Conflict Propagation and Component Recursion for Canonical Labeling

Tommi Junttila[⋆] and Petteri Kaski

Aalto University and Helsinki Institute for Information Technology
Department of Information and Computer Science
PO Box 15400, FI-00076 Aalto, Finland
{Tommi.Junttila,Petteri.Kaski}@tkk.fi

Abstract. The individualize and refine approach for computing automorphism groups and canonical forms of graphs is studied. Two new search space pruning techniques, conflict propagation based on recorded failure information and recursion over nonuniformly joined components, are presented. Experimental results show that the techniques can result in substantial decrease in both search space sizes and run times.

1 Introduction

Given as input a graph G with vertex set $\{1, 2, \ldots, n\}$, the *canonical labeling problem* asks us to compute a permutation $\kappa(G) : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that the graph $G^{\kappa(G)}$, obtained by relabeling the vertices of G with $\kappa(G)$, is independent of the labeling of the vertices in the input. Put otherwise, for any two isomorphic graphs, G and H, it is required that $G^{\kappa(G)} = H^{\kappa(H)}$.

It is currently not known whether the canonical labeling problem admits an algorithm that runs in time polynomial in n. This observation withstanding, canonical labeling tasks are recurrent in combinatorial computation, which has warranted the development of backtracking algorithms tailored for performance on practical instances, even if there are crafted instances where the running time scales exponentially in n. (We refer to [1,2,3] for a further discussion of the theoretical and practical background of the problem.) Currently the fastest algorithm implementations, such as *nauty* [4,5] and *bliss* [3], are based on the paradigm of recording the state of the search in an ordered partition of the vertices, whereby two basic operations are applied to drive the search: (i) individualization of vertices, and (ii) refinement of the ordered partition using isomorphism invariants. The standard invariant used in a refinement step is the so-called *color-degree* invariant (the invariant value at a vertex lists, for each cell of the ordered partition, the number of neighbors the vertex has in the cell).

In this paper our objective is to augment the basic paradigm of individualization and refinement with two further heuristics:

Conflict Propagation based on Recorded Failure Information. Conflict propagation is a technique encountered in many backtrack algorithms: whenever a conflicting

[⋆] Financially supported by the Academy of Finland (project 122399).

A. Marchetti-Spaccamela and M. Segal (Eds.): TAPAS 2011, LNCS 6595, pp. 151–162, 2011.

search state is encountered, one tries to propagate the conflict upwards in the search tree beyond the most recent branching point. Our implementation of conflict propagation occurs in the process of finding symmetries (automorphisms) of the input graph. In particular, whenever we discover that the current node in the search tree conflicts with (is not isomorphic to) the corresponding "first-path node" (and thus cannot produce any automorphisms), we attempt to propagate the conflict to the parent node of the current node. This propagation is carried out by (iteratively) checking the current node against recorded invariant values of the child nodes of first-path nodes that also conflicted; if the node conflicts in a different way, we can deduce that its parent node cannot be isomorphic to its corresponding first-path node, either.

Component recursion. Component recursion attempts to partition the search state into "components" that can be searched independently of each other. Here it is important to note that nontrivial components need not always exist, which sets the technique somewhat apart from the classical divide-and-conquer paradigm. In particular, one needs to be able to quickly detect the absence of nontrivial components.

An immediate notion of a component in the context of canonical labeling are the connected components of a graph. Our implementation of component recursion is based on the following notion of a nontrivial component. An ordered partition is *equitable* if its cells cannot be split further using the color-degree invariant. We say that two cells of an equitable ordered partition are *nonuniformly joined* if each vertex in one cell has both neighbors and non-neighbors in the other cell. A *nonuniform component* is a connected component in the graph with the cells as vertices, and with edges joining the cells that are nonuniformly joined. We show that nonuniform components enable a form of component recursion where canonical labeling reduces to the task of canonically labeling the nonuniform components. We also note that similar concepts have been used previously in the design of graph isomorphism algorithm with a vertex-exponential upper bound on the running time [6].

The main practical motivation for these two heuristics is that they are computationally cheap to incorporate into existing algorithm implementations, such as *bliss* [3] and *nauty* [4,5], because they rely on information that is already computed but not fully exploited by the implementations. For example, conflict propagation utilizes only information that is already computed when traversing first-path nodes in the search tree, so it merely suffices to store the information for later use. Similarly, the nonuniform components are obtained as a side-effect of executing the heuristic for selecting which cell to split in an individualization step.

Based on an implementation of the heuristics in the tool *bliss*, we find that the heuristics significantly improve the performance of *bliss* on a number of families of benchmark graphs, yet the overhead of evaluating the heuristics is negligible for all the benchmark graphs.

2 Preliminaries

A *graph* is an ordered pair $G = (V, E)$, where V is a finite set and E is a set of 2-element subsets of V. The elements of V are called *vertices* and the elements of E *edges*. We write $\mathcal{G}(V)$ for the set of all graphs with vertex set V. Throughout this paper we assume that $V = \{1, 2, \ldots, n\}$. We denote by $\mathrm{Sym}(V)$ the group of all permutations of V. The image of $x \in V$ under $\gamma \in \mathrm{Sym}(V)$ is denoted by x^γ. The composition of permutations $\gamma_1, \gamma_2 \in \mathrm{Sym}(V)$ is defined for all $x \in V$ by $x^{(\gamma_1\gamma_2)} = (x^{\gamma_1})^{\gamma_2}$. For example, in cycle notation, $(1\ 2)(2\ 3) = (1\ 3\ 2)$. A permutation acts on a subset $W \subseteq V$ by $W^\gamma = \{x^\gamma : x \in W\}$ and on a graph G by $G^\gamma = (V^\gamma, E^\gamma)$, $V^\gamma = V$, and $E^\gamma = \{\{x^\gamma, y^\gamma\} : \{x, y\} \in E\}$.

