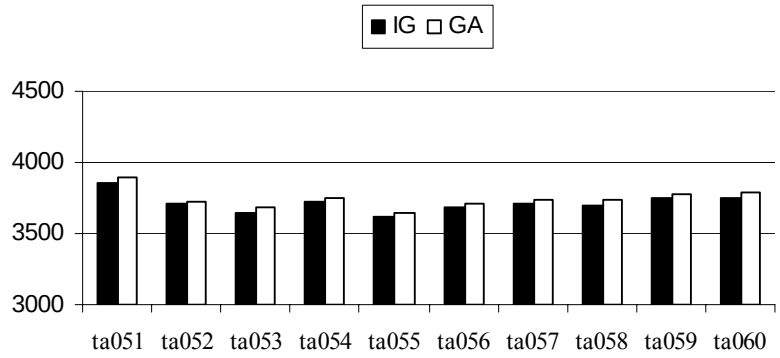


Figura 5.4 Comparație între rezultatele cele mai bune pentru instanțele ta081-ta090

După cum se poate observa din Figura 5.3 şi Figura 5.4, rezultatele obținute de algoritmul genetic propus sunt foarte apropiate de rezultatele obținute de algoritmul greedy iterativ. Diferența dintre cele mai bune rezultate ale algoritmului greedy iterativ şi cele mai bune rezultate ale algoritmului genetic se încadrează între 0.40% şi 1.08% pentru instanțele ta051-ta060 şi 1.05% şi 2.69% pentru instanțele ta081-ta090.

5.5. Concluzii şi direcții viitoare de cercetare

În acest capitol a fost prezentat un algoritm genetic pentru rezolvarea unei probleme de tip permutation flow shop. Rezultatele obținute de către algoritmul propus sunt asemănătoare cu rezultatele raportate de alți algoritmi.

În viitor, vor fi luate în considerare alte instanțe de testare din setul standard (ex. 100 de activități şi 20 de maşini, 200 de activități şi 20 de maşini) pentru a testa performanța algoritmului propus şi se va investiga influența diferiților operatori genetici asupra calității soluției.

6. Problemă de alocare a unui video-proxy cache

Scopul acestui capitol este să prezinte două noi moduri de definire a utilității obiectelor stocate într-un video proxy-cache precum şi să prezinte un nou algoritm genetic pentru determinarea coeficienților care apar în aceste două definiții cu scopul de a maximiza byte hit rate. Rezultatul obținut de către algoritmul propus cu o funcție de utilitate este similar sau chiar mai bun decât rezultatul obținut de o altă metrică.

Pentru problema de alocare a unui video-proxy cache scheduling problem s-au introdus două noi funcții de utilitate care iau în considerare utilitatea obiectelor atunci când realizează operațiuni de actualizare a cache-ului. Coeficienții care rafinează aceste funcții sunt determinați folosind un algoritm genetic [Mihăilă&Cobârzan2008]. S-au realizat măsurători cu privire la eficiența în termeni de byte hit rate când se folosesc funcțiile de utilitate iar rezultatele obținute sunt comparate cu cele produse de algoritmi clasici de înlocuire a cache-ului [Cobârzan&Mihăilă2008].

6.1. Introducere

Volumul de materiale multimedia şi în special conținutul video de pe Internet a crescut constant în ultimii ani. Având în vedere caracteristicile acestui tip de date (dimensiuni mari, limite acceptate ale latenței în timpul derulării etc.) se pune mare presiune asupra infrastructurii de transport, care de cele mai multe ori este Internetul. Un proxy-cache este o entitate care acționează ca un intermediar într-o tranzacție în care un client solicită un obiect multimedia/video. Într-un astfel de caz proxy-cache acționând din partea clientului, începe să recupereze obiectul de pe un server de origine şi îl transferă (streaming) către client. În cazul în care un cahe este de asemenea activ, conținutul care se transferă, sau părți din acesta, poate fi salvat local.

Există numeroase abordări pentru video caching: caching-ul unui prefix în [Sen et. al. 1999], caching-ul unui prefix şi a unui set de cadre selectate în [HsiuMa&Du2000], caching-ul unui prefix combinat cu transmisie periodică în [Yang& Towsley2002] sau caching-ul unor

segmente hotspot în [Fabmi et. al. 2001]. Alte propuneri din aceeaşi categorie includ cahing-ul unui prefix bazat pe popularitate [Park et. al. 2001], caching cu prefix bazat pe segment [Wu et. al. 2001], şi variable sized chunk caching [Balafoutis et. al. 2002]. Au avut loc şi diverse abordări ale acestei probleme: servere video multiple accesibile via un sistem terțiar de stocare care gestionează web-ul [Brubeck&Brubeck1996] sau cooperative caching video server [Acharya2002] .

