SCIENTIFIC PAPERS
OF THE UNIVERSITY OF PARDUBICE
Series B
The Jan Perner Transport Faculty
8 (2002)

CONSTRAINT PROGRAMMING: AN APPLICATION FOR GRAPH COLORING

Ľudmila JÁNOŠÍKOVÁ
Radoslav STASINKA

Katedra dopravných sieti, Fakulta riadenia a informatiky, Žilinská univerzita

1. Introduction

Constraint programming is a software technology for declarative description and solution of problems with finite and discrete set of feasible solutions. Its development began in artificial intelligence in seventies. In recent years, possibilities of its applications in other areas, like operations research and computer graphics have been intensively studied.

A problem is formulated in a declarative way using variables and constraints. To find a solution, variables are instantiated in a sequential order so that every constraint is satisfied. The works published (e.g. [3, 7]) have claimed that the method is particularly suitable for solving NP-hard problems and problems with constraints, which are difficult to formalize or cannot be expressed in the form of linear equations.

In this paper, we describe a constraint programming application for the graph coloring problem. The goal is to assign colors (or any distinct marks) to vertices of the graph so that no two adjacent vertices have the same color. We can either find coloration with a given number of colors k (so called k-coloring) or with the minimum number of

colors χ (optimal coloring). The k-coloring problem is a NP-hard problem for $k < 2\chi$ and an open problem for $k \geq 2\chi$ [2]. The optimal coloring problem is NP-hard [6].

Our goal was to find out whether constraint programming is a suitable method for the graph coloring problem and which modification is the best for which types of graphs.

The graph coloring problem can be viewed as a constraint satisfaction problem. Section 2 reviews how a constraint satisfaction problem can be solved using constraint programming. Section 3 describes a constraint programming application for the k-coloring problem and the optimal coloring problem. It also presents the computational results and their analysis.

2. Constraint Satisfaction Problem and Constraint Programming

A constraint satisfaction problem (CSP) consists of

- a set of variables $X = \{x_1, ..., x_n\}$, where each variable x_i can take a value from a finite set D_i of possible values, and
- a set of constraints limiting the combinations of values that the variables can simultaneously take.

Set D_i is called the **domain** of variable x_i. The **solution** of a CSP is an assignment of values to the variables, which satisfies all constraints.

The CSP is most often solved using systematic search through the possible assignments of values to variables. A depth-first search strategy is usually applied. In one step, a partial value assignment is extended by one variable if the variable can take a value from its domain so that no constraint between this variable and already instantiated variables is violated. If a partial solution is extended to all variables, a solution is found. If no feasible value can be assigned to the current variable, the search process backtracks to the previous variable and a new value is tried. If the root of the search tree is again reached and all branches originating in the root have been explored, the problem has no solution. This way of solution space search is known as **backtracking**.

To reduce the number of explored branches in the search tree, after a value has been assigned to the current variable, future variables are made **consistent** with this variable by deleting those values from their domains, which are disallowed by constraints. We will refer to this process as **constraint propagation**.

Constraint propagation can be of a different scope. It can refer just to uninstantiated variables that share constraints with the current variable. This constraint propagation algorithm is called **forward checking**. Domain revision of an uninstantiated variable can affect domains of other uninstantiated variables that share constraints with it. Hence, constraint propagation can be expanded to other variables that do not have common constraints with the current variable. The algorithm checking all uninstantiated variables is called **look-ahead**.

If constraint propagation causes the domain of any uninstantiated variable to become empty, it is known that a given value of the current variable would lead to failure therefore another value has to be assigned. If no domain becomes empty, another uninstantiated variable is chosen and the search continues to the depth. If the next variable is a consistent one, any value from its domain can be assigned and constraints between this variable and already instantiated variables do not need be tested.

By reducing the domains of future variables we reduce the number of explored branches of the search tree. However, processing of one node is more expensive then in simple backtracking. The whole running time of algorithm with constraint propagation can be worse then without constraint propagation, although the number of explored nodes is smaller.

Another issue to consider is the order in which variables are considered for instantiation. The order can be set beforehand (static), it can change during the search (dynamic), or it can be defined using a combination of a static and a dynamic key. One of the possible keys for variable ordering is the domain size, where the variable with the smallest number of possible remaining alternatives is selected for instantiation. Another possibility is to first instantiate the variable, which participates in the largest number of constraints. The order in which variables are chosen for instantiation can have substantial impact on the search complexity [4].

