Parallel Simulated Annealing Algorithm for Graph Coloring Problem

Szymon Łukasik[1,2] Zbigniew Kokosiński[2] Grzegorz Świętoń[2]

12 września 2007

[1]Polish Academy of Sciences, Systems Research Institute
[2]Cracow University of Technology, Department of Automatic Control

Introduction

Graph Coloring Problem (GCP)

Let: $G = (V, E)$ be a given graph with n vertices V connected with m edges E
Graph Coloring Problem: Find an assignment of k colors to vertices $c : V \to \{1, \ldots, k\}$, $k \leqslant n$ such as $\forall (u, v) \in E : c(u) \neq c(v)$ and k is minimal (graph chromatic number $\chi(G)$.

GCP is one of the NP-hard problems.
Main heuristic solving methods:

- local and tabu search (Galinier & Hertz, 2006),
- genetic algorithms (Fleurent & Ferland, 1996),
- simulated annealing (Johnson et al., 1991),

+ one metaheuristics application - parallel genetic algorithm (Kokosiński et al., 2005).

Parallel Simulated Annealing

Two main concepts (Lee & Lee, 1996):

a) parallel moves (trials) - single Markov chain is being evaluated by multiple processing units calculating possible moves from one state to another,

b) multiple Markov chains - uses multiple threads for computing independent chains of solutions and exchanging the obtained results on a regular basis (synchronous or asynchronous).

Recent years brought rapid development of this technique (both in theory and practice). Some applications and improvements are:

- vehicle routing (Czech, 2001), flow shop (Wodecki & Bożejko, 2001), global optimization (Onbaşoğlu & Özdamar, 2005)
- multisteps, backtrack jumps (Bożejko & Wodecki, 2004), genetic algorithm at higher level (Tomoyuki et al., 2000).

Parallel Simulated Annealing for GCP

Proposed Algorithm


```
if proc=master
   best_cost:=infinity;
if proc=slave
   // generate initial solution randomly at each slave
   solution[proc]:=Generate_Initial_Solution();
for iter:=1 to iter_no do
   if proc=slave
      // each slave generates a new solution
      neighbor_solution[proc]:=Generate_Neighbor_Sol(solution[proc]);
      // and accepts it as a current one according to SA methodology
      solution[proc]:=Anneal(neighbor_solution[proc],solution[proc],T);
   // all solutions are then gathered at master
   Gather_at_master(solution[proc]);
   if proc=master
      current_cost:=infinity;
      // find solution with minimum cost and set it as a current one
      for j:=1 to slaves_no do
         if Cost(solution[j])<current_cost
            current_solution:=solution[j];
            current_cost:=Cost(solution[j]);
         // update best solution found (if applicable)
         if current_cost<best_cost
            best_solution:=current_solution;
            best_cost:=current_cost;
            // if stop condition fulfilled — end main loop
            if best_cost<=target_cost
                 break;
   // distribute periodically current solution among all slaves
   if iter mod e_i = 0
      solution[proc]:=Distribute(current_solution);
   // update annealing temperature if appropriate
   T:=Update_Temperature(T);
if proc=master
   return best_solution;
```

Solution representation & Cost assessment

- A graph coloring c is represented by a sequence of natural numbers $c =< c[1], \ldots, c[n] >, c[i] \in \{1, \ldots, k\}$.
- Cost of solution is determined using following cost function (Kokosiński et al., 2005):

$$f(c) = \sum_{(u,v)\in E} q(u,v) + d + k$$

where q – is a penalty function:

$$q(u,v) = \begin{cases} 2 & when\ c(u) = c(v) \\ 0 & otherwise \end{cases}$$

d – is a coefficient for solution with conflicts:

$$d = \begin{cases} 1 & when\ \sum_{(u,v)\in E} q(u,v) > 0 \\ 0 & when\ \sum_{(u,v)\in E} q(u,v) = 0 \end{cases}$$

and k – is the number of colors used.

Neighborhood Generation - Algorithm


```
// check for vertices with color conflicts
conflict_vertices:=Find_Conflicting_Vertices(c);

if sizeof(conflict_vertices)>0 do
   // conflicts were found — choose random conflicting vertex
   vertex_to_change:=random(conflict_vertices);
   // and replace its color randomly
   // with one of the colors {1, ... ,k+1}
   c[vertex_to_change]:=random(k+1);
else
   // if no conflicts are found choose random vertex
   vertex_to_change:=random(n);
   // and replace its color randomly
   // with one of the colors {1, ... ,k}
   c[vertex_to_change]:=random(k);

return c;
```

Experimental results

Testing environment

- Algorithm was implemented using `MPICH-2` library and tested on LINUX-based system.
- All experiments were carried out on one Intel® Xeon™ machine using simulated parallelism (number of iterations was inversely proportional to the number of slaves).
- As test instances standard `DIMACS` graphs were used.
- Initial temperature was determined from a pilot run consisting of 1% (relative to overall iteration number) positive transitions.
- For experiments following values of SA control parameters were chosen: $\alpha = 0.95$ and $\beta = 1.05$.
- The termination condition was either achieving the optimal solution or the required number of iterations.