6.2. Sistemul distribuit de video-proxy cache prpous

În [Cobârzan2005] a fost introdus un sistem distribuit pentru video proxy-caching care este capabil să ajusteze dinamic numărul de noduri care participă la sistemul de cache federativ în funcție de patternul solicitării clientului şi de încărcătura curentă. În această propunere de sistem operațiile de înlocuire a cache-ului sunt realizate ținând cont de valoarea utilității obiectelor, însemnând că obiectele cu utilitatea cea mai mică vor fi eliminate când trebuie să se facă loc pentru a primi noi obiecte. Utilitatea unui obiect se calculează folosind o funcție u definită în [Cobârzan2005] după cum urmează $u : LC \rightarrow R$ (LC reprezintă conținutul cache-ului local).

$$u(o) = coef_1 * size(o) + coef_2 * \frac{1}{timeLastAccess(o)} + coef_3 * hitCount(o) + coef_4 * qualityValue(o), \tag{6.1}$$

unde:

- $size(o)$ este mărimea obiectului,
- $timeLastAccess(o)$ indică ultima dată când a fost solicitat obiectul,
- $hitCount(o)$ arată de câte ori a fost servit obiectul din cache
- $qualityValue(o) \in [0..1]$ este măsura calității obiectului
- $coef_1, coef_2, coef_3, coef_4 \in [0, 1]$ şi $coef_1 + coef_2 + coef_3 + coef_4 = 1$.

În timpul evaluării inițiale a performanței sistemului am observat că dacă folosim formula aşa cum este definită în [Cobârzan2005] impactul *timeLastAccess* şi *hitCount* este neglijabil datorită diferenței de ordin al magnitudinii atunci când se compară cu *size* şi *hitCount*. Pentru a corecta aspectele negative observate, am introdus două noi moduri de definire a utilității unui obiect:

$$\begin{aligned} u(o) = coef_1 * \frac{size(o)}{MSIZE} + coef_2 * \frac{timeLastAccess(o)}{MTLA} + \\ + coef_3 * \frac{hitCount(o)}{MHC} + coef_4 * qualityValue(o) \end{aligned} \tag{6.2}$$

şi

$$u(o) = -\left[coef_1 * \frac{1}{size(o)} + coef_2 * \frac{1}{\sqrt[4]{timeLastAccess(o)}} + coef_3 * \frac{1}{hitCount(o)} + coef_4 * \frac{1}{qualityValue(o)} \right] \quad (6.3)$$

unde:

- *MSIZE* – mărimea celui mai mare obiect stocat în cache-ul local *LC* până la acel moment;
- *MTLA* – momentul în care s-a făcut ultima dată o solicitare a unui obiect din *LC*;
- *MHC* – de câte ori a fost solicitat cel mai popular obiect din *LC* .

În plus, $coef_1, coef_2, coef_3, coef_4 \in [0, 1]$ precum şi $coef_1 + coef_2 + coef_3 + coef_4 = 1$ în continuare trebuie să fie valabile pentru ambele ecuaţii (6.2) şi (6.3). Trebuie menţionat de asemenea că formulele (6.2) şi (6.3) ar putea necesita o rafinare (tuning) ulterioară deoarece nu am luat în considerare valoarea calităţii obiectelor în timpul măsurătorilor noastre.

În [Cobârzan2005] am luat în considerare nişte valori fixe ale $coef_1$ până la $coef_4$. Scopul nostru este să putem oferi o metodă inteligentă de determinare a acelor coeficienţi astfel încât diferite metrics să fie maximizate/minimizate (byte hit ratio, object hit ratio/latency). Aceasta a dus la abordarea noastră actuală de a folosi algoritmii genetici. Ca prim pas ne-am axat pe alegerea celor patru valori $coef_1$ până la $coef_4$ în formulele de utilitate (6.2) şi (6.3) pe baza patternurilor solicitărilor clientului (numărul de solicitări a fiecărui obiect, momentul în care este solicitat) precum şi pe baza caracteristicilor obiectelor (mărime, distribuţie) astfel încât *byte hit ratio* să fie maximală.

6.3. Algoritmul genetic propus

Pentru a găsi valori „bune ” ale coeficienţilor pentru funcţiile de utilitate definite în (6.2) şi (6.3) am utilizat un algoritm genetic (GeCo) care porneşte de la o populaţie de soluţii (cromozomi) generate arbitrar. O nouă populaţie este generată aplicând următorii operatori genetici [Baeck et. al. 2000]: selecţia turnir binar pentru selectarea părinţilor şi încrucişare convexă şi mutaţie bazată pe rază pentru generarea de noi cromozomi fii care vor constitui noua populaţie. Procesul de evoluţie este similar cu schema evolutivă a algoritmilor genetici standard . Am folosit de asemenea un proces de selecţie elitist.

Soluţia este reprezentată ca un şir (cromozomi) de lungime egală cu numărul de coeficienţi minus 1. Am luat în considerare doar primii trei coeficienţi lăsându-l la o parte pe ultimul care are valoarea setată la 0. Pentru problema noastră aceasta înseamnă că nu ne interesează calitatea obiectelor solicitate/recuperate. Utilizarea celui dea-al patrulea coeficient are sens doar dacă operaţiunile de trasncodare sunt realizate în video proxy-cache. Valoarea care corespunde fiecărei poziţii i din şir reprezintă al i^{lea} coeficient. Aceste valori variază de la 0 la 1.

