Canonical Labelling of Site Graphs

Nicolas Oury[1]
School of Informatics[1]
Edinburgh University
Edinburgh, Scotland

Michael Pedersen[2,3]
Department of Plant Sciences[2]
Cambridge University
Cambridge, UK

Rasmus Petersen[3]
Microsoft Research[3]
Cambridge, UK

We investigate algorithms for canonical labelling of site graphs, i.e. graphs in which edges bind vertices on sites with locally unique names. We first show that the problem of canonical labelling of site graphs reduces to the problem of canonical labelling of graphs with edge colourings. We then present two canonical labelling algorithms based on edge enumeration, and a third based on an extension of Hopcroft's partition refinement algorithm. All run in quadratic worst case time individually. However, one of the edge enumeration algorithms runs in sub-quadratic time for graphs with "many" automorphisms, and the partition refinement algorithm runs in sub-quadratic time for graphs with "few" bisimulation equivalences. This suite of algorithms was chosen based on the expectation that graphs fall in one of those two categories. If that is the case, a combined algorithm runs in sub-quadratic worst case time. Whether this expectation is reasonable remains an interesting open problem.

1 Introduction

Graphs are widely used for modelling in biology. This paper focuses on graphs for modelling protein complexes: vertices correspond to proteins, and edges correspond to bindings. Moreover, vertices are labelled by protein names, and edges connect vertices on labelled *sites*, giving rise to a notion of site graphs. Importantly, site labels can be assumed to be unique within vertices, i.e. a protein can have at most one site of a given name. This uniqueness assumption introduces a level of rigidity which can be exploited in algorithms on site graphs. Rigidity is for example crucial for containing the computational complexity of one algorithm for stochastic simulation of rule-based models of biochemical signalling pathways [3].

In this paper we investigate how rigidity can be exploited in the design of efficient algorithms for canonical labelling of site graphs. Informally, a canonical labelling procedure must satisfy that the canonical labellings of two graphs are identical if and only if the two graphs are isomorphic. A graph isomorphism is understood in the usual sense of being an edge-preserving bijection, with the additional requirement that vertex and site labellings are also preserved. Hence two site graphs are isomorphic exactly when they represent protein complexes belonging to the same species. The graph isomorphism problem on general graphs is hard: it is not known to be solvable in polynomial time, but curiously is not known to be NP-complete either even though it is in NP [10]. The canonical labelling problem clearly reduces to the graph isomorphism problem, but there is no clear reduction the other way. In this sense the canonical labelling problem is harder than the graph isomorphism problem.

Efficient canonical labelling of site graphs has an application in a second algorithm for stochastic simulation of rule-based languages [7]. This algorithm at frequent intervals determines isomorphism of a site graph, representing a newly created species, with a potentially large number of site graphs representing all other species in the system at a given point in time. The algorithm can hence compute a canonical labelling when a new species is first created, and subsequently determine isomorphism quickly by checking equality between this and existing canonical labellings.

Ion Petre (Ed.): Fourth International Workshop on
Computational Models for Cell Processes (CompMod 2013)
EPTCS 116, 2013, pp. 13–28, doi:10.4204/EPTCS.116.3

We assume in this paper that the number of site labels and the degree of vertices in site graphs are bounded, i.e. they do not grow asymptotically with the number of vertices, which indeed appears to be the case in the biological setting. An algorithm for site graph *isomorphism* is presented in [9] and has a worst-case time complexity of $O(|V|^2)$, where V is the set of vertices. In this paper we exploit similar ideas, based on site uniqueness, in the design of new algorithms for *canonical labelling,* which also have a worst-case time complexity of $O(|V|^2)$. We furthermore characterise the graphs for which lower complexity bounds are possible: if the bisimulation equivalence classes are "small", meaning $O(1)$, or if the automorphism classes are "large", meaning $O(|V|)$, then an $O(|V| \cdot \log|V|)$ time complexity can be achieved in worst and average case, respectively.

Site graphs can be encoded in a simpler, more standard notion of graphs with edge-colourings. We formally define these graphs in Section 2, along with the notion of canonical labelling and other preliminaries. In Section 3 we specify two canonical labelling algorithms based on ordered edge enumerations, and in Section 4 we show how these algorithms can be improved using partition refinement. We conclude in Section 5.

2 Preliminaries

2.1 Notation

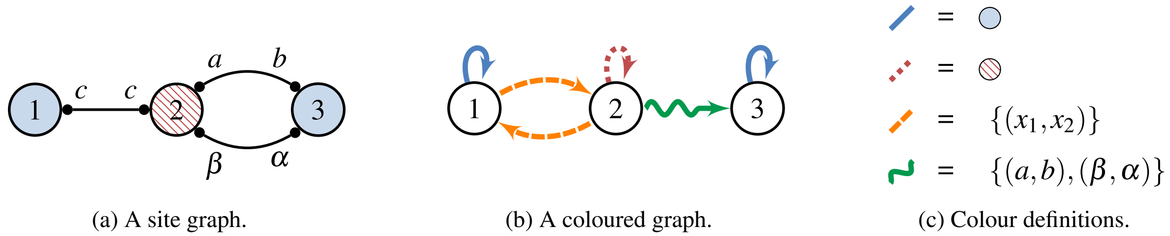
We write $X \to Y$ (respectively $X \to_{\mathsf{bij}} Y$) for the set of total functions (respectively total bijective functions) from X to Y. We use the standard notation $\prod x \in X . T(x)$ for dependent products, i.e. the set of total functions which map an $x \in X$ to some $y \in T(x)$. We write $\mathrm{Dom}(f)$ and $\mathrm{Im}(f)$ for the domain of definition and image of f, respectively, and $f \downarrow Z$ for the restriction of f to the domain Z. We view functions as sets of pairs $(x \mapsto y)$ when notationally convenient. We use standard notation for finite multisets, and use the brackets $\{\!|\cdot|\!\}$ for multiset comprehension. We write $\mathscr{T}(X)$ for the set of total orders on X. We write X^* for the Kleene closure of X, i.e. the set of finite strings over the symbols in X. Given a linearly ordered set $(X,<)$ we assume the lexicographic extension of $<$ to pairs and lists over X. Given a partially ordered set $(X,<)$ we write $\min_<(X)$ for the set of minimal elements of X under $<$, and if the ordering is total we identify $\min_<(X)$ with its unique least element. Given an equivalence relation ρ on a set X and $x \in X$, we write $[x]_\rho$ for the equivalence class of x under ρ, and we write X'/ρ for the partition of $X' \subseteq X$ under ρ. Finally, given a list x, we write $x.i$ for the ith element of x starting from 1.

