Liet. matem. rink., **44**, *spec. nr.*, 2004, 220–225

Lygiagrečiųjų šakų ir rėžių algoritmų programų kūrimas *

Milda BARAVYKAITĖ (VGTU)
el. paštas: mmb@fm.vtu.lt

1. Įvadas

Šakų ir rėžių algoritmas nusako principus, kaip ieškoti tikslaus optimizavimo uždavinio sprendinio. Konkrečiam uždaviniui šie principai įgyvendinami skirtingai. Praktikoje sprendžiami uždaviniai reikalauja daug skaičiavimų ir/ar didelių kompiuterių atminties resursų, todėl gali prireikti lygiagrečiųjų šakų ir rėžių algoritmų taikymo. Šio darbo tikslas sukurti algoritmo šabloną, palengvinantį lygiagrečiųjų šakų ir rėžių programų kūrimą.

2. Šakų ir rėžių algoritmo principai

Tarkime, sprendžiamas minimizavimo uždavinys:

$$\min_{x \in S} f(x),$$

čia $f(x)$ – minimizuojama funkcija, $x = (x_1, x_2, \ldots, x_n)$, S – galimų sprendinių sritis, kurioje ieškoma optimalaus sprendinio [1].

Algoritmo idėja – nustatyti sprendinių sritis, kuriose tikrai nebus optimalaus uždavinio sprendio ir tų sričių toliau netirti. Jei tai pavyksta, uždavinys išsprendžiamas greičiau, jei ne – algoritmas veikia kaip pilno perrinkimo algoritmas. Sritys, kuriose nebus optimalaus sprendinio, nustatomos pradinį uždavinį skaidant į mažesnius, įvedant papildomus ribojimus sprendiniui taip, kad mažesnį uždavinį būtų lengviau išspręsti arba įvertinti. Tai algoritmo šakos žingsnis. Įvertinimui skaičiuojami funkcijos reikšmių toje srityje rėžiai. Kuo tiksliau skaičiuojamas rėžis, tuo greičiau bus atmetamos neperspektyvios sritys. Jeigu užduoties srityje apatinis minimizuojamos funkcijos reikšmių rėžis yra didesnis už šiuo metu žinomą geriausią sprendinį, reiškia toliau šios srities galima netirti, nes ten optimalaus sprendinio nebus. Tikslus sprendinys randamas, kai nebelieka neištirtų sričių. Bet kuriuo sprendimo momentu yra žinomas geriausias iki to laiko rastas sprendinys ir dar neištirtos sprendinių sritys.

Galimos nekantri ir tingi algoritmo strategijos, kurias lemia kokia tvarka vykdomi pagrindiniai algoritmo šakos ir rėžio žingsniai. Pateikime apibendrintą nekantriosios strategijos algoritmą:

*Šis darbas atliktas pagal EUREKA projektą CTBSTROKE E!2981, remiamas Lietuvos valstybinio mokslo ir studijų fondo (sutartis nr. V-04049).

```
Geriausias žinomas sprendinys (GZS) = ∞ (arba euristinis sprendinys)
pradinė užduotis T0
LB = bound(T0)
įterpti (T0, LB) į Neištirtų užduočių eilę (NUE)
while (yra užduočių NUE) {
   Paimti Ti iš NUE
   Branch(Ti)   Ti = ⋃j Tj
   visiems Ti vaikams Tj {
      LB = bound (Tj)
      if ((Tj yra sprendinys) ir f(Tj) ⩽ GZS)
         Spr = Tj, GZS = f(Tj)
      else if (LB< GZS)
         įterpti Tj į NUE
      }
   }
Optimalus sprendinys = Spr, optimali reikšmė = GZS.
```

Algoritmo efektyvumą konkrečiam uždaviniui įtakoja ir tai, kokia tvarka tiriamos užduotys – veiksmai su NUE. Išskiriamos trys taisyklės: geriausios ribos taisyklė (*best first search*), kai pirmiausia tiriamos užduotys su geriausiais rėžiais, naujausios ribos taisyklė (*depth first search*), kai pirmiausia tiriamos vėliausiai sudarytos užduotys ir apžvalgos taisyklė (*breadth first search*), kai pirmiausia tiriamos anksčiausiai sudarytos užduotys [1].