3. Constraint Programming for Graph Coloring

3.1 *k*-coloring

The *k*-coloring problem can be viewed as a constraint satisfaction problem. A variable with domain {1, 2, ..., *k*} is associated with each vertex of the graph. All constraints are binary. They express that no two variables representing adjacent vertices can have the same value.

We solved this problem using several versions of the backtracking algorithm, which differ in constraint propagation scope and variable ordering:

- simple backtracking with backward check of constraints (we will refer to as BT),
- backtracking combined with a forward checking algorithm (labeled BT-FC),
- backtracking combined with a look-ahead algorithm (labeled BT-L).

We also tested several orders in which variables are chosen for instantiation. The next variable can be:

- version 1: a variable with the smallest domain size,
- version 2: a variable with the smallest domain size; the first variable is the one, that participates in most constraints (it corresponds to the highest degree vertex),
- version 3: a variable with the smallest domain size; if several variables have the same domain size, then the chosen variable is the one, that participates in most

constraints. Variables in this version are ordered according to two keys: the primary one (domain size) is dynamic, the secondary one (number of constraints) is static.

- version 4: a variable that shares most constraints with already instantiated variables (it corresponds to the vertex with the highest number of colored neighbors). If there are more variables of the same order, then the chosen variable is the one, that participates in most constraints. In this version, variables are ordered according to two keys as well: The number of already colored neighbors of the vertex represented by a given variable is a primary dynamic key, the vertex degree is a secondary static key.

The complexity of the backtracking algorithm is always exponential, because the number of search tree nodes is k^n, where n is the number of vertices of the graph.

In the area of implementation of the algorithms, most attention was paid to data structures that keep the ordered list of uninstantiated variables. Since the list is updated in each step of the backtracking algorithm, it has to be designed so that operations of adding and deleting from the list could be performed in $O(1)$ time.

If variables are ordered according to one key (see versions 1 and 2), then the list is implemented using an array of pointers to linear lists of variables. The size of the array is defined by the maximum key value, which is, in our case, the maximum domain size, i.e. the given number of colors. Each field of the array contains a pointer to a doubly-connected list of variables with the same domain size.

If variables are ordered according to two keys, then the list is implemented using a two-dimensional array of pointers to doubly-connected lists of variables with the same values of both keys. The primary key is the first index of the array, and the secondary key is the second index.

The goal of our experiments was to find out, which constraint propagation strategy and which variable ordering are the best for the k-coloring problem. We tested algorithm BT-FC with variable ordering 1, 2, 3 and 4 (we will use labels BT-FC1, BT-FC2, BT-FC3 a BT-FC4), and algorithm BT-L with variable ordering 1 and 4 (we will label BT-L1 a BT-L4). We performed experiments on a great number of various types of graphs, like general and planar graphs generated by random access, complete graphs, real planar graphs, and benchmark graphs published in literature and available on the Internet address ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color. The number of colors to be used for graph coloring was specified through an input parameter of the algorithm. The maximum number of colors was specified by the maximum vertex degree + 1, which is an upper bound of the chromatic number [5]. We measured time taken by the algorithm in order to find a solution or realize that the graph cannot be colored by the given number of colors.

The computational results for some benchmark graphs are reported in Table 1. The graph name and the number of colors *k* are tabulated in the first two columns. The third column contains the result of coloring, where the letter Y stands for 'can be colored', and the letter N stands for 'cannot be colored'. In the next columns, there are running times of the particular algorithms. Times are in milliseconds. Symbol X instead of time means that execution was interrupted after 10 minutes. If algorithms reached equal time for several colors, time is reported just once and the corresponding number of colors is indicated by an interval with lower and upper bounds (e.g. 2-6), or without upper bound, e.g. 31+.

Experiments were performed on a PC Pentium II/466 MHz.

Table *tab. 2* summarizes graph characteristics. Number of vertices, number of edges, minimum, average and maximum vertex degree, graph density and chromatic number are tabulated.

Let us analyze the results in *tab. 1* and try to define how much constraint propagation is useful. We have to compare column BT-FC1 against column BT-L1, or BT-FC4 against BT-L4. We can see that the propagation of a bigger number of constraints results in extension of the overall time of the calculation. Algorithm BT-L1 was, on average, 8.2-times slower than BT-FC1, and algorithm BT-L4 was 4.9-times worse than BT-FC4. Similar results were achieved on general and complete graphs. Therefore, we can positively say that it is better to perform constraint propagation in a limited scope. This conclusion is in accordance with proposition published in [4].