2.2 Site Graphs and Coloured Graphs

A site graph is a multi-graph with vertices labelled by protein names and edges which connect vertices on *sites* labelled by site names. An example is shown in Figure 2.1a where protein names are indicated by colours, and a formal definition is given in Appendix A, Definition 16. From an algorithmic perspective, the key property of site graphs is their *rigidity*: a vertex can have at most one site of a given name, so a site name uniquely identifies an adjacent edge. This rigidity property can be captured by the simpler, more standard notion of directed graphs with edge-colourings. An example is shown in Figure 2.1b, and the formal definition follows below.

Definition 1. Let $(\Sigma, \prec)$ be a given linearly ordered set of edge colours. Then $\mathscr{CG}$ is the set of all *edge coloured graphs* $G = (V, E, \phi)$ where:

- $V = \{1, \dots, k\}$ is a set of vertices.
- $E \subseteq V \times V$ is a set of directed edges.



(a) A site graph. (b) A coloured graph. (c) Colour definitions.

Figure 2.1: An example of a site graph (a), its encoding as a coloured graph (b), and the formal definition of colours (c) following the encoding in Appendix A with the site name ordering $a \preceq_s b \preceq_s c \preceq_s \alpha \preceq_s \beta$. The direction of e.g. the edge $(2,3)$ is determined by $a \preceq_s b$ and (a,b) being the minimum pair of connected sites between vertices 2 and 3.

- $\phi : E \to \Sigma$ is an edge colouring satisfying for all $v \in V$ that $\phi \downarrow \{(v,v') \in E\}$ and $\phi \downarrow \{(v',v) \in E\}$ are injective.

Given a coloured graph G, we write V_G, E_G and ϕ_G for the vertices, edges and edge colouring of G, respectively. We write $\mathrm{Adj}(v)$, respectively $\mathrm{Adj}^{-1}(v)$, for the set of all outgoing, respectively incoming, edges adjacent to v. We measure the size of graphs as the sum of the number of edges and vertices, i.e. $|G| \stackrel{\Delta}{\simeq} |V_G| + |E_G|$. Since we assume the degree of vertices to be bounded, we have that $O(|G|) = O(|V_G|) = O(|E_G|)$.

The condition on edge colours states that a vertex can have at most one incident edge of a given colour, thus capturing the rigidity property. We give an encoding of site graphs into coloured graphs in Appendix A. The encoding is injective, respects isomorphism, and is linear in size and time (Proposition 18 in Appendix A). It hence constitutes a reduction from the site graph isomorphism problem to the coloured graph isomorphism problem. This allows us to focus exclusively on coloured graphs in the following. Since we are interested in protein complexes, we furthermore assume that the underlying undirected graphs are always connected. Figure 2.1b is in fact an example encoding of Figure 2.1a, with the specific choice of colours used by the encoding shown in Figure 2.1c.

2.3 Isomorphism and Canonical Labelling

Our notion of coloured graph isomorphism is standard. In addition to preserving edges, isomorphisms must preserve edge colours.

Definition 2. Let G and G' be coloured graphs. An *isomorphism* is a bijective function $\sigma : V_G \to_{\mathrm{bij}} V_{G'}$ satisfying that $\sigma(G) = G'$, i.e.:

- $\forall v_1, v_2 \in V_G. [(v, v_2) \in E_G \Leftrightarrow (\sigma(v_1), \sigma(v_2)) \in E_{G'}]$.
- $\forall e \in E_G. \phi_G(e) = \phi_{G'}(\sigma(e))$.

Furthermore define $G \simeq G'$ (G and G' are *isomorphic)* iff there exists an σ relating G and G', and define $(G,v) \simeq (G',v')$ (or simply $v \simeq v'$) iff there exists an σ s.t. $\sigma(v) = v'$. We denote by $I(G,G')$ the set of isomorphisms from G to G'. An isomorphism from G to itself is called an *automorphism*.

Since all graphs of the same size have the same vertices, all isomorphisms are in fact automorphisms. We next define our notion of canonical labelling. Having vertices given by integers allows the canonical labelling of a graph to be a graph itself.

Definition 3. A *canonical labeller* is a function $L : \prod G \in \mathscr{CG}. [G]_{\simeq}$ satisfying for all $G, G' \in \mathscr{CG}$ that $L(G) = L(G') \Leftrightarrow G \simeq G'$. We say that $L(G)$ is a *canonical labelling* of G, and that G is *canonical* if $L(G) = G$.