Am utilizat atât formula (6.2) cât şi formula (6.3) în calcularea utilităţii unui obiect. Înainte de aceasta am scalat valorile genei din cromozom pentru a ne asigura că suma coeficienţilor este egală cu 1. Astfel, valoarea fiecărui coeficient este calculată folosind formula:

$$coef_i = \frac{gene_i}{\sum_{i=1}^{3} gene_i} \qquad (6.4)$$

unde *chromosome* = ($gene_1$, $gene_2$, $gene_3$).

Evaluarea calității fiecărei soluții (cromozom) a fost realizată calculând *byte hit ratio*. Scopul nostru a fost să maximizăm această valoare.

6.4. Rezultate experimentale

Algoritmul GeCo a fost testat aplicând o serie de experimente în timpul cărora ne-am axat pe valorile obținute pentru *byte hit ratio*. Am folosit un număr de 12 instanțe de testare. Am comparat rezultatele obținute atunci când am utilizat strategii de înlocuire a cache-ului bazate pe utilitate (funcțiile de utilitate (6.2) şi (6.3) a căror coeficienți sunt determinați folosind GeCo) cu cele calculate folosind algoritmi clasici de înlocuire a cache-ului (LRU şi LFU) [Podlipnig&Böszörményi2003].

Când am realizat măsurătorile am pornit de la presupunerea că nu se foloseşte segmentarea obiectelor video (tratare a obiectelor în stil web) şi că odată solicita, un obiect este complet recuperat de pe un server de origine (dacă nu este deja cached) şi este vizionat de la început până la sfârşit fără întreruperi sau anulări. De asemenea, nu au fost luate în considerare limitări în ce priveşte lățimea de bandă sau erorile de transmisie. Realizând că aceste presupuneri nu sunt realiste, rezultatele obținute reprezintă un bun punct de referință pentru cazul ideal.

Datele folosite pentru experimente au fost generate utilizând WebTraff [Markatchev&Williamson2002] un generator artificial de trafic web. Caracteristicile celor 12 instanțe utilizate sunt prezentate în Tabelul 6.1.

Tabelul 6.1 Caracteristici ale instanțelot generate

Trace ID	Number of requests	Number of objects	One-Timers(% of total objects)	Zipf slope
1	1000	300	70	0.3
2	1000	300	30	0.3
3	1000	300	70	0.75
4	1000	300	30	0.75
5	5000	1500	70	0.3
6	5000	1500	30	0.3
7	5000	1500	70	0.75
8	5000	1500	30	0.75
9	10000	3000	70	0.3
10	10000	3000	30	0.3
11	10000	3000	70	0.75
12	10000	3000	30	0.75

Intenția noastră a fost să măsurăm *byte hit ratio* pe o distribuție a popularității obiectelor atât lightly skewed (Zipf α = 0.3) cât şi more severe skewed (Zipf α = 0.75) cu o cantitate variată de one-timers (obiecte solicitate o singură dată) (30% pentru instanțele 2, 4, 6, 8, 10 şi 12 vs. 70% în instanțele 1, 3, 5, 7, 9 şi 11). Am variat de asemenea şi mărimea cache-ului de al 1% la 10% din mărimea totală a obiectelor solicitate (mărimea tuturor obiectelor solicitate de mai multe ori a fost luată în considerare doar o dată).

Valoarea optimă pentru fiecare dintre cele 12 instanțe a fost calculată folosind formula:

$$1 - \frac{TotalSizeOfUniqueObjects}{TotalSizeOfTransferredData} \quad (5)$$

Algoritmul genetic (GeCo) pe care l-am folosit are următoarele setări: mărimea populației: 100, numărul de generații: 10, probabilitatea de încrucişare: 0.7 şi probabilitatea de mutație: 0.6.

Am prezentat rezultatele în termeni de *byte hit rate* în Figura 6.1 şi Figura 6.2 obținute pentru instanțele 11 şi 12 cu coeficienții. Dacă studiem rezultatele putem observa că folosirea utilității (6.2) duce la rezultate mai bune decât folosirea utilității (6.3). În cazurile în care distribuția popularității obiectelor este more severe skewed (Zipf α = 0.75) utilitatea (6.2) generează în mod clar rezultate mai bune decât LRU în timp ce în cazul în care distribuția popularității obiectelor este lightly skewed (Zipf α = 0.3) utilitatea (6.2) produce rezultate comparabile cu LRU (dar totuşi ceva mai bune) (Figura 6.2).

Este de asemenea interesant să observăm că pentru instanțe mari, creşterile în termeni de *byte hit ratio* sunt neglijabile pentru o dimensiune a cache-ului mai mare de 7%. Aceasta înseamnă că nu avem nevoie de dimensiuni extrem de mari ale cache-ului pentru a obține valori ale *byte hit ratio* bune, ceea ce se traduce printr-o reducere a utilizării lățimii de bandă externe.

