# Using Local Search to Speed Up Filtering Algorithms for Some NP-Hard Constraints

Philippe Galinier, Alain Hertz, Sandrine Paroz, and Gilles Pesant

École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montreal, Canada H3C 3A7
{philippe.galinier, alain.hertz, sandrine.paroz, gilles.pesant}@polymtl.ca

## 1 Introduction

Constraint programming relies heavily on identifying key substructures of a problem, writing down a model for it using the corresponding constraints, and solving it through powerful inference achieved by the efficient filtering algorithms behind each constraint. But sometimes these individual substructures are still too difficult to handle because we do not have any efficient filtering algorithm for them. In other words, deciding satisfiability for some substructures is NP-hard.

Besides breaking them up into smaller tractable pieces and thereby sacrificing the possibility of more global inference, a few researchers have proposed ways to preserve such substructures (e.g., [8, 3]). In this paper, we propose a new approach which relies on using a local search heuristic. Local search has been very successful at solving difficult, large-scale combinatorial problems. Applied to a particular substructure, it may quickly find some solutions, each solution acting as a witness for the variable-value pairs appearing in it. In this way, a collection of diverse solutions can offer a support for many variable-value pairs. Local search offers however no help in general to confirm those variable-value pairs that should be filtered. If only a few unsupported candidates remain, a complete method can very well be affordable to decide about them.

To illustrate this approach, we consider the `SomeDifferent` constraint which states that some pairs of variables are restricted to take different values. Richter et al. [7] have studied this substructure, and proposed a filtering algorithm for it. The `SomeDifferent` constraint can be described with the following graph coloring model. Consider a set $X = \{x_1, \cdots, x_n\}$ of variables with domains $D = \{D_1, \cdots, D_n\}$, and a graph $G = (V, E)$ with vertex set $V = \{1, \cdots, n\}$ and edge set $E$. We denote $D(U) = \bigcup_{v \in U} D_v$ for any $U \subseteq V$, and $D_v$ is called the *color set* of $v$. A $D$-coloring of $G$ is a function $c : V \rightarrow D(V)$ that assigns a color $c(v) \in D_v$ to each vertex so that $c(u) \neq c(v)$ for all edges $(u, v) \in E$. The graph $G$ is *$D$-colorable* if such an assignment exists. The *list coloring problem* is to determine if a given graph $G$ with color sets $D$ is $D$-colorable. It is NP-complete, even when restricted to interval graphs [1] or bipartite graphs [4].

Following Richter et al. [7], a *point coloring* is defined as a pair $(v, i)$ with $v \in V$ and $i \in D_v$. A point coloring $(v, i)$ is *supported* in $G$ if there exists a

$D$-coloring $c$ of $G$ with $c(v) = i$. If a point coloring $(v, i)$ is *unsupported* in $G$, then color $i$ can be suppressed from $D_v$, and we say that the point coloring $(v, i)$ can be *filtered* out from $G$. The role of our filtering algorithm is to achieve domain consistency which corresponds to finding new domains $D'_1, \cdots, D'_n$ so that $i \in D'_v$ if and only if $(v, i)$ is a supported point coloring in $G$.

## 2 Description of the Filtering Algorithm

The proposed filtering procedure follows the following three steps. First, during Step 1 (*colorability testing*), we determine if the graph $G$ is $D$-colorable. If it is not the case, the algorithm stops. Otherwise, we go to Step 2. During Step 1, we apply some preliminary reduction techniques, then decompose the graph into connected components and test the colorability of each connected component. The colorability test procedure used in Step 1 is detailed in Section 2.1. Next, during Step 2 (*marking*), we generate the largest possible set $L$ of supported point colorings in the graph. This task is achieved by applying our local search procedure to each connected component. The local search procedure is described in Section 2.2. Last, during Step 3 (*filtering*), we test each point coloring $(v, i)$ that does not belong to $L$ and determine whether it is supported. Each connected component $G_j$ is considered in turn. For each vertex $v$ in $G_j$ and each color $i$ in $D_v - L$, we build a copy $H$ of the connected component, assign in $H$ value $i$ to $v$, and apply to $H$ a colorability test similar to the one used in Step 1. If $H$ is not colorable, point coloring $(v, i)$ is filtered out in $G$.

### 2.1 A Colorability Test Procedure

The input of the colorability testing procedure is a graph $G$ and a set $D$ of domains. We first apply some simple reduction techniques in order to achieve a preliminary filtering and to remove some "superfluous" edges in the graph. For example, when the color set of a vertex $v$ contains a single color $i$, color $i$ is filtered out from the color sets of the vertices $u$ adjacent to $v$, and all edges incident to $v$ are removed. In addition, we remove edges with both endpoints having disjoint color sets. Then, we decompose the graph into connected components $G_1, \cdots, G_r$. Finally, we test the colorability of each component $G_j$. If $G_j$ is not colorable, the algorithm stops immediately because the initial graph is not colorable. In order to determine if a particular connected component $G_j$ is colorable, we first apply our tabu search procedure (called `TabuSat`, see Section 2.2). If the tabu procedure does not find a solution, we use the exact graph coloring algorithm DSATUR [6]. The list coloring problem is transformed into a graph coloring problem using the technique proposed in [7].