3 Edge Enumeration Algorithms

This section introduces two algorithms based on enumeration of edges and the comparison of these enumerations.

3.1 Edge Enumerators

The rigidity property of coloured graphs, together with the linear ordering of colours, means that any given initial node uniquely identifies an enumeration of all the edges in a graph. Such an enumeration can be obtained by traversing the graph from the initial node while always following edges according to the given linear ordering on colours. There are many possible enumeration procedures, so we generalise the notion of an *edge enumerator* as follows.

Definition 4. An *edge enumerator* is a function of the form $\eta : \prod G \in \mathscr{CG}. V_G \to \mathscr{T}(E_G) \times (V_G \to_{\mathsf{bij}} V_G)$ satisfying for all G, G' and $v \in V_G, v' \in V_{G'}$ with $v \simeq v'$, $\eta(G,v) = (\lhd, \alpha)$ and $\eta(G',v') = (\lhd', \alpha')$ that $\alpha(G, \lhd) = \alpha'(G', \lhd')$.

An edge enumerator must hence produce a linear ordering of edges and an alpha-conversion, with the key property that the alpha-converted graph is invariant under isomorphism. A more direct definition is possible, but we require the alpha-conversion to be given explicitly by the enumerator for use in subsequent algorithms. We get the following directly from the definition of edge enumerators.

Lemma 5. *Let η be an edge enumerator and let $\eta(G,v) = (\lhd, \alpha)$ and $\eta(G',v') = (\lhd', \alpha')$ for some G, G' and v, v'. If $\alpha(G) = \alpha(G')$ then $(G,v) \simeq (G',v')$.*

Algorithm 1 implements an edge enumerator. It essentially carries out a breadth first search (BFS) on the underlying undirected graph: edges are explored in order of colour, but with out-edges arbitrarily explored before in-edges (lines 7-9). The inner loop checks whether an edge has been encountered previously (line 12) in order to avoid an edge being enumerated twice. The generated alpha-conversion renames vertices by their order of discovery by the algorithm (line 17).

Proposition 6. *Algorithm 1 computes an edge enumerator.*

The intuition of the proof is that the control flow of the algorithm does not depend on vertex identity, and hence the output should indeed be invariant under automorphism. The full proof is by induction in the number of inner loop iterations. For the complexity analysis observe that the algorithm is essentially a BFS which hence runs in $O(|V_G| + |E_G|)$ time, which by assumption is $O(|V_G|)$ in our case. The extensions do not affect this complexity bound, assuming that the checks for containment of elements in Visited and Enum are $O(1)$; this is possible using hash map implementations.

3.2 A Pair-Wise Canonical Labelling Algorithm

Alpha-converted edge enumerations can be ordered lexicographically based on edge identity firstly, and on edge colours secondly. This ordering is defined formally as follows.

Definition 7. Define a linear order $\sqsubset$ on coloured edges as $(e,c) \sqsubset (e',c')$ iff $e < e' \vee (e = e' \wedge c \prec c')$. We extend the order to pairs (G,e) s.t. $(G,e) \sqsubset (G',e')$ iff $(e, \phi_G(e)) \sqsubset (e', \phi_{G'}(e'))$. Finally, we assume the lexicographic extension of $\sqsubset$ to pairs $(G, \lhd), (G', \lhd')$ of edge-ordered graphs.

Algorithm 1: An edge enumeration algorithm. We assume given an operator $\text{Sort}_{\prec}(X)$ which sorts the set X according to a linear ordering $\prec$. The operators :: and @ are list cons and append, respectively.

```
   input  : a graph G and a start vertex v ∈ V_G
   output: an enumeration of the edges in G from v
1  Enum ← [] /* a list of enumerated edges                                   */
2  α ← {(v → 1)} /* a vertex renaming identifying order of encounter         */
3  Q ← Queue(v) /* a queue initialised with v                                */
4  Visited ← {v} /* a set of visited vertices                                */
5  while Q is not empty do
6      v ← Dequeue(Q)
7      outEdges ← Sort_≺(Adj(v))
8      inEdges ← Sort_≺(Adj^{-1}(v))
9      edges ← outEdges@inEdges
10     for i = 1 to |edges| do
11         (v1, v2) ← edges.i
12         if (v1, v2) does not occur in Enum then
13             Enum ← (v1, v2) :: Enum
14             v_new ← v1 if v = v2 and v2 if v = v1
15             if v_new ∉ Visited then
16                 Visited ← Visited ∪ {v_new}
17                 α ← α ∪ {(v_new ↦ |Visited|)}
18                 Enqueue(Q, v_new)
19 return (Enum, α)
```

The ordering is used for canonical labelling by Algorithm 2, which essentially finds the edge enumeration from each vertex in the input graph, picks the smallest after alpha-conversion, and applies the associated alpha-conversion to the graph. Whenever two alpha-converted enumerations are identical (line 11), the source vertices are isomorphic by definition of edge enumerators. The isomorphism is then computed (line 12) and the set of pending vertices is filtered so that it contains at most one element from each pair of isomorphic vertices (lines 13-20). If both elements of a pair of isomorphic vertices are in the set of pending vertices, one element is removed from the pending set (lines 16-17); the least element under $<$ is chosen arbitrarily, which ensures that the other element does not get removed at a later iteration. If just one element of the pair is in the pending set (lines 18-19), the other element must have been visited earlier, and so the present pending element can be safely removed.