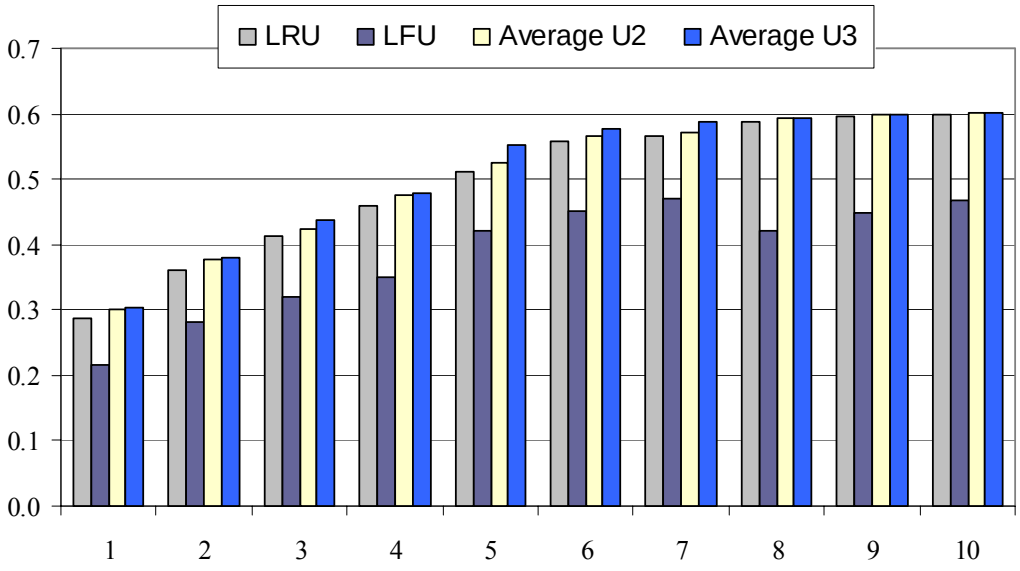


Figura 6.1 Valorile byte hit ratio pentru instanța 11

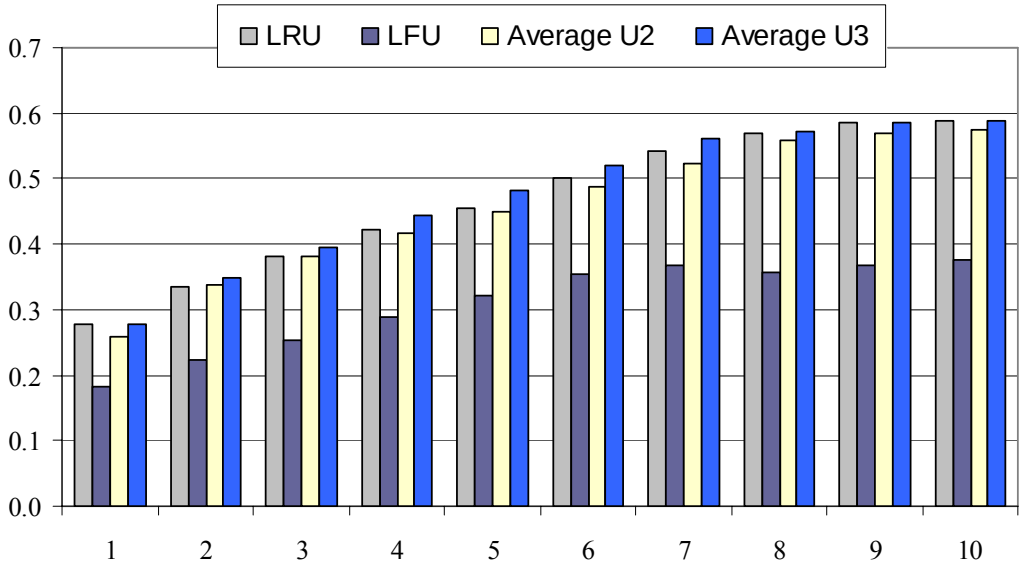


Figura 6.2 Valorile byte hit ratio pentru instanța 12

Tabelul 6.2 redă pe scurt cu cât exact este utilitatea (6.2) mai bună decât LRU când se iau în considerare rezultatele bune şi medii ale *byte hit rate*. Medie este de aproximativ 1.7%, procent semnificativ dacă e să luăm în considerare modele de stabilire a prețului.

Tabelul 6.2 Mean BHR difference between situations when using utility

Trace ID	Mean Best	Mean Average
1	1.0287%	0.6782%
2	1.1965%	0.9833%
3	4.0188%	3.7725%
4	0.3896%	0.1306%
5	1.1521%	0.6752%
6	0.3955%	0.0694%
7	2.4681%	2.1730%
8	0.8924%	0.6412%
9	1.0731%	0.8707%
10	0.7187%	0.4404%
11	1.8760%	1.7511%
12	1.3319%	1.1791%
Average	1.3784%	1.1137%

6.5. Concluzii şi direcții viitoare de cercetare

Am propus două noi modalități (ecuațiile (6.2) şi (6.3)) de definire a utilității obiectelor stocate într-un video proxy-cache care oferă o mai bună echilibrare între caracteristicile luate în considerare (*size*, *timeLastAccess*, *hitCount*) pentru fiecare obiect. Am folosit de asemenea un algoritm genetic (GeCo) pentru determinarea coeficienților care apar în acele două definiții cu

scopul de a maximiza *byte hit rate*. Rezultatele obținute în termeni de *byte hit rate* când s-au folosit algoritmul GeCo şi funcția de utilitate (6.2) sunt similare sau chiar mai bune decât cele obținute pentru LRU.