3. Spręsti uždaviniai

Tiriant šakų ir rėžių algoritmą buvo sprendžiamas simetrinis keliaujančio pirklio uždavinys ir minimizuojama Lipšico funkcija.

Keliaujančio pirklio uždavinys – aplankyti visus duotus miestus taip, kad viso maršruto svoris būtų mažiausias. Maršruto svoriu gali būti įvardintas jo ilgis, kaina, trukmė. Kiekvienas miestas gali būti aplankytas tik vieną kartą ir maršrutas turi baigtis tame pačiame mieste, iš kurio prasidėjo. Uždavinys vadinamas simetriniu, kai vienodai įvertinamas kelias į abi puses. Formaliai uždavinys užrašomas taip:

$$\min_{x_{ij}} \sum_{ij} d_{ij} x_{ij}$$

taip, kad

$$\sum_{i \neq j} x_{ij} = 2, \quad i \in \{1 \ldots n\},$$

$$\sum_{i,j \in \mathbb{Z}} x_{ij} < |\,\mathbb{Z}\,|, \quad \varnothing \subset \mathbb{Z} \subset \mathbb{V},$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in \{1 \ldots n\}.$$

Čia d_{ij} – svorio įvertis tarp miestų i ir j, x_{ij} parodo, ar kelias įtrauktas į maršrutą, n – miestų skaičius. Skaičiavimo eksperimentuose $n = 25$.

Lipšico funkcijos minimizavimo uždavinys:

$$\min f(x), \quad x \in D,$$

kur $a = (a_1, \ldots, a_n)^T$, $b = (b_1, \ldots, b_n)^T \in \mathbb{R}^n$, $a < b$, f: $D = \{X\colon a \leqslant x \leqslant b\} \to \mathbb{R}$ yra Lipšico funkcija, tenkinanti sąlygą: $|f(x_2) - f(x_1)| \leqslant L\|x_2 - x_1\|$, čia L yra Lipšico konstanta. Skaičiavimo eksperimentuose $f(x) = -1/((x_1 - 4)^2 + (x_2 - 4)^2 + 0,7) - 1/((x_1 - 2,5)^2 + (x_2 - 3,8)^2 + 0,73)$, kuri buvo minimizuojama $D = [0, 10]^2$, tikslumu 0,0001 [2].

4. Šablonai

Šakų ir rėžių algoritme galima išskirti dalis, kurios yra vienodos visiems uždaviniams, ir dalis, kurios skiriasi, priklausomai nuo sprendžiamo uždavinio. Todėl, apjungėme bendras visiems uždaviniams algoritmo dalis ir sukūrėme algoritmo šabloną, apibendrinantį veiksmus su objektais *Užduotis* ir *Sprendinys*. Šie priklauso nuo sprendžiamo uždavinio ir yra sukuriami šablono vartotojo. Kuriant algoritmo šabloną, realizuotos trys NUE tvarkymo taisyklės. Tokiu būdu kurdamas nuosekliąją programą, vartotojas gali testuoti, koks eilės tvarkymo algoritmas geriausiai tinka konkrečiam uždaviniui.

Nuosekliojo algoritmo šablonas papildytas lygiagrečiaisiais algoritmais, naudojančiais tuos pačius objektus, kaip ir nuoseklusis šablonas. Taip vartotojas gali gauti lygiagrečiąją programą neperrašydamas nuosekliajai programai rašyto kodo. Įdiegiant sudėtingesnius lygiagrečiuosius algoritmus vartotojui reikia papildomai nurodyti, kaip perduoti duomenis tarp procesorių, tačiau vartotojui nereikia gilintis į lygiagretųjį programavimą.