A *partition* of V is a set of nonempty pairwise disjoint subsets of V whose union is V. An *ordered partition* of V is a list $\pi = (W_1, W_2, \ldots, W_m)$ such that the set $\{W_1, W_2, \ldots, W_m\}$ is a partition of V. We write $\Pi(V)$ for the set of all ordered partitions of V. Each set W_i is called a *cell* of the partition. A partition is *discrete* if all its cells are singleton sets and *unit* if it has only one cell (the set V). An ordered partition π associates with each $x \in V$ the index $\pi(x)$ of the cell of π in which x occurs, that is, $\pi(x) = i$ if and only if $x \in W_i$. If π is discrete, the mapping $\bar{\pi} : x \mapsto \pi(x)$ is a permutation of V. Conversely, a permutation $\gamma \in \mathrm{Sym}(V)$ corresponds to the discrete ordered partition $(\{1^{\gamma^{-1}}\}, \{2^{\gamma^{-1}}\}, \ldots, \{n^{\gamma^{-1}}\})$. We identify discrete ordered partitions with permutations in this manner. For example, if $\pi = (\{3\}, \{1\}, \{2\})$, then the corresponding permutation is $\bar{\pi} = (1\ 2\ 3)$. A permutation $\gamma \in \mathrm{Sym}(V)$ acts on an ordered partition $\pi = (W_1, W_2, \ldots, W_m)$ by $\pi^\gamma = (W_1^\gamma, W_2^\gamma, \ldots, W_m^\gamma)$. In particular, if π is discrete, $\overline{\pi^\gamma} = \gamma^{-1}\bar{\pi}$.

For ordered partitions $\pi_1, \pi_2 \in \Pi(V)$, we say that π_1 is *at least as fine as* π_2 and write $\pi_1 \preceq \pi_2$ if π_2 can be obtained from π_1 by replacing zero or more times two consecutive cells with the union of the cells. If $\pi_1 \preceq \pi_2$ and $\pi_1 \neq \pi_2$, we say that π_1 is *finer than* π_2 and write $\pi_1 \prec \pi_2$. For $\pi = (W_1, ..., W_{i-1}, W_i, W_{i+1}, ..., W_m)$ and $\emptyset \neq S \subsetneq W_i$, let $\pi{\downarrow_S} = (W_1, ..., W_{i-1}, S, W_i \setminus S, W_{i+1}, ..., W_m) \prec \pi$. For $S = W_i$, let $\pi{\downarrow_S} = \pi$. Intuitively, $\pi{\downarrow_S}$ is the ordered partition obtained by "individualizing" the elements in S. If A is a union of cells of π, we denote by π^A the ordered partition of A obtained by restricting π to A.

Two graphs G_1, G_2 are *isomorphic* if there exists a permutation $\gamma \in \mathrm{Sym}(V)$ such that $G_1^\gamma = G_2$. Such a permutation γ is called an *isomorphism* of G_1 onto G_2. We write $G_1 \cong G_2$ to indicate that G_1 and G_2 are isomorphic. An isomorphism of a graph onto itself is an *automorphism*. The *automorphism group* $\mathrm{Aut}(G)$ of a graph G consists of all automorphisms of G with composition as the group operation. We extend these notions of isomorphism and automorphism to ordered tuples of objects on which $\mathrm{Sym}(V)$ acts element-wise. For example, for $G_1, G_2 \in \mathcal{G}(V)$ and $\pi_1, \pi_2 \in \Pi(V)$, we have $(G_1, \pi_1) \cong (G_2, \pi_2)$ if and only if there exists a permutation $\gamma \in \mathrm{Sym}(V)$ with $G_1^\gamma = G_2$ and $\pi_1^\gamma = \pi_2$.

2.1 Colored Graphs, Equitable Colorings, Refinement Functions

A *colored graph* is an ordered pair $(G, \pi) \in \mathcal{G}(V) \times \Pi(V)$, where π associates a "color" $\pi(x)$ with every $x \in V$. A colored graph (G, π) is *equitable* if every two vertices of the same color have the same number of adjacent vertices of each color. If G is clear from the context, we say that π is equitable.

Given a colored graph (G, π), one can attempt to refine the coloring in an isomorphism preserving way by applying a refinement function to the coloring. Formally, a function $R : \mathcal{G}(V) \times \Pi(V) \to \Pi(V)$ is a *refinement function* if for all (G, π)

$\in \mathcal{G}(V) \times \Pi(V)$ and all $\gamma \in \mathrm{Sym}(V)$ it holds that (i) $R(G, \pi) \preceq \pi$ and (ii) $R(G, \pi)^\gamma = R(G^\gamma, \pi^\gamma)$. In the rest of the paper, we always assume that $R(G, \pi)$ is equitable. The refinement function applied in tools such as *nauty* [4,5] and *bliss* [3] is a performance-wise optimized version of the following classic *coarsest equitable refinement* function.

Let (G, π) be a colored graph with $\pi = (W_1, W_2, \ldots, W_m)$. Associate with each vertex $x \in V$ the *color-degree vector* $d(G, \pi, x) = (|\{y \in W_i : \{x, y\} \in E\}| : i = 1, 2, \ldots, m)$. Each color-degree vector is thus a vector of m nonnegative integers, where the ith component of the vector gives the number of neighbors of x that have color i. The coarsest equitable refinement is obtained by repeating the following iteration until termination. Given (G, π) as input, we consider the cells $W_1, W_2, \ldots, W_m$ in order. If it holds for all the cells that the vertices in the cell have identical color-degree vectors, then (G, π) is equitable and we are done. Otherwise, let W_i be the first cell that contains vertices with differing color-degree vectors. We partition W_i to maximal cells of vertices with identical color-degree vectors, and order the cells according to lexicographic ordering of the color-degree vectors. We then replace W_i in π with the new cells, and iterate the procedure.

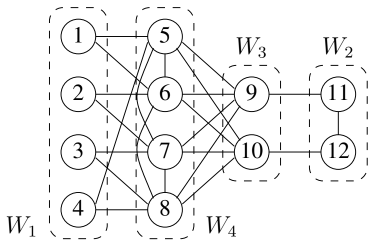


Fig. 1. A colored graph with equitable color classes (W_1, W_2, W_3, W_4)

As an example, consider the graph G shown in Fig. 1 (solid lines only). If we apply the coarsest equitable refinement function to $(G, \{1, ..., 12\})$, we obtain the equitable coloring $(\{1, 2, 3, 4\}, \{11, 12\}, \{9, 10\}, \{5, 6, 7, 8\})$ shown in the figure (dashed lines).

2.2 Individualize and Refine Depth-First Search

We now review basics of the "individualize and refine" approach for automorphism group finding and canonical labeling; for further details, see e.g. [4,5,3].