Proposition 8. *Algorithm 2 is a canonical labeller.*

The worst-case time complexity of Algorithm 2 is $O(|V_G|^2)$, namely when no vertices are isomorphic. If all nodes in the input graph are isomorphic, the number of pending vertices is halved at every iteration of the outer loop and hence the complexity is $O(|V_G| \cdot \log(|V_G|))$. More generally, this bound also holds in the average case if the largest automorphism equivalence class has size $O(|V_G|)$. There are several ways of improving the algorithm. One way is to further exploit automorphisms. When new automorphisms are found, these may generate additional automorphisms through composition with existing ones. In addition

Algorithm 2: A pair-wise canonical labelling algorithm.

input : a graph G and an edge enumerator η
output: a canonical labelling of G

```
1  V_pending ← V_G /* vertices yet to be enumerated */
2  ◁_min ← null /* the least ordering */
3  α_min ← null /* the least alpha-conversion */
4  while V_pending ≠ ∅ do
5      v ← any v ∈ V_pending
6      V_pending ← V_pending \ v
7      (◁, α) ← η(G, v)
8      if (◁_min = α_min = null) or (α(G, ◁) ⊏ α_min(G, ◁_min)) then
9          ◁_min ← ◁
10         α_min ← α
11     else if α(G, ◁) = α_min(G, ◁_min) then
12         a ← α_min^{-1} ∘ α /* find the automorphism */
13         V'_pending ← V_pending /* keep a copy of pending vertices */
14         for v' ∈ Dom(α) do
15             /* discard isomorphic vertices from pending */
16             if {v, α(v)} ⊆ V_pending then
17                 V'_pending ← V_pending \ Min_<{v, α(v)}
18             else
19                 V'_pending ← V_pending \ {v, α(v)}
20         V_pending ← V'_pending
21 return α_min(G)
```

to removing elements of the pending set, automorphisms could also be exploited by only considering the automorphism quotient graph in subsequent iterations. Another way of improving the algorithm is to compute edge enumerations lazily, up until the point where they can be distinguished from the current minimum. Neither of the above improvements, however, change the worst case complexity characteristics of the algorithm.

3.3 A Parallel Canonical Labelling Algorithm

The idea of lazily enumerating edges can be taken a step further with a second algorithm which enumerates edges from all nodes in parallel: at each iteration, a single edge is emitted by each enumeration. Following standard notions of lazy functions, we formally define the notion of a *lazy edge enumerator* below, where the singleton set $\{*\}$ corresponds to the *unit type.*

Definition 9. For each coloured graph G and $i \in \{0 \ldots |E_G|-1\}$, define the set of functions $T_i(G) \overset{\Delta}{\simeq} \{*\} \to E_G \times V_G \times V_G \times T_{i+1}$ with $T_{|E_G|} \overset{\Delta}{\simeq} \mathscr{CG}$. A *lazy edge enumerator* implementing a given enumerator η is then a function $\eta_{\mathrm{L}} : \prod G \in \mathscr{CG}.V_G \to T_0(G)$ satisfying for any G and $v \in V_G$ with $\eta(G,v) = (\lhd, \alpha)$ that $(E_G, \lhd).i = e_i$, $\alpha(e_i) = (v_i, v'_i)$ and $\eta_{\mathrm{L}|E_G|} = \alpha(G)$ where $(e_i, v_i, v'_i, \eta_{\mathrm{L}i}) \overset{\Delta}{\simeq} \eta_{\mathrm{L}i-1}(*)$ for $i \in \{1 \ldots |E_G|\}$

and $\eta_{L0} \overset{\Delta}{\simeq} \eta_L(G,v)$.

Hence a lazy edge enumerator produces, given a graph and a start vertex, a function which can be applied to yield an edge, an alpha-conversion of the edge (i.e. two vertices) and a continuation which in turn can be applied in a similar fashion; the final function thus applied yields an alpha-converted graph for notational convenience below. A lazy version of the BFS edge enumerator in Algorithm 1 can be implemented in a straightforward manner in a functional language. This does not affect the complexity characteristics of the algorithm, i.e. it still runs in $O(|V_G|)$ time.

Algorithm 3 computes canonical labellings using lazy edge enumerators. It first initialises a set of enumerators from each vertex in the input graph (line 1). It then enters a loop in which enumerators are gradually filtered out, terminating when the remaining enumerators complete with an alpha-converted graph. At termination, all remaining enumerators will have started from isomorphic vertices, and hence they all evaluate in their last step to the same alpha-converted graph. At each iteration, each enumerator takes a step, yielding the next version of itself and an alpha-converted coloured edge; the result is stored as a mapping from the former to the latter (line 3). A multiset of alpha-converted, coloured edges is then constructed for the purpose of counting the number of copies of each alpha-converted coloured edge (line 4). The alpha-converted coloured edges with the smallest multiplicity are selected, and of these the least under $\sqsubset$ is selected (lines 5-6). Only the enumerators which yielded this selected edge are retained in the new set of pending edge enumerators (line 7). Note that further discrimination according to connectivity with other enumerators would be possible: for example, the vertices which are sources of enumerators eliminated in step n could be distinguished from those which are sources of enumerators eliminated in step $m \neq n$. We have omitted this for simplicity.