### 2.2 A Tabu Search Heuristic Used for Marking

The role of the `TabuSD` algorithm is, when applied to a graph $G$, to return the largest possible set of supported point colorings. It can be seen as a natural

extension of the `Tabucol` algorithm [2] that solves the classical vertex coloring problem. The solution space $S$ is the set of all functions $c : V \rightarrow D(V)$ with $c(v) \in D_v$ for all $v \in V$. Hence, a solution is not necessarily a $D$-coloring since adjacent vertices $u$ and $v$ in $G$ can have the same color. In such a situation, we say that the edge linking $u$ to $v$ is a *conflicting edge*. When `TabuSD` visits a $D$-coloring $c$ (i.e., a solution without conflicting edges), all pairs $(v, c(v))$ are introduced in a list $L$ which contains all point colorings for which we have a proof that they are supported.

Let $f_1(c)$ denote the number of conflicting edges in $c$, and let $f_2(c)$ denote the number of point colorings $(v, c(v))$ in $L$. The objective function to be minimized by `TabuSD` is defined as $f(c) = \alpha f_1(c) + f_2(c)$. Parameter $\alpha$ is initially set equal to 1 and is then adjusted every ten iterations : if the ten previous solutions were all $D$-colorings of $G$ then $\alpha$ is divided by 2; if instead they all had conflicting edges, then $\alpha$ is multiplied by 2; otherwise, $\alpha$ remains unchanged.

A neighbor solution $c' \in N(c)$ is obtained by assigning a new color $c'(v) \neq c(v)$ to a vertex $v$ so that either $v$ is adjacent to a vertex $u$ with $c(u) = c(v)$, or $(v, c(v))$ belongs to $L$. When performing such a move from $c$ to $c'$, we forbid to reassign color $c(v)$ to $v$ for $K + \lambda\sqrt{|\ N(c)\ |}$ iterations : if $(v, c(v)) \in L$, $(v, c'(v)) \notin L$, and $v$ is not adjacent to any vertex with color $c(v)$, then $K$ is uniformly chosen in the interval $[30, 40]$ and we set $\lambda = 50$; otherwise, $K$ is uniformly chosen in $[20, 30]$ and we set $\lambda = 1$.

We use a first improvement strategy. More precisely, when evaluating the solutions in $N(c)$, it may happen that a non tabu neighbor $c'$ is reached with $f(c') < f(c)$. In such a case, we stop evaluating the neighbors of $c$ and move from $c$ to $c'$. Otherwise, `TabuSD` moves from $c$ to the best non tabu neighbor. The algorithm stops as soon as $L$ contains all point colorings of $G$ or is not modified for `maxiter` iterations. The `TabuSat` procedure used during the colorability test procedure is very similar to the `TabuSD` procedure, except that it stops as soon as it finds a $D$-coloring (i.e. a solution $c$ with $f_1(c) = 0$) or after `maxiter` iterations. For our experiments, we set `maxiter` to 2000.

## 3 Computational Experiments

We evaluated our algorithm on three types of instances: real data, random graphs and graphs with a unique $D$-coloring. All tests were performed on a 2.80GHz Pentium D with 1024K cache running Linux CentOS 2.6.9. We have done five runs on each instance, and we report average results.

### 3.1 Workforce Management Data

The real life problem studied in [7] is a workforce management problem in a certain department at IBM. We are given a set of jobs with dates during which each job was to be performed, and a list of people qualified to perform these jobs. Jobs that overlap in time cannot be performed by the same person. This is a typical `SomeDifferent` situation which can be modeled by a $D$-coloring problem

where jobs are vertices of the graph, colors correspond to people, and there is an edge between two vertices if the corresponding jobs overlap in time. The color set $D_v$ of a vertex $v$ is the set of people qualified to perform $v$. In all there are 290 instances with a number $n$ of jobs varying from 20 to 300. Our results are similar to those reported [7], both in the overall behavior for satisfiable versus unsatisfiable instances and in the computing times, which never exceed 0.35s. For comparison, without using our tabu search marking procedure, computing times were on average 34 times higher, reaching 10.61s on one instance. This data set actually has very few unsupported point colorings (less than 1%) so only a few values are filtered out, if any.