First, we need an additional concept of a *cell selector function*; it is a function S that, given a graph $G \in \mathcal{G}(V)$ and a non-discrete $\pi \in \Pi(V)$, returns a non-singleton cell in π in an isomorphism-respecting way; that is, for all $\gamma \in \mathrm{Sym}(V)$ it holds that $S(G, \pi)^\gamma = S(G^\gamma, \pi^\gamma)$. Two examples of cell selector functions are (a) the *first* cell selector: relative to the ordering of the cells, take the first non-singleton cell of π; and (b) the *maximum nonuniformly joined* cell selector: take the first non-singleton cell of π that is nonuniformly joined in (G, π) to the maximum number of cells.

In what follows we assume a fixed refinement function R and a fixed cell selector function S. The *search tree* of a graph $G \in \mathcal{G}(V)$ is the tree $\mathcal{T}(G)$ defined inductively as follows.

1. The root node of the tree is the coloring $R(G, (V))$.
2. If ρ is a node in the tree and ρ is discrete, then ρ is a leaf node.
3. If $\rho = (W_1, W_2, \ldots, W_m)$ is a node in the tree and ρ is not discrete, then it has at least two children defined as follows. Assume $S(G, \rho) = W_j = \{v_1, v_2, \ldots, v_k\}$; note that $k \geq 2$ by definition. Now ρ has exactly k children, the ith child being the

node $\rho_i = R(G, \rho\downarrow_{\{v_i\}})$. The fact that ρ_i is the child of ρ obtained by individualizing v_i and refining is denoted with $\rho[v_i\rangle\rho_i$.

As an example, a part of the search tree for the graph in Fig. 1 is shown in Fig. 2 (for each node, please ignore the ω component for now): individualizing the vertex 1 in the root node $\pi_0 = (\{1, 2, 3, 4\}, \{11, 12\}, \{9, 10\}, \{5, 6, 7, 8\})$ and refining gives the child node $\pi_1 = (\{1\}, \{2, 4\}, \{3\}, \{11, 12\}, \{9, 10\}, \{7, 8\}, \{5, 6\})$.

The fundamental property of search trees is that isomorphic graphs have isomorphic search trees. In particular, for all $\gamma \in \mathrm{Sym}(V)$ it holds that if $\pi_1[x\rangle\pi_2$ in $\mathcal{T}(G)$, then $\pi_1^\gamma[x^\gamma\rangle\pi_2^\gamma$ in $\mathcal{T}(G^\gamma)$. As a consequence, for each $\gamma \in \mathrm{Sym}(V)$, (a) if $\boldsymbol{p} = \rho_0[x_1\rangle\rho_1 \ldots [x_k\rangle\rho_k$ is a path in $\mathcal{T}(G)$, then $\boldsymbol{p}^\gamma = \rho_0^\gamma[x_1^\gamma\rangle\rho_1^\gamma \ldots [x_k^\gamma\rangle\rho_k^\gamma$ is a path in $\mathcal{T}(G^\gamma)$, (b) if ρ is a node in $\mathcal{T}(G)$, then ρ^γ is a node in $\mathcal{T}(G^\gamma)$. We say that a node π in $\mathcal{T}(G)$ is *isomorphic* to the node ρ in $\mathcal{T}(H)$ if $(G, \pi) \cong (H, \rho)$.

To find automorphisms and canonical labelings, the nodes in a tree are labeled with invariant values. A function I with domain $\mathcal{G}(V) \times \Pi(V) \times \Pi(V)$ is an *invariant* if for all $\gamma \in \mathrm{Sym}(V)$, $G \in \mathcal{G}(V)$, and $\pi_1, \pi_2 \in \Pi(V)$ with $\pi_1 \succeq \pi_2$ it holds that $I(G, \pi_1, \pi_2) = I(G^\gamma, \pi_1^\gamma, \pi_2^\gamma)$. For a path $\boldsymbol{p} = \rho_0[x_1\rangle\rho_1 \ldots [x_k\rangle\rho_k$ starting at the root of $\mathcal{T}(G)$, let $I(G, \rho_k) = I(G, \boldsymbol{p}) = (I(G, (V), \rho_0), I(G, \rho_0, \rho_1), \ldots, I(G, \rho_{k-1}, \rho_k))$. An invariant I is a *leaf certificate* if for all graphs $G, H \in \mathcal{G}(V)$ and all leaf nodes $\pi \in \mathcal{T}(G)$ and $\rho \in \mathcal{T}(H)$ it holds that $I(G, \pi) = I(H, \rho)$ if and only if $(G, \pi) \cong (H, \rho)$. As an example of a leaf certificate, recall the following from [3]. Let $S \subseteq V$ be the set of vertices that occur in singleton cells in a coloring $\pi_2 \preceq \pi_1$ but not in π_1 and let $\lambda \preceq \pi_2$ be any discrete coloring. Now $\mathcal{C}(G, \pi_1, \pi_2) = \big\{\{u^\lambda, v^\lambda\} : u \in S, \{u, v\} \in E\big\}$ is a leaf certificate whenever π_1 and π_2 are equitable (which is the case in our search trees as the refinement function always returns an equitable partition). As an example, for the search tree in Fig. 2, $\mathcal{C}(G, \pi_0, \pi_1) = \mathcal{C}(G, \pi_0, \pi_2) = \{\{1, 11\}, \{1, 12\}, \{4, 9\}, \{4, 10\}\}$; for intuition, observe that $\mathcal{C}(G, \pi_0, \pi_1)$ includes a subset of the edges in G^λ for any leaf node λ that is descendant of π_1. The leaf certificate in *bliss* is a combination of a leaf certificate of this type and an invariant value derived when evaluating the refinement function.