Algorithm 3: A parallel canonical labelling algorithm.

input : a graph G and a lazy edge enumerator η_L
output: a canonical labelling of G

1 Pending $\leftarrow \{\eta_L(G,v) \mid v \in V_G\}$
2 **while** *Pending* $\not\subset \mathscr{CG}$ **do**
3 StepMap $\leftarrow \{\eta_L{}' \mapsto ((v,v'),\phi_G(e)) \mid (e,v,v',\eta_L{}') = \eta_L(*) \wedge \eta_L \in \mathit{Pending}\}$
4 Steps $\leftarrow \{\!\!\{ \text{StepMap}(\eta_L{}') \mid \eta_L{}' \in \text{Dom(StepMap)} \}\!\!\}$
5 SmallestMult $\leftarrow \min_{<}\{\text{Steps}(e,c) \mid (e,c) \in \text{Steps}\}$
6 LeastEdge $\leftarrow \min_{\sqsubset}\{(e,c) \in \text{Steps} \mid \text{Steps}(e,c) = \text{SmallestMult}\}$
7 Pending $\leftarrow \{\eta_L{}' \in \text{Dom(StepMap)} \mid \text{StepMap}(\eta_L{}') = \text{LeastEdge}\}$
8 **return** the one member of Pending

Proposition 10. *Algorithm 3 is a canonical labeller.*

The initialisation in line 1 runs in $O(|V_G|)$ time. The loop always requires $|E_G| = O(|V_G|)$ iterations. In the worst case where all vertices are isomorphic, each line within the loop requires $O(|V_G|)$ time, so the worst case complexity is $O(|V_G|^2)$. Hence in the worst case there is no asymptotic improvement over Algorithm 2. In practice, however, Algorithm 3 is likely to perform significantly better.

The key question is how Algorithm 3 behaves in cases where there are *few* automorphisms, i.e. when the number of automorphisms is sub-linear in the number of vertices. One can hypothesise that asymmetry is then discovered sufficiently early to yield an $O(|V_G| \cdot \log(|V_G|))$ time complexity. If so, an overall $O(|V_G| \cdot \log(|V_G|))$ algorithm is obtained by running the pairwise and the parallel algorithms simultaneously, terminating when the first of the two algorithms terminates. However, this question

remains open. Therefore also the question of whether a sub-quadratic time complexity bound exists in the general case remains open.

4 A Partition Refinement Algorithm

The vertex set of a coloured graph can be partitioned based on "local views": vertices with the same colours of incident edges are considered equivalent and are hence included in the same equivalence class of the partition. If two vertices are in different classes, they are clearly not isomorphic. The partition can then be refined iteratively: if some vertices in a class P have c-coloured edges to vertices in a class Q while others do not, P is split into two subclasses accordingly. This *partition refinement* process can be repeated until no classes have any remaining such diverging edges. An efficient algorithm for partition refinement was given in 1971 by Hopcroft [5]. The original work was in the context of deterministic finite automata (DFA), where partition refinement of DFA states gives rise to a notion of language equivalence, thus facilitating minimisation of the DFA. Hopcroft's algorithm runs in $O(n \cdot \log(n))$ where n is the number of states of the input DFA.

We show in the next subsection how our coloured graphs can be viewed as DFA, thus enabling the application of Hopcroft's partition refinement algorithm. The literature does provide generalisations of Hopcroft's algorithm to other structures, including general labelled graphs [2] where a vertex can have multiple incident edges with the same colour. However, we adopt Hopcroft's DFA algorithm as this remains the simplest for our purposes. Furthermore, the algorithm has been thoroughly described and analysed in [6], which we use as the basis for our presentation. In the second subsection we extend Hopcroft's algorithm for use in canonical labelling. We finally discuss how partition refinement relates to the edge enumeration algorithms presented in the previous section.

4.1 A Deterministic Finite Automata View

We refer to standard text books such as [11] for details on automata, but recall briefly that a DFA is a tuple $(A, \Sigma, \delta, a_0, A')$ where A is a set of *states*, Σ is the *input alphabet*, $\delta : A \times \Sigma \to A$ is a total *transition function*, $a_0 \in A$ is an *initial state* and $A' \subseteq A$ is a set of *final states*. A coloured graph G is almost a DFA: V_G can be taken both as the set of states and as the set of final states, the edge colours in G can be taken as the input alphabet, and E_G determines a transition function albeit a *partial* one, meaning that the DFA is *incomplete*. There is no dedicated initial state, but initial states play no role in partition refinement [6]. A *total* transition function is traditionally obtained by adding an additional non-accepting sink state with incoming transitions from all states for which such transitions are not defined in the original DFA. However, it is convenient for our purposes to add a distinct such sink state for each state in the original DFA. This allows us one further convenience, namely to ensure that each transition on a colour c has a reverse transition on a distinct "reverse" colour, $c^- \notin \Sigma$. These ideas are formalised as follows.