Intenționăm să utilizăm o abordare similară a algoritmului GeCo prezentat pentru a determina coeficienții care reglementează dinamica sistemului propus în [Cobârzan2005] şi [Cobârzan&Böszörményi2007] mai precis adăugarea de noi noduri proxy-cache la sistemul federativ de cache, respectiv eliminarea lor atunci când aceştia nu mai sunt necesari. Strategiile de segmentare a obiectelor video cache folosind algoritmul GeCo merită de asemenea investigații suplimentare.

Concluzii şi direcţii viitoare de cercetare

Scopul acestei lucrări a fost să cerceteze utilizarea algoritmilor genetici în rezolvarea diferitelor clase de probleme de alocare. Deoarece problemele abordate sunt grele din punct de vedere al timpului nedeterminist polinomial alternativa de a folosi algoritmi genetici s-a dovedit a fi una de succes.

În acest sens au fost mai întâi prezentate elemente de bază ale problemelor de alocare (sarcini, resurse şi funcţii obiectiv), o serie de aspecte legate de aceste elemente, o schemă de clasificare a problemelor de programare precum şi complexitatea acestor probleme. Apoi am realizat o imagine de ansamblu asupra elementelor de bază ai algoritmilor genetici (reprezentare, funcţia fitness, selecţia, încrucişarea, mutaţia şi parametrii)

Utilizând bazele menţionate mai sus, am abordat trei probleme de programare:

- o problemă de alocare pe maşini/procesoare paralele şi uniforme care a fost rezolvată folosind un nou algoritm genetic. Algoritmul propus a obţinut rezultate mai bune decât alţi algoritmi.
- o problemă de alocare de tip permutation flow shop care a fost rezolvată folosind un nou algoritm genetic hibrid. . Noutatea algoritmului propus constă în utilizarea unei combinaţii între o procedură de iniţializare arbitrară şi o procedură de iniţializare bazată pe euristica constructivă NEH precum şi în definirea unui nou operator de încrucişare şi în folosirea unui operator de mutaţie definit ca o combinaţie între un euristica constructivă NEH şi mutaţia prin translatare. Rezultatele obţinute de către algoritmul propus sunt comparabile cu rezultatele obţinute de alţi algoritmi.
- o problemă de alocare a unui video proxy-cache. Pentru această problemă au fost definite două noi formule pentru utilitatea obiectelor stocate într-un video proxy-cache. A fost utilizat un nou algoritm genetic pentru a determina coeficienţii care apar în aceste două definiţii cu scopul de a maximiza byte hit rate. Rezultatul obţinut de către algoritmul propus cu o funcţie de utilitate este similar sau chiar mai bun decât rezultatele obţinute de altă metrică.

Intenționez să îmi îndrept viitoarele cercetări în următoarele direcții convergente:

- alocarea de sarcini şi resurse în managementul proiectelor pentru a investiga domeniul proiectelor care pare să reprezinte motorul dezvoltării economice,
- optimizarea multi-obiectiv pentru a lua în considerare mai mult decât o funcție obiectiv ca scop al optimizării,
- optimizarea în medii dinamice pentru a simula/capta schimbările care apar în timpul ciclului de viață al unui proiect
- programare stocastică pentru a simula mai bine situațiile reale din viață.

Bibliografie

[Abraham et. al. 2000] Abraham A., Buyya R. and Nath B., Nature's Heuristics for Scheduling Jobs in Computational Grids, in *Proceedings of 8th IEEE International Conference on Advanced Computing and Communications*, Tata McGraw-Hill Publishing Co. Ltd, New Delhi, pp. 45-52, 2000.

[Abraham et. al. 2006] Abraham A, Liu H, Zhang W, Chang TG, Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm, *Proceedings of 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems*, England, pp. 500-507, 2006.

[Abraham et. al. 2008] Abraham A, Liu H., Grosan C., and Xhafa F., *Nature Inspired Metaheuristics for Grid Scheduling: Single and Multiobjective Optimization Approaches, Metaheuristics for Scheduling: Distributed Computing Environments*, Studies in Computational Intelligence, Springer Verlag, Germany, pp. 247-272, 2008.

[Acharya2002] S. Acharya and B. Smith. Middleman: A video caching proxy server. In *Proceedings of the 10th International Workshop on Network and Operating System Support forDigital Audio and Video*, 2002.

[Ashlock2006] Ashlock, D., *Evolutionary Computation for Modeling and Optimization*, Springer, 2006

[Back et. al. 2000] Back T., D.B. Fogel, and Z. Michalewicz (Eds), *Evolutionary Computation: Basic Algorithms and Operators*, Vol. 1 and Vol. 2, Institute of Physics Publishing, Philadelphia, PA, 2000.

[Baeck2000] Baeck, T., Fogel, D., Michalewicz. (eds)., *Evolutionary Computation*, vol. 1 and 2, Institute of Physics Publishing, 2000.