5. Lygiagretieji šakų ir rėžių algoritmai

Galimi įvairūs lygiagretieji šakų ir rėžių algoritmai: kai tiriant užduotį vienas iš algoritmo žingsnių vykdomas lygiagrečiai, kai lygiagrečiai tiriamos kelios užduotys ar kai lygiagrečiai vykdomi skirtingi šakų ir rėžių algoritmai [4]. Algoritmus galima klasifikuoti ir pagal tai, kiek neištirtų užduočių eilių kuriama, ar procesoriai apsikeičia duomenimis.

Šablone įdiegti paprasti lygiagretieji šakų ir rėžių algoritmai naudojantys kelias užduočių eiles. Algoritmo pradžioje sukuriama pradinė užduočių eilė, kuri išdalinama procesoriams. Toliau kiekvienas procesorius vykdo nuoseklųjį šakų ir rėžių algoritmą savo srityje, kai visi procesoriai baigia skaičiavimus, išrenkamas geriausias rastas sprendinys. Taip pat įdiegtas algoritmas, kuris papildytas geriausio žinomo sprendinio pasikeitimu: radę geresnį sprendinį, procesoriai paskleidžia jį kitiems.

Tokiu būdu lygiagrečiajame algoritme užduotys tiriamos ne tokia pačia tvarka, kokia jos tiriamos analogiškai apibrėžtame nuosekliajame algoritme, todėl sunku įvertinti lygiagrečiojo algoritmo kokybę. Siūloma skaičiuoti algoritmo pertekliškumo koeficientą $\frac{W_p}{W}$, kur W_p – darbo apimtis sprendžiant uždavinį p procesorių, W – nuosekliojo algoritmo darbo apimtis [5]. Darbo apimtį siūloma matuoti ištirtų užduočių skaičiumi, darant prielaidą, kad kiekvienai užduočiai ištirti reikia tiek pat laiko. Daugelio lygiagrečiųjų šakų ir rėžių algoritmų pertekliškumo koeficientas yra didesnis už vienetą [5].

6. Skaičiavimo eksperimentų rezultatai

Naudojant šabloną nesunku išbandyti įvairias NUE tvarkymo strategijas. Kai eilė tvarkoma geriausios ribos taisykle, Lipšico funkcijos minimizavimo uždavinys išsprendžiamas ištyrus 788 užduotis, kai naudojama naujausios ribos taisyklė – 694053, o kai apžvalgos taisyklė – 3327. Keliaujančio pirklio uždavinys naujausios ribos taisykle išsprendžiamas ištyrus 28360 užduotis, apžvalgos taisykle – 4424 užduotis.

1 lentelėje pateikti Lipšico funkcijos minimizavimo rezultatai gauti VGTU klasteryje *Vilkas*. Uždavinys spręstas algoritmu be geriausio sprendinio siuntimų, naudojama geriausios ribos eilės tvarkymo taisyklė. Pradžioje procesoriams išdalinama po 10 užduočių. Pateikiamas skaičiavimo laikas sekundėmis, spartinimo ir efektyvumo koeficientai, taip pat bendras procesorių ištirtų užduočių skaičius ir pertekliškumo koeficientas.

Iš rezultatų matyti, kad algoritmas nėra efektyvus, išauga ištirtų užduočių skaičius, pertekliškumo koeficientas didesnis už 2. Lygiagrečiojo algoritmo efektyvumą dar pablogina nevienodas procesorių apkrovimas darbu.

2 lentelėje pateikiami algoritmo su geriausio sprendinio siuntimu vykdymo rezultatai. Geriausiems sprendiniams persiųsti naudojamas nesinchronizuotas duomenų perdavimas.

Iš rezultatų matyti, kad algoritmo pertekliškumas sumažėjo, tačiau algoritmas nėra efektyvus. 3 lentelėje pateikiami kiekvieno procesoriaus ištirtų užduočių skaičiai demonstruoja, kad procesoriai yra netolygiai apkrauti darbu.

Tiriant keliaujančio pirklio uždavinį gaunami panašūs rezultatai. 4 lentelėje pateikti algoritmo be geriausio sprendinio siuntimų rezultatai, o 5 lentelėje – algoritmo su geriausio sprendinio siuntimu, naudojama naujausios ribos taisyklė, pradžioje išdalinama po vieną užduotį.