The search tree in *nauty* and *bliss* is traversed using two interleaved depth-first searches. (a) The "automorphism search" looks for leaf nodes ρ that have the same

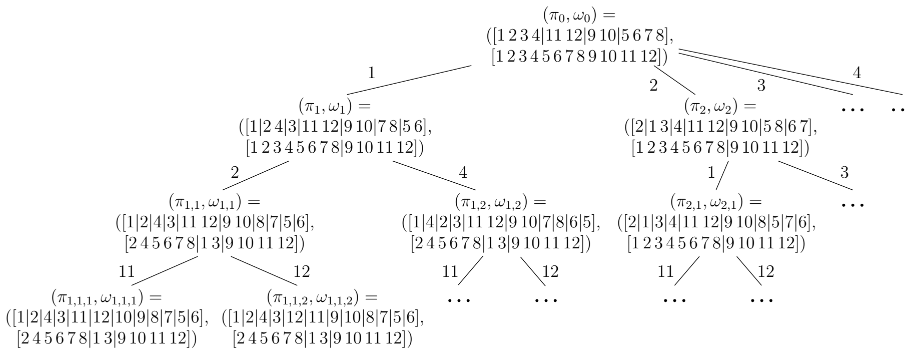


Fig. 2. A part of the search tree for the graph in Fig. 1 under the coarsest equitable refinement function and a cell selector function

certificate value as the leaf node π in the first full path (that is, a path from the root to a leaf) traversed by the search. Whenever such a leaf node ρ is found, we have discovered the automorphism $\bar{\pi}\bar{\rho}^{-1}$. Indeed, $I(G, \pi) = I(G, \rho)$ implies $(G, \pi) \cong (G, \rho)$ and thus $G^{\bar{\pi}\bar{\rho}^{-1}} = G$. For example, in the search tree in Fig. 2 we have $\mathcal{C}(G, \pi_{1,1,2}) = \mathcal{C}(G, \pi_{1,1,1})$ and thus the automorphism $(9, 10)(11, 12)$ is discovered at $\pi_{1,1,2}$. (b) The "canonical labeling search" looks for a leaf node ρ that has the lexicographically largest certificate value $I(G, \rho)$; the canonical labeling is then the permutation $\bar{\rho}$ (if $\gamma \in \mathrm{Sym}(V)$, then $\mathcal{T}(G^\gamma)$ has the node (G^γ, ρ^γ) with $I(G^\gamma, \rho^\gamma) = I(G, \rho)$ and $G^{\gamma\overline{\rho^\gamma}} = G^{\gamma\gamma^{-1}\bar{\rho}} = G^{\bar{\rho}}$). Both searches are pruned by the automorphisms discovered during the search and various other techniques [3,4].

3 Pruning with Recorded First-Path Failures

We now present our first new pruning technique that aims at reducing the search space traversed by the "automorphism search". The idea is to record some information about the children of each first-path node ν that are not isomorphic to the first-path child of ν; that is, those children that did not result in the discovery of an automorphism. This information can in some cases be used to infer that a node is not isomorphic to the corresponding first-path node and thus can be skipped together with its subtree.

Let us fix a graph $G \in \mathcal{G}(V)$ and consider the search tree $\mathcal{T}(G)$. Let I be an invariant and assume that $\boldsymbol{p} = \nu_0[x_1\rangle\nu_1 \ldots [x_l\rangle\nu_l$ is the first full path in $\mathcal{T}(G)$ traversed by the depth-first search. For each node ν_i on $\boldsymbol{p}$, define the *failing set* by $\mathit{fail}(\nu_i) = \{I(G, \rho) : \nu_i[x\rangle\rho \text{ and } (G, \rho) \ncong (G, \nu_{i+1})\}$. In other words, $\mathit{fail}(\nu_i)$ consists of the invariant values of the children of ν_i that are not isomorphic to the first-path child ν_{i+1} of ν_i. We compute the sets incrementally during the depth-first search so that if the search has backtracked above the first-path node ν_i, then $\mathit{fail}(\nu_i)$ has been fully computed. Note that it is possible to have $I(G, \nu_{i+1}) \in \mathit{fail}(\nu_i)$ as there can be another child ρ of ν_i such that $(G, \rho) \ncong (G, \nu_{i+1})$ but $I(G, \rho) = I(G, \nu_{i+1})$.

Consider the situation when the search is traversing a path $\boldsymbol{q} = \rho_0[y_1\rangle\rho_1 \ldots [y_k\rangle\rho_k$ with (a) $j + 2 \leq k \leq l$, (b) $\nu_i = \rho_i$ and $x_i = y_i$ for all $i \leq j$, (c) $I(G, \rho_i) = I(G, \nu_i)$ for all $j + 1 \leq i \leq k - 1$, and (d) $I(G, \rho_k) \neq I(G, \nu_k)$. It thus follows from (d) that $\boldsymbol{q}$ cannot be, or be extended into, a full path isomorphic to $\boldsymbol{p}$.

Now, if it also holds that $I(G, \rho_k) \notin \mathit{fail}(\nu_{k-1})$, then we can infer that (G, ρ_{k-1}) is not isomorphic to (G, ν_{k-1}) as, by the definition of fail, (G, ν_{k-1}) does not have any (failing) child with an I-value equal to $I(G, \rho_k)$. Therefore, we can skip the other children of ρ_{k-1} and backtrack to ρ_{k-2}. If ρ_{k-2} is on the first path, then we add $I(G, \rho_{k-1})$ to $\mathit{fail}(\rho_{k-2})$; note that $I(G, \rho_{k-1}) = I(G, \nu_{k-1})$ in this case. If ρ_{k-2} is not on the first path and $I(G, \rho_{k-1}) = I(G, \nu_{k-1}) \notin \mathit{fail}(\nu_{k-2})$, we can again infer that (G, ρ_{k-2}) is not isomorphic to (G, ν_{k-2}) and can thus backtrack to ρ_{k-3}. This procedure is repeated until we either (i) backtrack to a node ρ_i with $i > j$ and $I(G, \rho_{i+1}) \in \mathit{fail}(\nu_i)$ or (ii) backtrack to the node ν_j, in which case we add $I(G, \rho_{j+1})$ to $\mathit{fail}(\nu_j)$.