[Balafoutis et. al. 2002] E. Balafoutis, A. Panagakis, N. Laoutaris, and I. Stavrakakis. The impact of replacement granularity on video caching. In *IFIP Networking 2002*, volume 2345 of *Lecture Notes in Computer Science*. Springer, 2002.

[Blazewicz et. al. 2007] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G. and Weglarz, J., *Handbook on scheduling. From Theory to Applications*, Springer, 2007.

[Blazewicz1983] Blazewicz J., Lenstra, J.K., A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics* 5, p. 11-24, 1883

[Brubeck&Brubeck1996] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed vod system. *IEEE MultiMedia*, 3(3):37–47, 1996.

[Brucker1999] Brucker, P., Drexl, A., Moehring, R., Neumann, K., Pesch, E., Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods, *European Journal of Operational Research*, no. 112, pp. 3-41, 1999.

[Brucker2007] Brucker, P., *Scheduling Algorithms*, Springer, 2007

[Budiu1999] Budiu, M., 1999, http://www.cs.cmu.edu/~mihaib/articole/

[Buyya et. al. 2000] Buyya R, Abramson D, Giddy J, Grid Resource Management, Scheduling, and Computational Economy, *International Workshop on Global and Cluster Computing,* Japan, 2000.

[Chen et. al. 1995] Chen, C.-L., Vempati, V. S., and Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. European Journal of Operational Research, 80(2):389–396.

[Cobârzan&Böszörményi2007] C. Cobârzan and L. Böszörményi. Further developments of a dynamic distributed video proxy-cache system. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007)*, pages 349–357. IEEE Computer Society, 2007.

[Cobârzan2005] C. Cobârzan. Dynamic proxy-cache multiplication inside LANs. In *Euro-Par 2005*, volume 3648 of *Lecture Notes in Computer Science*, pages 890–900. Springer, 2005.

[Cobârzan&Mihăilă2008] Cobârzan C., **Mihăilă C.**, A Genetic Algorithm for Utility Based Video Proxy-Caching, in *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 231-238, Timişoara, România, 2008

[Conway2003] R. Conway, W. Maxwell and L. Miller. *Theory of scheduling*. Dover Publications Inc., reprint edition, 2003.

[Cook1971] Cook, S., The Complexity of Theorem Proving Procedures, *in Proceedings of the third annual ACM symposium on Theory of computing*, pp.151–158, 1971.

[Dannenbring1977] Dannenbring, D. G. An evaluation of flow shop sequencing heuristics. Management Science, 23, 11 (1977), 1174--1182.

[Demeulemeester2002] Demeulemeester, E.L., Herroelen, W.S., *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers, Dordrecht, 2002.

[Drozdowski1996] Drozdowski M., *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*, Poznan University of Technology Press, Poznan, 1996.

[Dumitrescu2000] Dumitrescu, D., Lazzerini, B., Jain, L., Dumitrescu, A., *Evolutionary Computation.* CRC Press, Boca Raton, FL, 2000.

[Dumitrescu et.al.2002a] Dumitrescu, D., Iantovics, B., **Florea, C.**, Multi-Agent Systems: a new allocation protocol and evolutionary search for equilibrium; , in *Proceedings of the Symposium "Zilele academice clujene" - Computer Science Section*, 119-133, Cluj-Napoca, Romania, 2002.

[Dumitrescu et.al.2002b]Dumitrescu, D., **Florea, C.**, Patranjan; P., Evolutionary Reorganization in MAS; , in *Proceedings of the European Conference on Information Technology* (ECIT02), 1-5, Iasi, Romania, 2002.

[Dumitrescu et.al.2002c]Dumitrescu, D., **Florea, C.**, Patranjan; P., A New Evolutionary Model for Multi Agent Systems, in *Proceedings of* the *4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC02), 137-143, Timisoara, Romania, 2002.

[Eiben2003] Eiben, A.E., Smith, J.E., *Introduction to Evolutionary Computing*, Springer, 2003.

[Fabmi et. al. 2001] H. Fabmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. Hsu. Proxy servers for scalable interactive video support. *Computer*, 34(9):54–60, 2001.

[Florea&Dumitrescu2003] **Florea, C**., Dumitrescu, D., Negotiation in Multiagent Systems; in *Proceedings of the Conference on Applied and Industrial Mathematics* (CAIM03), Oradea, Romania, 2003.

[Fogel1966] Fogel, L.J., Owens, A.J., Walsh, M.J., *Artificial Intelligence through Simulated Evolution*, John Wiley, 1966.

[Garey&Johnson1979] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman, New York, 1979.

[Garey1979] Garey, M.R., Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, WH Freeman&Co, San Francisco, 1979.

[Goldberg1989] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning,* Kluwer Academic Publishers, Boston, MA, 1989.

[Grabowski&Wodecki2004] Grabowski, J. and Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research, 31(11):1891–1909.

[Graham1979] Graham, R.L., Lawler, E.L., Lenstra, J.K., A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Annals of Discrete Mathematics* 5, p. 287-326, 1979

[Groşan et.al.2003] Groşan, C., Oltean, M., **Florea, C.**, NP-complete problems using Evolutionary Algorithms, *Lucrarile Seminarului de Didactica Matematicii al Universitatii Babes-Bolyai*, Vadu-Crisului, Romania, 2003.