1 lentelė. Lipšico funkcijos minimizavimo uždavinys, algoritmas be geriausio sprendinio siuntimų

Proc.	T	S	E	W_p	W_p/W
1	0,943	1	1	788	–
2	1,376	0,685	0,343	1600	2,030
3	1,035	0,911	0,304	1685	2,138
4	1,233	0,765	0,191	1744	2,213
5	0,992	0,951	0,190	1840	2,33

2 lentelė. Lipšico funkcijos minimizavimo uždavinys, algoritmas su geriausio sprendinio siuntimu

Proc.	T	S	E	W_p	W_p/W
1	0,943	1	1	788	–
2	0,907	1,039	0,519	1227	1,557
3	0,680	1,386	0,462	1284	1,629
4	0,495	1,905	0,476	1167	1,481
5	0,371	2,541	0,508	1263	1,603

3 lentelė. Lipšico funkcijos minimizavimo uždavinys, algoritmas su geriausio sprendinio siuntimu, procesorių apkrovimas

Proc.	1 proc.	2 proc.	3 proc.	4 proc.	5 proc.
1	788	–	–	–	–
2	705	522	–	–	–
3	656	476	152	–	–
4	480	443	111	133	–
6	118	118	118	437	472

4 lentelė. Keliaujančio pirklio uždavinys, algoritmas be geriausio sprendinio siuntimų

Proc.	T	S	E	W_p	W_p/W
1	113,234	1	1	28360	–
2	108,913	1,039	0,519	42402	1,495
3	106,691	1,061	0,353	54693	1,928
4	106,542	1,062	0,266	70101	2,472
5	106,482	1,063	0,212	80688	2,845

5 lentelė. Keliaujančio pirklio uždavinys, algoritmas su geriausio sprendinio siuntimu

Proc.	T	S	E	W_p	W_p/W
1	113,16	1	1	28360	–
2	108,855	1,039	0,519	42460	2,079
3	106,231	1,065	0,355	45485	3,196
4	105,059	1,077	0,269	63979	4,308
5	103,874	1,071	0,214	69360	5,356

7. Išvados

Naudojant šabloną yra nesunkiai gaunama lygiagrečioji programa. Taip pat patogu eksperimentuoti su įvairiomis NUE strategijomis ir jų deriniais.

Eksperimentų rezultatai patvirtino, kad skirtingai nuo duomenų lygiagretumo, šakų ir rėžių lygiagretieji algoritmai pasižymi neprognozuojamai kintančia paieškos sritimi [6]. Algoritmo šablonas turėtų būti papildytas lygiagrečiaisiais šakų ir rėžių algoritmais su dinaminiu apkrovimo balansavimu – programos vykdymo metu užduotys iš labiau apkrautų procesorių perduodamos mažiau apkrautiems.

Literatūra

1. J. Clausen, *Branch and Bound Algorithms – Principles and Examples.* `http://vaidila.vdu.lt/tempus/courses/ mathprog/nd_bound.pdf`.
2. R. Horst, P.M. Pardalos, N.V. Thoai, Introduction to global optimization, in: *Nonconvex Optimization and its Applications*, Second Edition, Vol. 48, Kluwer Academic Publishers (2000), pp. 237–265.

3. J. Mockus, A set of examples of global and discrete optimization 2, *Home Work for Graduate Students*, 10, 36–40.
4. J. Čilinskas, *Parallel Branch and Bound for Nonlinear Global Optimization*, daktaro disertacija (2000).
5. A. Grama, A. Gupta, G. Karypis, V. Kumar, *Introduction to Parallel Computing*, Second Edition, Addison Wesley (2003).
6. C. Xu, F. Lau, *Load Balancing in Parallel Computers. Theory and Practice*. Kluwer Academic Publishers (1997).

SUMMARY

M. Baravykaitė. The temolate for parallel branch algorithm

The aim of this work is to propose parallel branch and bound algorithm template. For this purpose sequential template is introduced. Implementing additional features of this template some parallel branch and bound algorithms are comstructed and analyzed. template.

Keywords: branch and bound algorithms, optimization, parallel algorithm.