For example, consider the search tree in Fig. 3, where we use the symbols $\mathsf{a}, \mathsf{b}, \ldots, \mathsf{h}$ following the colon to denote invariant values at the nodes. Suppose that $(G, \pi_{1,1,1,1}) \cong (G, \pi_{1,1,1,2})$ holds. Then $\mathit{fail}(\pi_{1,1,1}) = \emptyset$. Because $I(G, \pi_{1,1,2}) = \mathsf{c} \neq \mathsf{b} = I(G, \pi_{1,1,1})$

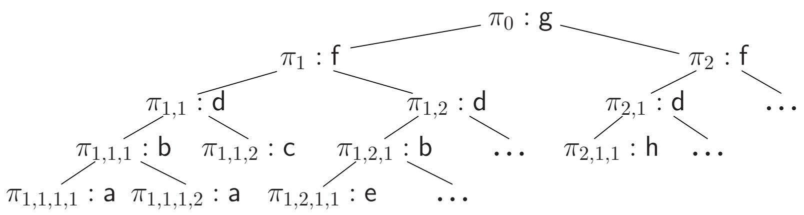


Fig. 3. An example for pruning with recorded first-path failures

it follows that $(G, \pi_{1,1,2}) \ncong (G, \pi_{1,1,1})$ and thus $\mathit{fail}(G, \pi_{1,1}) = \{\mathsf{c}\}$. Suppose we are visiting the node $\pi_{1,2,1,1}$ and observe that $I(G, \pi_{1,2,1,1}) = \mathsf{e} \notin \mathit{fail}(\pi_{1,1,1})$. It follows that $\pi_{1,2,1}$ is not isomorphic to $\pi_{1,1,1}$. Furthermore, from $I(G, \pi_{1,2,1}) = \mathsf{b} \notin \mathit{fail}(\pi_{1,1})$ it follows that $\pi_{1,2}$ is not isomorphic to $\pi_{1,1}$. We can thus backtrack directly to π_1 and add d to $\mathit{fail}(\pi_1)$. When we are visiting $\pi_{2,1,1}$, we can backtrack past $\pi_{2,1}$ as $\mathsf{h} \notin \mathit{fail}(\pi_{1,1})$ but not past π_2 as $\mathsf{d} \in \mathit{fail}(\pi_1)$. Indeed, it may be the case that $(G, \pi_{2,1}) \cong (G, \pi_{1,2})$ and thus we have to check whether there is a child of π_2 isomorphic to $\pi_{1,1}$ leading to a leaf node isomorphic to $\pi_{1,1,1,1}$.

In the current implementation of *bliss*, the invariant I is a hash value of the leaf certificate. Thus, as $\mathit{fail}(G, \nu_i)$ for each first-path node ν_i has at most n elements, this pruning technique requires a quadratic amount of memory in n in the worst-case. However, in practise we have not found a family of benchmark graphs where the memory consumption would be significant.

4 Pruning with Nonuniform Component Recursion

Let (G, π) be a colored graph with $\pi = (W_1, W_2, \ldots, W_m)$. For $1 \leq i \neq j \leq m$ we say that W_i is *uniformly joined* to W_j if either (i) every vertex in W_i is adjacent to all the vertices in W_j, or (ii) no vertex in W_i is adjacent to any vertex in W_j. Observe that this is a symmetric relation. Furthermore, observe that any two singleton cells are uniformly joined. As an example, consider the colored graph in Fig. 1. The color class W_1 is uniformly joined to all the other color classes except W_4. The color class W_3 is uniformly joined to all the other color classes except W_2.

Define an undirected graph with vertex set $W_1, W_2, \ldots, W_m$ and edges joining *nonuniformly* joined cells. This graph is called the *nonuniformity graph* of (G, π), and its connected components are called *nonuniform components*. In what follows we will identify a nonuniform component in the nonuniformity graph with the corresponding set of vertices in (G, π). That is, we say that $C \subseteq V$ is a *nonuniform component* of (G, π) if $C = \cup_{j \in J} W_j$ where $\{W_j : j \in J\}$ is the set of vertices of a connected component in the nonuniformity graph of (G, π). Observe that if C is a nonuniform component of (G, π), then C^γ is a nonuniform component in (G^γ, π^γ) for each $\gamma \in \mathrm{Sym}(V)$.

Consider again the colored graph in Fig. 1. The associated nonuniformity graph consists of the vertices $\{W_1, W_2, W_3, W_4\}$ and the edges $\{\{W_1, W_4\}, \{W_2, W_3\}\}$. The nonuniform components of the colored graph are thus $W_1 \cup W_4$ and $W_2 \cup W_3$.

The uniform join relation is *hereditary*, that is, any cells obtained by splitting uniformly joined cells remain uniformly joined. As a consequence, nonuniform components are monotone. Formally:

Lemma 1. *Let (G, π) be a colored graph and let $\sigma \preceq \pi$. If W and Z are uniformly joined cells in (G, π), then for all cells $X \subseteq W$ and $Y \subseteq Z$ in σ it holds that X and Y are uniformly joined in (G, σ). Furthermore, every nonuniform component of (G, σ) is a subset of a nonuniform component of (G, π).*

A central property of nonuniform components is that the automorphism group of a colored graph is the direct product of the automorphism groups of the colored subgraphs induced by the nonuniform components.

Lemma 2. *Let C be a nonuniform component of a colored graph (G, π). If α is an automorphism of (G, π), then so is β defined by (i) $\beta(v) = \alpha(v)$ when $v \in C$, and (ii) $\beta(v) = v$ when $v \in V \setminus C$.*

We now show how to exploit the nonuniform components to prune the search space. The idea is to traverse the components one by one and recursively; this allows us to use the component boundaries for earlier detection of automorphisms and improvements in the canonical labeling.

First, we have to redefine cell selector functions to be component sensitive. In what follows, a *cell selector function* is a function S that, given a graph $G \in \mathcal{G}(V)$, an ordered partition π of V, and a union U of some non-trivial nonuniform components of (G, π), returns a non-singleton cell in π that is a subset of U in a way that $S(G, \pi, U)^\gamma = S(G^\gamma, \pi^\gamma, U^\gamma)$ for each $\gamma \in \mathrm{Sym}(V)$. A cell selector function S *factors over nonuniform components* if in addition it holds that $S(G, \pi, U) = S(G, \sigma, U)$ for all $\sigma \preceq \pi$ such that $\sigma^U = \pi^U$. That is, the components outside of U do not influence the cell selection. Two examples of cell selector functions are (a) the *first* cell selector: relative to the ordering of the cells, we take the first non-singleton cell of π in U; and (b) the *maximum nonuniformly joined* cell selector: we take the first non-singleton cell of π in U that is nonuniformly joined in (G, π) to the maximum number of cells. Both of these selectors factor over nonuniform components.