[Grosan et. al. 2007] Grosan, C., Abraham, A., and Helvik, B., Multiobjective Evolutionary Algorithms for Scheduling Jobs on Computational Grids, *IADIS International Conference, Applied Computing 2007*, pp. 459-463, 2007.

[Guangchang et. al. 2006] Guangchang Ye, Ruonan Rao, Minglu Li, A Multiobjective Resources Scheduling Approach Based on Genetic Algorithms in Grid Environment, Fifth International Conference on Grid and Cooperative Computing Workshops, pp. 504-509, 2006.

[Holland1975] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[HsiuMa&Du2000] W. Hsiu Ma and D. H.-C. Du. Reducing bandwidth requirement for delivering video over wide area networks with proxy server. In *IEEE International Conference onMultimedia and Expo (II)*, pages 991–994. IEEE Computer Society, 2000.

[Johnson1954] Johnson, S. M. Optimal two-and three-stage production schedules. Naval Research Logistics Quarterly, 1 (1954), 61--68.

[Karp1972] Karp, R.M., Reducibility Among Combinatorial Problems, *Complexity of Computer Computations*, pp. 85-103, Plenum Press, 1972

[Kumar et. al. 2004] Kumar , K.P., Agarwal , A., and Krishnan, R., Fuzzy based resource management framework for high throughput computing, in *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, 555-562, 2004.

[Leung2000] Leung, J.Y-T. (ed.), *Handbook of Scheduling. Algorithms, Models and Performance Analysis,* Chapman & Hall/CRC Press, Boca Raton, 2000.

[Leung2004] Leung, J.Y-T, Anderson, J.H., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman and Hall / CRC, Boca Raton, Florida, 2004.

[Liu1995] Liu, Z., Sanlaville, E., Preemptive scheduling with variable profile, precedence constraints and due dates, Discrete Applied Mathematics 58, p. 253-280, 1995

[Markatchev&Williamson2002] N. Markatchev and C. Williamson. Webtraff: A gui for web proxy cache workload modeling and analysis. In *MASCOTS'02: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*. IEEE Computer Society, 2002.

[Mihăilă&Cobârzan2008] **Mihăilă C.**, Cobârzan C., Evolutionary approach for multimedia caching, in *IEEE Proceedings of the Evolutionary Techniques in Data Processing Workshop,* International Conference on Database and Expert Systems Application (DEXA), 531-536, Torino, Italia, 2008

[Mihăilă&Dumitrescu] **Mihăilă, C.**, Dumitrescu, D., Quantum Computing and Multiagent Systems, in *Proceedings of the Symposium "Colocviul Academic Clujean de Informatica",* Cluj-Napoca, Romania, 2005.

[Mihăilă&Mihăilă2008a] **Mihăilă C.**, Mihăilă A., An Evolutionary Algorithm for Uniform Parallel Machines Scheduling, in *IEEE Proceedings of the European Modelling Symposium*, 76-80, Liverpool, United Kingdom, 2008.

[Mihăilă&Mihăilă2008b] Mihăilă A., **Mihăilă C.**, Uniform Parallel Machines Scheduling using an Evolutionary Algorithm, in *IEEE Proceedings of the International Workshop on Evolutionary Multiobjective Optimization Design and Applications*, International

Conference on Intelligent Systems Design and Applications (ISDA), 401-406, Kaohsiung, Taiwan, 2008

[Mihăilă et.al.2008a] Mihăilă A., Mihiş A., **Mihăilă C.,** Genetic Algorithm for Logical Topic Text Segmentation, in *IEEE Proceedings of the International Conference on Digital Information Management*, 500-505, London, United Kingdom, 2008

[Mihăilă et.al.2008b] **Mihăilă C**., Nițchi, I.Ş., R., Mihăilă, A., Coroş R., A genetic algorithm for permutation flow shop scheduling problem, in Annals of Tiberiu Popoviciu Seminar of Functional Equation, Approximation and Convexity, p. 241-250, Cluj-Napoca, România, 2008

[Mihiş et.al.2006] Mihiş, A., Creţu, C., **Mihăilă, C.**, Şerban, C., Code Simplification using Boolean Functions Simplification, in *Proceedings of the International Conference of Mathematics & Informatics*, Supplement of "Studii si cercetari stiintifice. Seria Matematică", no.16, University of Bacau, 493-502, Bacău, Romania, 2006.

[Montana2007]Montana, D., Hussain, T., Vidaver, G., A Genetic-Algorithm-Based Reconfigurable Scheduler, *Evolutionary Scheduling,* Springer, 2007.

[Murata et. al. 1996] Murata, T., Ishibuchi, H., and Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering, 30(4):1061–1071.

[Nawaz et. al. 1983] Nawaz, M., Enscore Jr., E. E., and Ham, I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. OMEGA, 11, 1 (1983), 91--95.

[Nițchi et.al.2007a] Nițchi, I.Ş., Mihăilă, A., **Mihăilă, C.**, About Project Management Planning Optimization using Genetic Algorithms, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 79-82 ,Cluj-Napoca, România, 2007.