Planar graphs with 50 to 1000 vertices were colored with 3, 4 and 5 colors. All versions of the backtracking algorithm combined with constraint propagation finished their runs immediately, that is why no conclusion about the scope of constraint propagation and variable ordering can be drawn from the results on planar graphs.

Comparison with the simple backtracking algorithm is interesting. On sparse graphs like planar graphs and benchmark graphs le450_5a, le450_15a and le450_25a, the simple backtracking algorithm needs several times more time than the constraint propagation algorithms. On the contrary, on the other graphs with density over 10 %, BT finishes immediately, execution gets slower along with higher number of colors approximating the chromatic number. Good results of the constraint propagation algorithms on sparse graphs can be explained by short time needed to explore one node in the search tree. Since every vertex of the graph has few neighbors, after it has been colored (the corresponding variable has been assigned), only few constraints need to be propagated. In this case, node exploration, resulting in search tree reduction, shortens the overall running time.

As for the order in which variables are selected from the set of uninstantiated variables, the results are not as clear as the results of constraint propagation. Computational experiments indicate that the suitable strategy depends on the average

vertex degree. In *tab. 1*, we can see that the running times of the BT-FC3 and BT-FC4 algorithms are approximately the same for graphs le450_15a, le450_25a and zeroin.i.2.

Tab. 1 *Computational experiments for k-coloring*

Graph name	*k*	Result	BT	BT-FC1	BT-FC2	BT-FC3	BT-FC4	BT-L1	BT-L4
fpsol2.i.1	2-7	N	0	0-50	0-50	0-50	0-110	0-110	0-280
	8	N	0	**50**	50	440	660	280	1870
	9	N	0	**270**	330	2800	3840	2690	15100
	10	N	0	**2520**	2690	25370	30870	20710	121710
	11	N	0	27190	**26690**	240520	227670	855130	X
	65+	Y	0	110	110	110	110	380	220
inithx.i.1	2-7	N	0	0-50	0-110	0-110	0-160	0-330	0-660
	8	N	0	330	**280**	660	660	1590	3020
	9	N	60	**1930**	2190	5880	5210	11260	19060
	10	N	110	**18840**	22130	65200	43390	114130	152750
	54+	Y	50	170	170	160	160	280	280
le450_15a	2-4	N	0	0	0	0	0	0	0
	5	N	6920	0	0	0	0	0	0
	6	N	X	0	0	0	0	160	60
	7	N	X	160	110	50	**0**	1160	270
	8	N	X	2360	1750	430	**380**	14330	1870
	9	N	X	39430	26690	3350	**3130**	255460	15380
	15	Y	X	X	X	X	X	X	X
	16	Y	X	X	X	0	X	60	X
	17	Y	X	0	0	0	X	0	X
le450_25a	2-6	N	0-50	0	0	0	0	50	0
	7	N	440	0	50	**0**	60	170	110
	8	N	4170	220	550	160	**160**	1100	820
	9	N	109080	1590	6200	880	**820**	12350	4340
	10	N	X	24010	85030	6700	**6420**	177130	36470
	25+	Y	60	0	0	0	0	0-110	0-110
le450_5a	2	N	0	0	0	0	0	0	0
	3	N	2030	0	0	0	0	0	0
	4	N	X	0	0	0	0	0	0
	5	Y	X	6870	22520	**3790**	201240	41740	10710
	6-7	Y	X	X	X	X	X	X	X
	8	Y	X	X	X	**481250**	X	X	X
	9	Y	X	9610	X	**0**	X	X	37740
	10	Y	X	0	0	0	X	0	50
mulsol.i.3	2-7	N	0	0	0	0	0	0	0-110
	8	N	0	50	**50**	110	160	490	660
	9	N	0	440	**440**	880	990	3680	5500
	10	N	0	3850	**3630**	6970	8960	34320	56350
	11	N	0	35480	**34440**	65750	82000	278420	455720
	31+	Y	0	0	0	0	0	0	0
zeroin.i.1	2-7	N	0	0	0	0-50	0-60	0-110	0-110
	8	N	0	110	**60**	160	160	550	610
	9	N	0	660	**550**	1040	1160	5110	8130
	10	N	0	5770	**4830**	8730	10430	46250	57620
	11	N	0	55200	**45150**	82170	101170	439070	509490
	49+	Y	0	0	0	0	0	0-110	0-110