Definition 11. Let $G \in \mathscr{CG}$. Define the *states* $A_G \stackrel{\Delta}{\simeq} V_G \cup V_G^{\mathbf{u}}$ where $V_G^{\mathbf{u}} \stackrel{\Delta}{\simeq} \{\mathbf{Undef}_v \mid v \in V_G\}$. Define the *final states* $F_G = V_G$. Define the *input alphabet* $\Sigma_G \stackrel{\Delta}{\simeq} \mathrm{Im}(\phi_G) \cup \{c^- \mid c \in \mathrm{Im}(\phi_G)\}$. Define the *reversible colour transition* function $\delta_G : A_G \times \Sigma_G \to A_G$ and the *colour path* function $\hat{\delta}_G : A_G \times \Sigma_G^* \to A_G$ as follows:

$$\delta_G(a,x) \overset{\Delta}{\simeq} \begin{cases} v' & \text{if } (a,v') \in E_G \wedge \phi_G(a,v') = x \\ v & \text{if } (v,a) \in E_G \wedge \phi_G(v,a) = c \wedge x = c^- \\ \mathbf{Undef}_a & \text{otherwise and } a \in V_G \\ a & \text{otherwise} \end{cases}$$

$$\hat{\delta}_G(a,w) \overset{\Delta}{\simeq} \begin{cases} a & \text{if } w = \varepsilon \\ \hat{\delta}_G(a',w') & \text{if } w = xw' \wedge a' = \delta_G(a,x) \end{cases}$$

Hence the tuple $(A_G, \Sigma_G, \delta_G, F_G)$ is a DFA with no initial state. It follows from rigidity of coloured graphs and from the choice of **Undef** states that δ_G is injective. The colour path function $\hat{\delta}_G$ gives the end state of a path specified by a colour word w from an initial state a. Hence $\hat{\delta}_G(a,w) \in V_G = F_G$ exactly when the word w is accepted from the state a, or equivalently when the colour path w exists in the graph G. With this in mind, the following notion of *vertex bisimulation* corresponds exactly to the notion of DFA state equivalence given in [6] and proven to be the relation computed by Hopcroft's algorithm (Corollary 15 in [6]).

Definition 12. Let $G \in \mathscr{CG}$. Define the *vertex bisimulation* relation $\rho_G \subseteq A_G \times A_G$ as $a\,\rho_G\,a'$ iff $\forall w \in \Sigma_G^*. (\hat{\delta}_G(a,w) \in F_G \Leftrightarrow \hat{\delta}_G(a',w) \in F_G)$.

4.2 Adapting Hopcroft's Algorithm

The strategy of using partition refinement for canonical labelling is to limit the number of vertices under consideration to those of a single equivalence class in the partition V_G/ρ_G. If the selected class has size 1, the unique vertex in this class can be chosen as the source of canonical labelling via edge enumeration. If the selected class is larger, one of the two edge enumeration algorithms from Section 3 can be employed, but starting from only the vertices of this class.

The challenge then is how, exactly, to select an equivalence class from the unordered partition A_G/ρ_G resulting from Hopcroft's algorithm. The selection must clearly be invariant under automorphism in order to be useful for canonical labelling. One approach could be to employ edge enumeration algorithms on the quotient graph G/ρ_G, but in the worst case this yields quadratic time and hence defeats the purpose of partition refinement. Instead we give an extension of Hopcroft's algorithm which explicitly selects an appropriate class. The following definition introduces the relevant notation, adapted from [6], required by the algorithm.

Definition 13. Let $G \in \mathscr{CG}$ and let θ be an equivalence relation on V_G. Let $P, Q \in A_G/\theta$ and let $x \in \Sigma_G$. Then define $P_{Q,x} \overset{\Delta}{\simeq} P \cap \{\delta_G^{-1}(a,x) \mid a \in Q\}$ and $P^{Q,x} \overset{\Delta}{\simeq} P \setminus P_{Q,x}$. Also define the *refiners* $\mathrm{ref}(P,\theta) \overset{\Delta}{\simeq} \{(Q,x) \in (A_G/\theta) \times \Sigma_G \mid P_{Q,x} \neq \emptyset \wedge |P_{Q,x}| < |P|\}$ and the *objects* $\mathrm{obj}(Q,x,\theta) \overset{\Delta}{\simeq} \{P \in A_G/\theta \mid (Q,x) \in \mathrm{ref}(P,\theta)\}$.

Informally, the set $\mathrm{obj}(Q,x,\theta)$ specifies the θ-classes which can be refined based on x-labelled transitions from states in the class Q. The original version of Hopcroft's algorithm from [6] is listed in Algorithm 5 in Appendix A for the sake of completeness. The algorithm maintains two sets: one is the partition at a given stage (line 1), and the second is the set L of pending refiners (line 2), i.e. pairs of classes and transitions which will be used for refining at a later stage. The algorithm then loops until there are no more refiners in L. At each iteration a refiner (Q,x) is selected arbitrarily from L and used to refine all the classes in $\mathrm{obj}(Q,x,\theta)$. The key insight of Hopcroft is to selectively add new refiners to

Algorithm 4: An extensions of Hopcroft's partition refinement algorithm. The AddBetter routine is defined as for Algorithm 5 in Appendix A.

input : A graph G
output: The relation ρ_G and a selected $P \in (V_G^{\mathbf{u}}/\rho_G)$

```
1  M ← V_G
2  A_G/θ ← {V_G, V_G^u}
3  L ← []
4  foreach x ∈ Σ_G in increasing order of ≺ do
5      add (V_G^u, x) to beginning of L
6  while L ≠ ∅ do
7      remove the first pair (Q, x) from L
8      foreach P ∈ obj(Q, x, θ) do
9          replace P with P_{Q,x} and P^{Q,x} in A_G/θ
10         if P = M then
11             M ← P_{Q,x}
12     foreach P just refined do
13         foreach x′ ∈ Σ_G in increasing order of ≺ do
14             if (P, x′) ∈ L then
15                 replace (P, x′) with (P_{Q,x}, x′) first and (P^{Q,x}, x′) second in L
16             else
17                 AddBetter((P_{Q,x}, x′), (P^{Q,x}, x′), x′, L)
18 return (θ, M)
```

the set L, namely only the *better* half of any classes which are split and not already included as a refiner (line 14). We choose the smallest half to be the better one, although other choices are possible.