We now redefine search trees so that the nodes carry additional information on the components. A node in the tree will now be a pair (π, ω), where $\pi \in \Pi(V)$ is a coloring as earlier and $\omega \in \Pi(V)$ is a *component stack*; it is required that each nonuniform component of (G, π) is a subset of a cell in ω. The search tree $\mathcal{T}(G)$ of G is now redefined inductively as follows:

1. The root node of the tree is the pair $(R(G, (V)), (V))$.
2. If (π, ω) is a node in the tree and π is discrete, then (π, ω) is a leaf node.
3. If $N = (\pi, \omega)$ is a node in the tree and π is not discrete, then N has at least two children defined as follows. Let V_j be the first cell in ω such that the induced coloring π^{V_j} is not discrete. Assume $S(G, \pi, V_j) = W = \{v_1, v_2, \ldots, v_k\}$ and let $C \subseteq V_j$ be the nonuniform component of (G, π) that contains W. We say that C is *opened* at (π, ω). Now N has exactly k children, the ith child being the node $N_i = (R(G, \pi\downarrow_{\{v_i\}}), \omega\downarrow_C)$. The fact that N_i is the child of N obtained by individualizing v_i is denoted with $N[v_i\rangle N_i$.

The fact that the tree is well-defined, that is, the requirements on component stacks are fulfilled, follows quite directly by recalling the monotonicity property of nonuniform components (Lemma 1). Symmetry properties similar to those on the basic search trees hold because $(\pi_1, \omega_1)[x\rangle(\pi_2, \omega_2)$ in $\mathcal{T}(G)$ if and only if $(\pi_1^\gamma, \omega_1^\gamma)[x^\gamma\rangle(\pi_2{}^\gamma, \omega_2{}^\gamma)$ in $\mathcal{T}(G^\gamma)$ for each $\gamma \in \mathrm{Sym}(V)$.

As an example, consider the search tree shown in Fig. 2 for the graph in Fig. 1 but now including the ω partitions. The vertex 1 belongs to the nonuniform component $C_0 = \{1, 2, \ldots, 8\}$ of (G, π_0) and thus ω_1 includes it before the other component $\{9, \ldots, 12\}$. Now the vertex 2 belongs to the component $C_1 = \{2, 4, 5, 6, 7, 8\} \subseteq C_0$ of (G, π_1) and thus $(\pi_1, \omega_1)[2\rangle(\pi_{1,1}, \omega_{1,1})$ with $\omega_{1,1}$ further refining the component C_1 into $\{2, 4, 5, 6, 7, 8\}$. In $(\pi_{1,1}, \omega_{1,1})$ the component C_0 is discrete and the search then focuses on the other component $\{9, \ldots, 12\}$ of (G, π_0).

We say that a refinement function R is *closed* if $R(G, \pi) = R(G, R(G, \pi))$ holds for all colored graphs (G, π). Note that an arbitrary refinement function can be transformed into a closed function by iterating the function at most $n - 1$ times until a fixed point is reached. A colored graph (G, π) is *R-stable* if $R(G, \pi) = \pi$. Observe that if R is closed, then $(G, R(G, \pi))$ is always R-stable. A refinement function R *factors over uniform components* if for all R-stable colored graphs (G, π), for all nonuniform components C of (G, π), and for all $\sigma, \tau \preceq \pi$ it holds that $\sigma^C = \tau^C$ implies $R(G, \sigma)^C = R(G, \tau)^C$. Observe that if R factors over uniform components, (G, π) is R-stable, C is a nonuniform component in (G, π), and $\sigma \preceq \pi$ with $\sigma^C = \pi^C$, then $R(G, \sigma)^C = R(G, \pi)^C = \pi^C$. That is, if we split some cells of an R-stable coloring and then refine the obtained partition, then only the nonuniform components with splits get refined and the refinement does not depend on the other components. The coarsest equitable refinement function is closed and factors over nonuniform components.

In what follows we assume a fixed cell selector function S and a fixed, closed refinement function R, both of which factor over nonuniform components.

Let $(\pi, \omega)[x_1\rangle(\pi_1, \omega_1) \ldots [x_k\rangle(\pi_k, \omega_k)$ be a full path in $\mathcal{T}(G)$ and let C be the component opened at (G, π). We say that C is *closed* at the first node (π_i, ω_i) in which π_i^C is discrete. As an example, the component $\{1, 2, \ldots, 8\}$ opened at π_0 and the component $\{2, 4, 5, 6, 7, 8\}$ opened at π_1 are both closed at $\pi_{1,2}$ in the search tree in Fig. 2. Furthermore, starting from any non-leaf node in the tree, the nonuniform component opened at the node is closed before any other components are refined at all:

Lemma 3 (Localization). *Let $(\pi, \omega)[x_1\rangle(\pi_1, \omega_1) \ldots [x_k\rangle(\pi_k, \omega_k)$ be a path in $\mathcal{T}(G)$ and let C be the nonuniform component opened at (π, ω). Then, for all $1 \leq i \leq k$ it holds that either (i) π_i^C is discrete (in which case π_j^C is discrete and $x_j \notin C$ for all $i < j \leq k$), or (ii) $x_i \in C$, $\pi_i^{V \setminus C} = \pi^{V \setminus C}$, and $\omega_i^{V \setminus C} = \omega_1^{V \setminus C}$.*

Furthermore, paths operating on different components are switchable as follows:

Lemma 4 (Switching). *Let $\boldsymbol{p} = (\pi, \omega)[x_{\mathrm{p},1}\rangle(\pi_{\mathrm{p},1}, \omega_{\mathrm{p},1}) \ldots [x_{\mathrm{p},k}\rangle(\pi_{\mathrm{p},k}, \omega_{\mathrm{p},k})$ and $\boldsymbol{q} = (\pi, \omega)[x_{\mathrm{q},1}\rangle(\pi_{\mathrm{q},1}, \omega_{\mathrm{q},1}) \ldots [x_{\mathrm{q},m}\rangle(\pi_{\mathrm{q},m}, \omega_{\mathrm{q},m})$ be two paths in $\mathcal{T}(G)$ such that the nonuniform component C opened at (π, ω) is closed at both $(\pi_{\mathrm{p},k}, \omega_{\mathrm{p},k})$ and $(\pi_{\mathrm{q},m}, \omega_{\mathrm{q},m})$. Then $\boldsymbol{p}[y_1\rangle(\pi_{\mathrm{r},1}, \omega_{\mathrm{r},1}) \ldots [y_h\rangle(\pi_{\mathrm{r},h}, \omega_{\mathrm{r},h})$ is a path in $\mathcal{T}(G)$ if and only if $\boldsymbol{q}[y_1\rangle(\pi_{\mathrm{s},1}, \omega_{\mathrm{s},1})$*

$\ldots[y_h\rangle(\pi_{\mathrm{s},h},\omega_{\mathrm{s},h})$ is a path in $\mathcal{T}(G)$ with $\pi_{\mathrm{r},i}^{V\setminus C} = \pi_{\mathrm{s},i}^{V\setminus C}$ and $\omega_{\mathrm{r},i}^{V\setminus C} = \omega_{\mathrm{s},i}^{V\setminus C}$ for all $1 \le i \le h$.