Simulated Annealing parameters settings

Graph G(V,E)	Description	$P(\Delta_{cost,0})$[1] Results	$P(\Delta_{cost,0})$[1] Best P	T_f[2] Results	T_f[2] Best T_f	k_0[3] Results	k_0[3] Best k_0
anna, $\chi(G) = 11$	best f(c)	11.12	70%	11.00	$0.04 \cdot T_0$	11.12	$\chi(G)$
$\|V\| = 138$	avg. f(c)	11.32		11.14		11.17	
$\|E\| = 493$	$\sigma_{f(c)}$	0.29		0.21		0.05	
queen8_8, $\chi(G) = 9$	best f(c)	11.33	60%	10.60	$0.06 \cdot T_0$	11.33	$\chi(G)$
$\|V\| = 64$	avg. f(c)	11.46		11.84		11.41	
$\|E\| = 728$	$\sigma_{f(c)}$	0.10		1.26		0.05	
mulsol.i.4, $\chi(G) = 31$	best f(c)	38.23	80%	31.03	$0.2 \cdot T_0$	37.63	$\chi(G) - 5$
$\|V\| = 197$	avg. f(c)	38.66		33.65		38.22	
$\|E\| = 3925$	$\sigma_{f(c)}$	0.46		1.64		0.36	
myciel7, $\chi(G) = 8$	best f(c)	12.95	60%	8.00	$0.2 \cdot T_0$	11.38	$\chi(G) - 5$
$\|V\| = 191$	avg. f(c)	13.22		9.47		12.93	
$\|E\| = 2360$	$\sigma_{f(c)}$	0.34		1.60		0.96	

[1] 500 runs, $iter\_no = 10000$, $T_f = 0.1$, $k_0 = \chi(G)$

[2] 500 runs, $iter\_no = 10000$, $P(\Delta_{cost,0}) = 70\%$, $k_0 = \chi(G)$

[3] 500 runs, $iter\_no = 10000$, $P(\Delta_{cost,0}) = 70\%$, $T_f = 0.05 \cdot T_0$,

Parallel Simulated Annealing for GCP performance evaluation

Graph G(V,E)	Description	SA Results	PSA Results Best	PSA Results Worst	PSA Results Average	PSA Config. Best	PSA Config. Worst
games120	avg. f(c)	9	9	9	9	7 slaves	18 slaves
$\chi(G) = 9$	c.–f. c /opt. c	100 / 100	100 / 100	100 / 100	100 / 100	$e_i = 1$	$e_i = \infty$
$\lvert V\rvert = 120$	avg. iter. /opt. c	477	78	258	176		
$\lvert E\rvert = 638$	avg. t[s] /best c	0.72	0.05	0.40	0.14		
anna	avg. f(c)	11	11	11.32	11.02	4 slaves	18 slaves
$\chi(G) = 11$	c.–f. c /opt. c	100 / 100	100 / 100	100 / 72	100 / 98	$e_i = 1$	$e_i = 1$
$\lvert V\rvert = 138$	avg. iter. /opt. c	5821	199	1177	462		
$\lvert E\rvert = 493$	avg. t[s] /best c	1.31	0.08	1.73	0.31		
myciel7	avg. f(c)	8	8	8.66	8.05	3 slaves	18 slaves
$\chi(G) = 8$	c.–f. c /opt. c	100 / 100	100 / 100	100 / 43	100 / 95	$e_i = 1$	$e_i = 1$
$\lvert V\rvert = 191$	avg. iter. /opt. c	7376	797	1524	1539		
$\lvert E\rvert = 2360$	avg. t[s] /best c	1.85	0.20	1.83	1.03		
miles500	avg. f(c)	20	20	20.1	20.01	6 slaves	17 slaves
$\chi(G) = 20$	c.–f. c /opt. c	100 / 100	100 / 100	100 / 90	100 / 98	$e_i = 1$	$e_1 = 1$
$\lvert V\rvert = 128$	avg. iter. /opt. c	38001	544	422	2842		
$\lvert E\rvert = 1170$	avg. t[s] /best c	4.71	0.21	0.58	1.06		
mulsol.i.4	avg. f(c)	31.04	31.19	38.24	34.74	2 slaves	17 slaves
$\chi(G) = 31$	c.–f. c /opt. c	100 / 96	100 / 81	100 / 0	100 / 1	$e_i = \infty$	$e_i = 1$
$\lvert V\rvert = 197$	avg. iter. /opt. c	19007	15908	-	13451		
$\lvert E\rvert = 3925$	avg. t[s] /best c	4.30	1.83	2.64	2.08		
queen8_8	avg. f(c)	9.97	9.81	10.05	9.97	5 slaves	16 slaves
$\chi(G) = 9$	c.–f. c /opt. c	100 / 3	100 / 19	100 / 0	100 / 3	$e_i = 1$	$e_i = \infty$
$\lvert V\rvert = 64$	avg. iter. /opt. c	66488	8831	-	8820		
$\lvert E\rvert = 728$	avg. t[s] /best c	1.66	0.64	3.82	1.65		
le450_15b	avg. f(c)	18.58	17.39	21.79	18.71	9 slaves	18 slaves
$\chi(G) = 15$	c.–f. c /opt. c	100 / 0	100 / 0	100 / 0	100 / 0	$e_i = 1$	$e_i = \infty$
$\lvert V\rvert = 450$	avg. iter. /opt. c	-	-	-	-		
$\lvert E\rvert = 8169$	avg. t[s] /best c	42.88	3.54	6.47	4.99		