The problem with the original algorithm for canonical labelling purposes is essentially that A_G/θ and L are maintained as sets, hence introducing non-determinism at several points. One solution could be to maintain both A_G/θ and L as lists instead, and adapt the algorithm to process their content in a consistent order. But this would complicate the analysis and implementation details meticulously described in [6]. Our adapted version, Algorithm 4, instead maintains only L as a list, which is not at odds with the implementation in [6]. There, L is a list of integer pairs with the first element identifying a partition and the second element identifying an edge colour. We then change the control flow governing L to become deterministic by exploiting the linear ordering of edge colours (lines 4 and 13), and otherwise consistently operate on elements of the list (lines 7, 15 and 17). The linear ordering on edge colours is assumed extended to states, e.g. by ordering the "reverse" colours after the standard colours. Finally, we explicitly maintain a "least" class (line 1): whenever the least class is refined, we consistently choose one sub-class to become the new least class (lines 10 and 11). Note how the rigidity property is key to this extension of Hopcroft's algorithm.

The extensions do not affect the correctness of the algorithm because the particular non-deterministic choices made in Hopcroft's algorithm do not affect its final output (Corollary 10 in [6]). The extensions do not affect the complexity analysis either, so the algorithm still runs in $O(|A_G| \cdot \log|A_G|)$ time, which is the same as $O(|V_G| \cdot \log|V_G|)$. In particular, the ordering of colours can be computed up front in

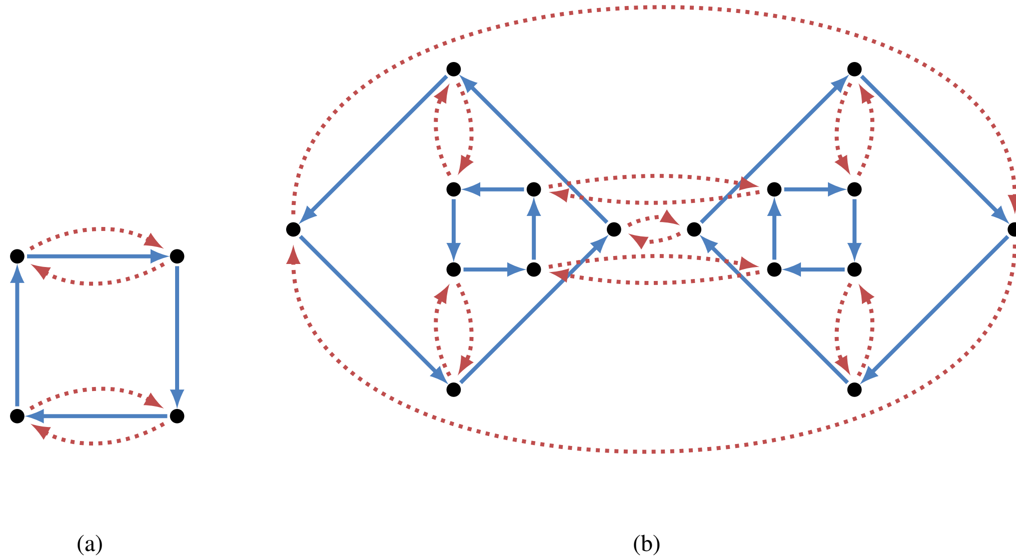


Figure 4.1: (a) A graph with one bisimulation class and two non-trivial automorphism classes (obtained from reflection on the two diagonals). (b) A graph with one bisimulation class and just one non-trivial automorphism class (obtained from vertical reflection); all vertices participate in single-coloured cycles of the same type.

$O(|\Sigma_G| \cdot \log|\Sigma_G|)$ by standard sorting algorithms; the list operations on L can be implemented in $O(1)$; and the comparison in line 10 can likewise be implemented in $O(1)$ given that the classes in L can be represented by integers.

The key property needed for canonical labelling is that the "least" class M returned by the algorithm is invariant under automorphism. This is indeed the case; as for the edge enumeration algorithms, the intuition is that the control flow does not depend on vertex identity.

Proposition 14. *Let $G, G' \in \mathscr{CG}$ and let $\sigma \in I(G, G')$. Let (ρ, M) and (ρ', M') be the results of running Algorithm 4 on G and G', respectively. Then $\sigma(M) = M'$.*

It follows that the composite algorithm which first runs partition refinement via Algorithm 4, and then runs one of the edge enumeration algorithms 2 or 3 on the returned least class, is in fact a canonical labeller. In the cases where the selected ρ_G-class has size 1, the composite algorithm runs in worst case time $O(|V_G| \cdot \log|V_G|)$. More generally, this bound also holds if there are "few" bisimulations, i.e. if V_G/ρ_G has size $O(|V_G|)$. This is due to all bisimulation equivalence classes having the same size as the following proposition shows, and hence the particular choice of equivalence class does not affect time complexity.

Proposition 15. *For any coloured graph G and any $P, Q \in (V_G/\rho_G)$, $|P| = |Q|$.*

4.3 Bisimulation Versus Isomorphism

One could hope that the bisimulation and isomorphism relations for coloured graphs were identical, for then the choice of vertex from a selected ρ_G-class would not matter, giving a worst-case $O(|V_G| \cdot \log|V_G|)$

canonical labelling algorithm. But, unsurprisingly, this is not the case. Figure 4.1a shows a graph which has a single bisimulation equivalence class but two automorphism equivalence classes. Hence graphs of this kind, where all vertices have the same "local view", capture the difficult instances of the graph isomorphism and canonical labelling problem for site graphs. The difficulty is essentially that isomorphisms are bijective *functions* and hence must account for vertex identity at some level. This is not the case for bisimulation equivalence.