For compactness in what follows we will omit the recursion stack components from the notation. In what follows let $\boldsymbol{p} = \pi_0[x_1\rangle\ldots[x_i\rangle\pi_i[x_{i+1}\rangle\pi_{i+1}\ldots[x_k\rangle\pi_k$ and $\boldsymbol{q} = \pi_0[x_1\rangle\ldots[x_i\rangle\pi_i[y_{i+1}\rangle\rho_{i+1}\ldots[y_m\rangle\rho_m$ be rooted paths in $\mathcal{T}(G)$ such that the nonuniform component C of (G,π_i) opened at π_i is closed both at π_k and ρ_m.

An invariant *I factors over nonuniform components* if, informally, its value depends only on the refined components. Formally, we require that for all R-stable colored graphs (G,π), all nonuniform components C of (G,π), all $\rho \preceq \pi$ with $\rho^C = \pi^C$, and all R-stable $\sigma \preceq \pi$ with $\sigma^{V\setminus C} = \pi^{V\setminus C}$, it must hold that $I(G,\pi,\rho) = I(G,\sigma,\tau)$ for the coloring $\tau \preceq \pi$ with $\tau^C = \sigma^C$ and $\tau^{V\setminus C} = \rho^{V\setminus C}$. The leaf certificate $\mathcal{C}$ defined in Sect. 2.2 factors over nonuniform components. If I is a also a leaf certificate, from Lemmas 3 and 4 it follows that $I(G,\boldsymbol{p}) = I(G,\boldsymbol{q})$ if and only if $(G,\pi_k) \cong (G,\rho_m)$. (To establish the "only if" direction, consider any extension of $\boldsymbol{p}$ to a full path, and apply Lemma 4 to obtain a full path extending $\boldsymbol{q}$. Suppose κ and λ are the leaf nodes at the ends of these paths. Then $\alpha = \bar{\kappa}\bar{\lambda}^{-1} \in \mathrm{Aut}(G,\pi_i)$ satisfies $\pi_k^\alpha = \rho_m$. Below we assume that I is a leaf certificate that factors over nonuniform components.

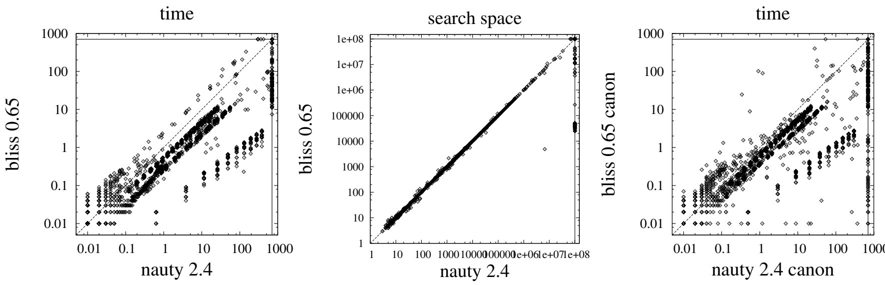
Early automorphism detection. We can apply the previous observation as follows. (a) If $\boldsymbol{p}$ is a prefix of the first path and $\boldsymbol{q}$ is the current path traversed in the "automorphism search" with $I(G,\boldsymbol{p}) = I(G,\boldsymbol{q})$, then we have found the automorphism $\alpha = \bar{\kappa}\bar{\lambda}^{-1}$. We can now report α, skip the sub-tree rooted at ρ_m, backtrack the "automorphism search" to π_i, and consider the next sibling of the child ρ_{i+1}. (b) Similarly, if $\boldsymbol{p}$ is a prefix of the current best path and $\boldsymbol{q}$ is the current path traversed in the "canonical labeling search" with $I(G,\boldsymbol{p}) = I(G,\boldsymbol{q})$, we can skip the subtree rooted at ρ_m, backtrack the search to π_i, and consider the next sibling of ρ_{i+1}.

For example, in the search tree in Fig. 2 we have that $(G,\pi_{1,1}) \cong (G,\pi_{1,2})$. Therefore, the "automorphism search" can skip the subtree rooted at $\pi_{1,2}$ and report the found the automorphism $(2,4)(5,6)(7,8)$ of G.

Early best path improvement detection. When the applied leaf certificate function I factors over nonuniform components, we can use Lemma 4 to get further pruning. If $\boldsymbol{p}$ is a prefix of the current best path $\boldsymbol{p}[z_1\rangle\kappa_1\ldots[z_l\rangle\kappa_l$, $\boldsymbol{q}$ is the current path traversed in the "canonical labeling search", and $I(G,\boldsymbol{q}) > I(G,\boldsymbol{p})$, then, by applying Lemma 4, $\boldsymbol{r} = \boldsymbol{q}[z_1\rangle\nu_1\ldots[z_l\rangle\nu_l$ with $\nu_i^{V\setminus C} = \kappa_i^{V\setminus C}$ for all $1 \le i \le l$ and $I(G,\nu_l) > I(G,\kappa_l)$ is the best path in $\mathcal{T}(G)$ visiting the node ρ_m and that $I(G,\nu_{i-1},\nu_i) = I(G,\kappa_{i-1},\kappa_i)$ (define $\nu_0 = \kappa_0 = \pi_k$) for all $1 \le i \le l$. Therefore, the "canonical labeling search" does not have to traverse the sub-tree rooted at ρ_m but can set $\boldsymbol{r}$ as the new best path and ν_l as the new candidate for the canonical labeling, backtrack one level, and consider the next sibling of ρ_m. As an example, if the first path in the search tree in Fig. 2 is the best path found so far, the "canonical labeling search" is traversing the node $\pi_{2,1}$, and $\mathcal{C}(G,\pi_0[1\rangle\pi_1[2\rangle\pi_{1,1}) < \mathcal{C}(G,\pi_0[2\rangle\pi_2[1\rangle\pi_{2,1})$, then the search can skip the sub-tree rooted at $\pi_{2,1}$ and set the canonical labeling candidate to $\bar{\nu}$, where $\nu = (\{2\},\{1\},\{3\},\{4\},\{11\},\{12\},\{10\},\{9\},\{8\},\{5\},\{7\},\{6\})$, and the best path certificate to $(\mathcal{C}(G,[V],\pi_0),\mathcal{C}(G,\pi_0,\pi_2),\mathcal{C}(G,\pi_2,\pi_{2,1}),\mathcal{C}(G,\pi_{1,1},\pi_{1,1,1}))$.