Tab. 2 *Graph characteristics*

	Graph name	Number of vertices	Number of edges	Minimum vertex degree	Average vertex degree	Maximum vertex degree	Graph density [%]	Chromatic number
Benchmarks	fpsol2.i.1	496	11654	0	46.99	252	9.49	65
	inithx.i.1	864	18707	0	43.3	502	5.02	54
	le450_15a	450	8168	2	36.3	99	8.09	15
	le450_25a	450	8260	2	36.71	128	8.18	25
	le450_5a	450	5714	13	25.4	42	5.66	5
	mulsol.i.3	184	3916	0	42.57	157	23.26	31
	zeroin.i.1	211	4100	0	38.86	111	18.51	49
Generated	rnd_010.col	10	24	2	4.8	8	53.33	4
	rnd_020.col	20	98	4	9.8	19	51.58	7
	rnd_030.col	30	191	4	12.73	24	43.91	8
	rnd_040.col	40	360	4	18	36	46.15	10
	rnd_050.col	50	573	3	22.92	41	46.78	12
	rnd_100.col	100	2431	3	48.62	91	49.11	X
Complete	upl_08.col	8	28	7	7	7	100	8
	upl_09.col	9	36	8	8	8	100	9
	upl_10.col	10	45	9	9	9	100	10
	upl_11.col	11	55	10	10	10	100	11
	upl_12.col	12	66	11	11	11	100	12
	upl_13.col	13	78	12	12	12	100	13

Algorithm BT-FC2 was the best one on graphs with higher average vertex degree. It means that graphs with lower average vertex degree (below 36) are worth ordering uninstantiated variables according to two keys, where the number of constraints in which the variable participates (degree of the vertex represented by the given variable) is a secondary key. The choice of the primary key has no substantial impact on the algorithm's performance. Graphs with higher average vertex degree are not worth keeping lists ordered according to two keys, but the domain size is sufficient in order to be taken into consideration. The most difficult variable participating in the most constraints will be a root of the search tree. Experiments on generated general graphs confirm this conclusion.

3.2 Optimal graph coloring

Graph coloring, which uses the minimum number of colors, is an optimization problem.

A constraint satisfaction problem is transformed to an optimization problem by adding an objective function $f(x_1, x_2, \ldots, x_n) : D_1 \times D_2 \times \ldots \times D_n \rightarrow \Re$, which has to be minimized or maximized (we will consider minimization further ahead). The optimization problem can be solved by using constraint programming in such a way that, at first, the corresponding CSP is solved, while ignoring the objective function. Let $y_1, y_2, \ldots, y_n$ be a solution. The solution space is then limited by adding the constraint $f(x_1, x_2, \ldots, x_n) < f(y_1, y_2, \ldots, y_n)$ and the new CSP is solved. The added constraint specifies that a new solution must have a better objective value than the preceding one. Then, the constraint for the objective function is updated and the search process continues. The procedure ends when no feasible solution exists. The last feasible solution is the optimal one.

To find the chromatic number of the graph, the graph coloring algorithm repeats with decreasing number of colors until a feasible solution exists. At the beginning, the number of colors was set to the maximum vertex degree + 1 for generated general graphs, to n for complete graphs, and to 4 for planar graphs. Algorithm BT-FC2 was used, because it outperforms the other constraint programming algorithms for the k-coloring problem in most cases. The results were compared to the standard backtracking algorithm for optimal graph coloring [1]. This algorithm runs on graphs with at most 100 vertices in a reasonable time (on a PC). On larger graphs, the computation takes hours.

The computational results for graphs with less than 100 vertices are summarized in *tab. 3*. They correspond to our experience with the k-coloring problem. Even for the optimal coloring problem, the standard backtracking algorithm outperforms the backtracking algorithm combined with constraint propagation on graphs with higher density. BT-FC2 is the winner on planar graphs. It runs in a fraction of a second even on a graph with 1000 vertices.

Another conclusion can be drawn from the experiments. As we can see in *tab. 1*, the backtracking algorithm combined with constraint propagation finds the solution for the number of colors approximating the chromatic number from above immediately. The number of colors, for which the computation gets slower, can be used as a more precise upper bound for the standard backtracking algorithm.