### 3.2 Random Graphs

For our second test set, we considered the same random graphs as in [7]. These graphs are defined with a quadruplet $(n, p, d, max_k)$ of parameters: $n$ is the number of vertices, $p$ is the probability of having an edge between two vertices, $d = |D(V)|$ is the number of different colors, and $max_k$ is the maximum size of an individual color set. For each vertex $v$, an integer $k_v$ is uniformly chosen in the interval $[1, \cdots, max_k]$, and $D_v$ is generated by randomly choosing $k_v$ colors in the interval $[1, \cdots, d]$. The instances in [7] have $n = 20, 30, \cdots, 100$, $p = 0.1, 0.3, 0.6$, $d = 300$, and $max_k = 10, 20$. We have generated additional instances considering also $n = 200, 500$, $p = 0.9$, and $max_k = 5, 40, 80$. For the original instances, our method is much faster since the maximum cpu time reported in [7] is 608.73s while our filtering algorithm never requires more than 0.058s. For the additional instances with up to 100 vertices, the maximum cpu time is 0.2s, which can be considered as reasonable for an algorithm used to achieve domain consistency. For random graphs with 200 vertices, the maximum cpu time grows up to 1.5s, and it reaches 41.5s for $n = 500$. The increase of the cpu time is mainly due to the use of `TabuSD` which requires many iterations to find a support for almost all point colorings. Again the percentage of unsupported point colorings is very small, with a maximum of 0.6%.

### 3.3 Graphs with a Unique $D$-Coloring

Mahdian and Mahmoodian [5] have described a family of graphs with a unique $D$-coloring. More precisely, given an integer $k$, they define a graph with $3k - 2$ vertices and in which every color set $D_v$ contains exactly $k$ colors. For each vertex $v$ there is exactly one supported point coloring $(v, i)$ with $i \in D_v$, which means that $(k - 1)(3k - 2)$ point colorings can be filtered out.

These instances are difficult, mainly because `Dsatur` is required to confirm that unsupported point colorings can indeed be filtered out. Computing times exceed 100 seconds for graphs with 25 vertices (for $k = 9$), which shows the limit of applicability of our filtering algorithm. While most point colorings are shown supported with `TabuSD` for small values of $k$, `Dsatur` does the job for larger values since `TabuSD` is not able to find the unique $D$-coloring.

With the hope of generating graphs with a significant yet more realistic number of point colorings to filter out, we created some variations of these graphs by deleting a certain percentage $p$ of edges. We performed tests for $k = 5, 6, 7, 8, 9$ and $p = 0.02, 0.05, 0.1, 0.15$ and for each pair of parameters we generated ten different graphs. The results were not homogeneous. For example, while 113s are needed to solve the original graph with $k = 9$, the time needed for the graphs with $k = 9$ and $p = 0.05$ ranged from 0.32s to 182.7s. Moreover, with $p = 0.15$, no point coloring could be filtered out, which defeated our purpose.

## 4 Conclusion and Future Work

We presented a filtering algorithm for the `SomeDifferent` constraint (i.e. the list coloring problem) which combines a tabu search to quickly find a supporting solution for as many point colorings as possible, and an exact algorithm to validate or filter out the remaining point colorings. Our filtering algorithm turned out to be about as fast as the one of Richter et al. [7] when tested on data from a workforce management problem, and significantly faster for random data. In future work, we intend to test our implementation of the `SomeDifferent` constraint within a constraint programming model, in conjunction with other types of constraints, in order to measure the potential increased efficiency.

The general principles of the proposed approach are not specific to the graph coloring substructure. Indeed, the technique can be adapted to other NP-hard constraints in order to obtain a filtering procedure that enforces domain consistency. This can be done by developing mainly two specific (problem-dependent) low-level procedures: an exact algorithm and a local search procedure. We plan to investigate this approach for other NP-hard constraints.

## References


1. Miklós Biro, Mihály Hujter, and Tuza Zsolt. Precoloring extension 1. interval graphs. *Discrete Mathematics 100*, pages 267–279, 1992.
2. Alain Hertz and Dominique de Werra. Using tabu search for graph coloring. *Computing 39*, pages 345–351, 1987.
3. Irit Katriel. Expected-case analysis for delayed filtering. In *Proceedings of CP-AI-OR'06*, volume 3990 of *LNCS*, pages 119–125. Springer, 2006.
4. Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics 75*, pages 135–155, 1997.
5. Mohammad Mahdian and Ebad S. Mahmoodian. A characterization of uniquely 2-list colorable graphs. *Ars Combinatoria 51*, pages 295–305, 1999.
6. Juergen Peemoeller. A correction to brélaz's modification of brown's coloring algorithm. *Communications of the ACM 26/8*, pages 593–597, 1983.
7. Yossi Richter, Ari Freund, and Yehuda Naveh. Generalizing alldifferent: The somedifferent constraint. In *Proceedings of CP'06*, volume 4204 of *LNCS*, pages 468–483. Springer, 2006.
8. Meinolf Sellmann. Approximated consistency for knapsack constraints. In *Proceedings of CP'03*, volume 2833 of *LNCS*, pages 679–693. Springer, 2003.