5 Experimental Evaluation of the Pruning Techniques

As the benchmark set of graphs we use the collection of graphs downloadable at the *bliss* web site ⟨`http://www.tcs.hut.fi/Software/bliss`⟩. The experimental version 0.65 of *bliss* used here, as well as some more detailed result graphs, are available at ⟨`http://users.ics.tkk.fi/tjunttil/experiments/TAPAS2011`⟩. To see that our base line (*bliss* version 0.65 *without* the new pruning techniques but with the ones in [3]) is comparable to state-of-the-art, consult Fig. 4 showing a comparison to *nauty* version 2.4 ⟨`http://cs.anu.edu.au/~bdm/nauty/`⟩ on the same benchmark set of graphs. In all experiments, we permute and run each benchmark twice and use time limit of ten minutes; the timed-out runs are plotted on the lines at 700 seconds (time plots) and 10^8 nodes (search space plots). Due to space limits and to the fact that our purpose is to evaluate the proposed pruning techniques, we omit the comparison to a similar tool *saucy* [7] (no canonical labeling, only automorphism group computation) and to *traces* [8] (which uses a mixed depth-first/breadth-first search instead of depth-first and also considers a computationally more intensive refinement function).

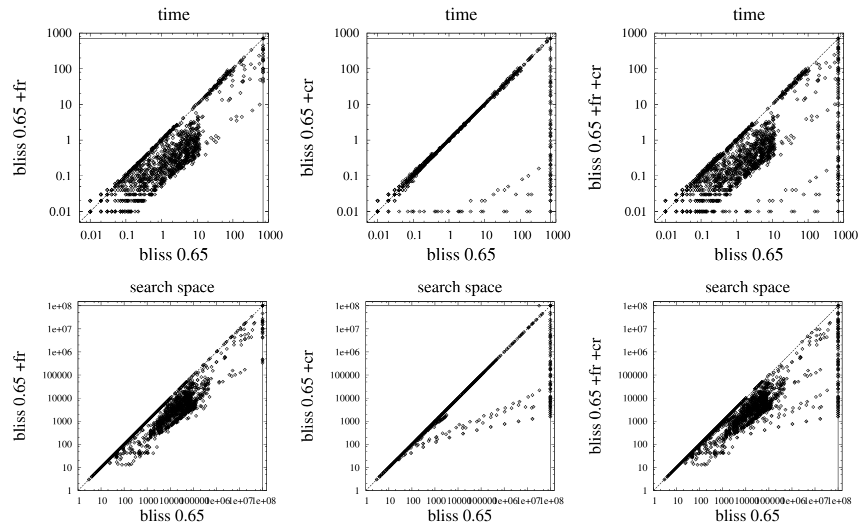


(a) automorphism group computation only (b) automorphism group computation only (c) canonical labeling and automorphism group

Fig. 4. Base comparison to *nauty* version 2.4 (using sparse representation of graphs)

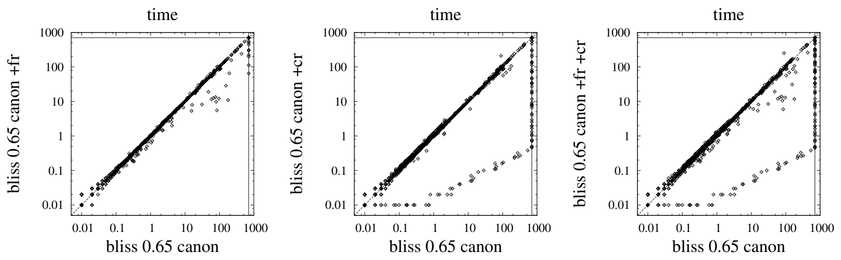
Fig. 5 shows the results of activating the proposed pruning techniques one-by-one. Here we consider only automorphism group computation, i.e. the "canonical labeling search" is not run. We see that the failure recording technique provides up to one order of magnitude run-time and search space size improvement on some families (including Hadamard matrix and Steiner triple system graphs). The component recursion technique only produces pruning on some graph families but when it does, the reduction in search space and run time is substantial. Fig. 6 shows the results when canonical labeling computation is enabled as well. As failure recording can only deduce non-isomorphism, not that a subtree contains only paths that provide worse canonical labelings, it does not help in pruning the "canonical labeling search". The effect of of the component recursion is as before; when it helps, the reduction is substantial.

To sum up, the proposed search space pruning techniques can provide substantial reduction in both search spaces and in run times. And, equally importantly, the proposed techniques do not significantly increase run time when they cannot produce any search space pruning.



(a) with failure recording (b) with component recursion (c) with both.

Fig. 5. Effect of the proposed techniques on automorphism group computation



(a) with failure recording (b) with component recursion (c) with both.

Fig. 6. Effect of the proposed techniques on canonical labeling

References

1. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: Proc. STOC 1983, pp. 171–183. ACM, New York (1983)
2. Babai, L., Codenotti, P.: Isomorhism of hypergraphs of low rank in moderately exponential time. In: Proc. FOCS 2008, pp. 667–676. IEEE, Los Alamitos (2008)
3. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proc. ALENEX 2007. SIAM, Philadelphia (2007)
4. McKay, B.D.: Practical graph isomorphism. Congressus Numerantium 30, 45–87 (1981)
5. McKay, B.D.: Nauty user's guide (version 2.4). Technical report, Department of Computer Science, Australian National University (2009)
6. Goldberg, M.K.: A nonfactorial algorithm for testing isomorphism of two graphs. Discrete Applied Mathematics 6, 229–236 (1983)
7. Darga, P.T., Sakallah, K.A., Markov, I.L.: Faster symmetry discovery using sparsity of symmetries. In: Proc. DAC 2008, pp. 149–154. ACM, New York (2008)
8. Piperno, A.: Search space contraction in canonical labeling of graphs (preliminary version). CoRR report abs/0804.4881, arXiv.org (2008), `http://arxiv.org/abs/0804.4881`