4. Conclusion

There are our experiences with application of constraint programming to the graph coloring problem described in this paper. We have examined the k-coloring problem and the optimal coloring problem. In both cases, the same results have been derived: the backtracking algorithm combined with constraint propagation significantly outperforms the simple backtracking algorithm on sparse and planar graphs. This result is very

encouraging. It confirms that constraint programming is a prospective method, which can be surprising when accomplishing short running times, even when used for solving well-known problems.

***Tab. 3** Computational experiments for optimal coloring*

Graph name	Chromatic number	Result	BT-FC2 minimal	Backtracking minimal
pl_050	4	Y	50	0
pl_060	4	Y	60	5160
pl_070	4	Y	50	29270
pl_080	4	Y	60	43840
pl_090	4	Y	50	411390
pl_100	4	Y	110	X
rnd_010	4	Y	220	0
rnd_020	7	Y	490	0
rnd_030	8	Y	600	0
rnd_040	10	Y	84310	160
rnd_050	12	Y	X	3510
upl_08	8	Y	110	0
upl_09	9	Y	60	0
upl_10	10	Y	110	0
upl_11	11	Y	610	0
upl_12	12	Y	4780	0
upl_13	13	Y	49050	0

Lektoroval:Doc. Ing. Antonín Kavička, Ph.D.

Předloženo: 11.6.2003

References

1. BROWN J. R. *Chromatic scheduling and chromatic number problems*. *Management Science* 19: 456-463, (1972).
2. GAREY M. R., JOHNSON D. S. *The complexity of near-optimal graph coloring*. *Journal of the ACM*, 23:43-49, (1976).
3. KILBY P., PROSSER P., SHAW P. *A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints*. *Constraints* 5(4): 389-414, (2000).
4. KUMAR V. *Algorithms for constraint satisfaction problems: A survey*. *AI Magazine*13(1): 32-44, (1992).
5. MCHUGH J. A. *Algorithmic Graph Theory*. Prentice-Hall, (1990).
6. PLESNÍK J. *Grafové algoritmy*. Veda, Bratislava, (1983).
7. VAN HENTENRYCK P., MICHEL L., PERRON L., RÉGIN J.-C. *Constraint programming in OPL*. In *Proceedings of the International Conference on Principles and Practice of Declarative Programming*, Paris, France, (1999).

Resumé

PROGRAMOVÁNÍ S OMEZUJÍCÍMI PODMÍNKAMI: APLIKACE PRO BARVENÍ GRAFU

Ľudmila JÁNOŠÍKOVÁ, Radoslav STASINKA

Článek se zabývá algoritmy pro barvení grafu založenými na programování s omezujícími podmínkami. Popisuje princip programování s omezujícími podmínkami a jeho implementaci na uvedený problém. Obsahuje výsledky výpočetních experimentů, které navzájem porovnávají různé modifikace algoritmu pro obarvení grafu určitým počtem barev. Na základě nejrychlejší modifikace jsme sestavili algoritmus pro obarvení grafu minimálním počtem barev. Tento algoritmus jsme porovnali s klasickým backtracking algoritmem.

Summary

CONSTRAINT PROGRAMMING: AN APPLICATION FOR GRAPH COLORING

Ľudmila JÁNOŠÍKOVÁ, Radoslav STASINKA

The paper deals with graph coloring algorithms based on constraint programming. Principles of constraint programming and its implementation for the graph coloring problem are described. Computational results are reported to compare several modifications of the algorithm for graph coloring with a given number of colors. The fastest modification forms the basis of an optimization algorithm for graph coloring with the minimum number of colors. This algorithm is compared with the standard backtracking algorithm.

Zusammenfassung

NAME IHRES ARTIKELS

Ľudmila JÁNOŠÍKOVÁ, Radoslav STASINKA

Der Artikel beschäftigt sich mit Algorithmen für die Graphanfärbung auf der Grundlage von dem Programmieren mit beschränkten Bedingungen. Er beschreibt das Prinzip des Programmierens mit beschränkten Bedingungen und seine Anwendung auf das angegebene Problem. Er enthaltet Ergebnisse der Rechnungexperimente, die verschiedene Modifikationen des Algorithmus für die Graphanfärbung mit einer bestimmten Farbanzahl miteinander vergleichen. Aufgrund der schnellsten Modifikation haben wir einen Algorithmus für die Graphanfärbung mit der minimalen Farbanzahl konstruiert. Diesen Algorithmus haben wir mit dem klassischen Backtrackingalgorithmus verglichen.