[1] $P(\Delta_{cost,0}) = 70\%$ and $T_f = 0.05 \cdot T_0$ for both SA and PSA. SA tested with $iter\_no = 100000$. PSA executed for 2...18 slaves with $e_i = \{1, 2, 4, 6, 8, 10, \infty\}$ and $iter\_no/slaves\_no$ iterations

Comparison with Parallel Genetic Algorithm

Graph G(V,E)	t[s]	
	PSA[1]	PGA[2]
anna, $\chi(G) = 11$ $\lvert V\rvert = 138, \lvert E\rvert = 493$	0.23	0.60
mulsol.i.4, $\chi(G) = 31$ $\lvert V\rvert = 185, \lvert E\rvert = 3946$	2.89	3.00
myciel7, $\chi(G) = 8$ $\lvert V\rvert = 191, \lvert E\rvert = 2360$	0.34	1.40
mulsol.i.1, $\chi(G) = 49$ $\lvert V\rvert = 197, \lvert E\rvert = 3925$	14.9	9.27
miles500, $\chi(G) = 20$ $\lvert V\rvert = 128, \lvert E\rvert = 1170$	0.48	18.0
queen8_8, $\chi(G) = 9$ $\lvert V\rvert = 64, \lvert E\rvert = 728$	51.8	0.87

[1] PSA: 3 slaves, $P(\Delta_{cost,0}) = 70\%, T_f = 0.05 \cdot T_0$, $e_i = 1$ and $e_i = \infty$ for $mulsol.i$

[2] PGA as in (Kokosiński et al., 2005): 3 islands, subpopulations - 60 individuals, migration rate 5, best individuals migration size 5, $k_0 = 4$ and operators: CEX crossover (with 0.6 probability), First–Fit mutation (with 0.1 probability)

Final remarks

Comments and future work

- Algorithm performance depends on chosen cooling schedule and generated initial coloring. Further research could concern adaptive cooling schedule and generating initial solution by means of an approximate method.
- There exists optimal - relatively small number of slaves - for which highest algorithm performance is observed. In most cases parallel moves method outperform multiple Markov chains strategy.
- PSA algorithm was proved to be an effective tool for solving GCP - it achieves a similar performance level as PGA. Interesting result might be obtained with hybrid PSA-PGA algorithm.

Thank you for your attention!

Bibliography

[1] Bożejko, W., Wodecki, M.: The New Concepts in Parallel Simulated Annealing Method. Proc. ICAISC'2004, LNCS **3070** (2004) 853–859

[2] Czech, Z.J.: Three Parallel Algorithms for Simulated Annealing. Proc. PPAM'2001, LNCS **2328** (2001) 210–217

[3] Fleurent, C., Ferland, J.A.: Genetic and Hybrid Algorithms for Graph Coloring. Annals of Operations Research **63** (1996) 437–461

[4] Galinier, P., Hertz, A.: A Survey of Local Search Methods for Graph Coloring. Computers & Operations Research **33** (2006) 2547–2562

[5] Johnson, D.S., Aragon, C.R., McGeoch, L., Schevon, C.: Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. Operations Research **39/3** (1991) 378–406

[6] Kokosiński, Z., Kwarciany, K., Kołodziej, M.: Efficient Graph Coloring with Parallel Genetic Algorithms. Computing and Informatics **24/2** (2005) 109–121

[7] Lee, S.-Y., Lee, K.G: Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains. IEEE Transactions on Parallel and Distributed Systems **7/10** (1996) 993–1008

[8] Onbaşoğlu, E, Özdamar, L.: Parallel Simulated Annealing Algorithms in Global Optimization. Journal of Global Optimization **19** (2005) 27–50

[9] Tomoyuki, H., Mitsunori, M., Ogura, M.: Parallel Simulated Annealing using Genetic Crossover. Science and Engineering Review of Doshisha University **41/2** (2000) 130–138

[10] Wodecki, M., Bożejko, W.: Solving the Flow Shop Problem by Parallel Simulated Annealing Proc. PPAM'2001, LNCS **2328** (2001) 236–244