[Nițchi et.al.2007b] Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A., **Mihăilă, C.**, About the Logical Model for Intelligent Agents, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 83-90, Cluj-Napoca, România, 2007.

[Nițchi et.al.2007c] Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A**.**, **Mihăilă, C.**, On the collaborative systems for e-business, in *Proceedings of the International Conference on Competitiveness and European Integration*, 266-272, Cluj-Napoca, România, 2007.

[Nowicki&Smutnicki1996] Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flowshop problem. European Journal of Operational Research, 91(1):160–175.

[Oltean et.al.2009] Oltean M., Groşan C., Dioşan L., **Mihăilă C.,** *Genetic Programming with Linear Representation a Survey,* International Journal on Artificial Intelligence Tools, 197-238, 2009

[Osman&Potts1989] Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science, 17(6):551–557.

[Park et. al. 2001] S.-H. Park, E.-J. Lim, and K.-D. Chung. Popularity-based partial caching for vod systems using a proxy server. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*. IEEE Computer Society, 2001.

[Podlipnig&Böszörményi2002] S. Podlipnig and L. Böszörményi. Replacement strategies for quality based video caching. In *IEEE International Conference on Multimedia and Expo (ICME), Vol. 2*, pages 49–53. IEEE Computer Society, 2002.

[Podlipnig&Böszörményi2003] S. Podlipnig and L. Böszörményi. A survey of web cache replacement strategies. *ACMComput. Surv.*, 35(4):374–398, 2003.

[Rajendran&Ziegler2004] Rajendran, C., and Ziegler, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research, 155, 2 (2004), 426--438.

[Rechenberg1973] Rechenberg, I., *Evolution Strategy* , Frommann-Holzboog, Stuttgart, pp.147-159, 1973.

[Reeves&Yamada1998] Reeves, C. and Yamada, T. (1998). Genetic algorithms, path relinking, and the flowshop sequencing problem. Evolutionary Computation, 6(1):45–60.

[Rejaie&Kangasharju2001] R. Rejaie and J. Kangasharju. Mocha: A quality adaptive multimedia proxy cache for internet streaming. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 3–10. ACM Press, 2001.

[Ritchie&Levine2003] Ritchie, G. and Levine, J., A fast, effective local search for scheduling independent jobs in heterogeneous computing environments, Technical report, Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, 2003.

[Ritchie&Levine2004] Ritchie, G. and Levine, J., A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in *23rd Workshop of the UK Planning and Scheduling Special Interest Group*, 2004.

[Ruiz et. al. 2004] Ruiz, R., Maroto, C., and Alcaraz, J. (2004). Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, the International Journal of Management Science.

[Ruiz&Maroto2004] Ruiz, R. and Maroto, C. (2004). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. In press.

[Ruiz&Stützle2007] Ruiz, R., and Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177 (2007), 2033--2049.

[Sasabe et. al. 2001] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Proxy caching mechanisms with video quality adjustment. In *Proceedings of SPIE International Symposium on The Convergence of Information Technologies and Communications*, pages 276–284, 2001.

[Schmidt1984] Schmidt, G., Scheduling on semi-identical processors, Zeitschrift für Operations Research. A28, p. 153-162, 1984

[Sen et. al. 1999] S. Sen, J. Rexford, and D. F. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM*, pages 1310–1319. IEEE Computer Society, 1999.

[Smith 2005] Smith, S.F., *Is Scheduling a Solved Problem?*, Multidisciplinary Scheduling: Theory and Applications, Springer, pp. 3-17, 2005.

[Suliman2000] Suliman, S. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. International Journal of Production Economics, 64:143–152.

[Taillard1990] Taillard, E. Some efficient heuristic methods for the flowshop sequencing problems. European Journal of Operational Research, 47 (1990), 65--74.

[Taillard1993] Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64(2):278–285.

[Taillard2004] Taillard, E. (2004). Summary of best known lower and upper bounds of Taillard's instances. http://mistic.heig-vd.ch/taillard/.

[Tompkins2003] Tompkins, M. F., *Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations*, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[Veltman1993] Veltman, B., Multiprocessor Scheduling with Communication Delays, Ph.D Thesis, CWI-Amstrerdam, 1993.

[Weawer2006] Weawer Patrick, A brief history of scheduling, myPrimavera06, Canberra, 2006, http://www.pmforum.org/library/papers/2006/A_Brief_History_of_Scheduling.pdf

[Wu et. al. 2001] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 36–44. ACM Press, 2001.

[Yang& Towsley2002] S. S. Yang Guo and D. Towsley. Prefix caching assisted periodic broadcast for streaming popular videos. In *Proceedings of ICC (International Conference on Communications)*, pages 2607 – 2612. IEEE Computer Society, 2002.

[Yarkhan&Dongarra2002] Yarkhan, A. and Dongarra, J., Experiments with scheduling using simulated annealing in a grid environment, in *3rd International Workshop on Grid Computing* (GRID2002), 232-242, 2002.

[Zomaya&The2001] Zomaya, A.Y. and The, Y.H., Observations on using genetic algorithms for dynamic load-balancing, *IEEE Transactions On Parallel and Distributed Systems*, 12(9):899-911, 2001.