One possible solution could be to annotate vertices with additional, "semi-local" information such as edge enumeration up to a constant length, as also suggested in [9], and then take this into account during partition refinement. However, this is unlikely to improve the asymptotic running time. Another approach could be to analyse the cycles of the input graph after partition refinement, and then take this analysis into account during a second partition refinement run on the quotient graph from the first run. The observation here is that all same-coloured paths in the quotient graph are cycles, and these cycles can be detected in linear time. However, vertices with the same *local* view and single-coloured cycles are not necessarily isomorphic, as demonstrated by Figure 4.1b. It seems that simple cycles of arbitrary colour combinations must be taken into account, e.g. to obtain cycle bases of the graph, but there is no clear means of doing so in linear time. Further attempts in this direction have been unsuccessful.

However, we observe that there are classes of coloured graphs for which bisimulation and isomorphism do coincide. This holds for example for coloured graphs with out and in-degree at most 1, and more generally for acyclic coloured graphs (trees). For these classes, the partition refinement approach does yield an $O(|V_G| \cdot \log|V_G|)$ worst case canonical labelling algorithm. Furthermore, linear time may be possible for these classes using e.g. the partition refinement in [4], assuming an extension similar to that of Algorithm 4 can be realised.

Finally, we note that there is an open question of how the parallel edge enumeration algorithm behaves on graphs with "few" bisimulations. We conjecture that the algorithm may in these cases run in $O(|V_G| \cdot \log|V_G|)$ time. If so, partition refinement would be unnecessary. However, for the time being, the partition refinement approach does serve the purpose of providing the upper bound of $O(|V_G| \cdot \log|V_G|)$ for graphs with few bisimulations.

5 Conclusions

We have considered the problem of canonical labelling of site graphs, which we have shown reduces to canonical labelling of standard digraphs with edge colourings. We have presented an algorithm based on edge enumeration which runs in $O(|V_G|^2)$ worst-case time, and in $O(|V_G| \cdot \log|V_G|)$ average-case time for graphs with many automorphisms. A variant of this algorithm, based on parallel enumeration of edges, is likely to perform well in praxis, but in general does not improve on the worst case complexity bounds. However, the question of how the parallel algorithm performs in cases with few automorphisms remains open. If it is found to run in $O(|V_G| \cdot \log|V_G|)$ time, this yields an overall $O(|V_G| \cdot \log|V_G|)$ average-case algorithm and hence resolves the open question of whether such an algorithm exists. We have also introduced an algorithm based on partition refinement which can be used as a preprocessing step, yielding $O(|V_G| \cdot \log|V_G|)$ worst case time for graphs with few bisimulation equivalences.

A different line of attack is taken in [9] which introduces a notion of a graph's "gravity centre", namely a subgraph on which it is sufficient to detect automorphisms. Hence this approach is efficient when the gravity centre is small, and could also be a useful pre-processing step for canonical labelling. However, many of the difficult graphs that we have considered, including the one in Figure 4.1b, do not appear to have small gravity centres.

Other related work includes that on the *general* graph isomorphism problem which has been extensively studied in the literature. Hence highly optimised algorithms, such as the one by McKay [8], exist, although none run in sub-exponential time on "difficult" graphs. Many other special cases have been studied, including notably graphs with bounded degree for which worst-case polynomial time algorithms do exist [1]. Site graphs can indeed be encoded into standard graphs with bounded degree. However, polynomial time algorithms on such encodings do not appear to be sub-quadratic or even quadratic. Perhaps surprisingly, isomorphism of site graphs, or equivalently digraphs with edge colourings, has to the best of our knowledge not been treated in the general literature. Although the difficult cases are perhaps rare and of limited practical relevance, the question is theoretically interesting. It certainly appears to be conducive to infection with the graph isomorphism disease [10].

Acknowledgements

We thank the anonymous reviewers for useful comments. The work of the second author was supported by an EPSRC Postdoctoral Fellowship (EP/H027955/1).

References

[1] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM. doi: 10.1145/800061.808746.

[2] A. Cardon and Maxime Crochemore. Partitioning a graph in O(|a| log2 |v|). *Theor. Comput. Sci.*, 19:85–98, 1982. doi: 10.1016/0304-3975(82)90016-0.

[3] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *APLAS*, volume 4807 of *LNCS*, pages 139–157. Springer, 2007. doi: 10.1007/978-3-540-76637-7_10.

[4] Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci*, 311:221–256, 2004. doi: 10.1016/S0304-3975(03)00361-X.

[5] John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford, CA, USA, 1971.

[6] Timo Knuutila. Re-describing an algorithm by hopcroft. *Theor. Comput. Sci.*, 250(1-2):333–363, January 2001. doi: 10.1016/S0304-3975(99)00150-4.

[7] Matthew R. Lakin, Loïc Paulevé, and Andrew Phillips. Stochastic simulation of multiple process calculi for biology. *Theor. Comput. Sci.*, 431:181–206, 2012. doi: 10.1016/j.tcs.2011.12.057.

[8] B. D. McKay. Practical graph isomorphism. *Congr. Numer.*, 30:45–87, 1981.

[9] Tatjana Petrov, Jerome Feret, and Heinz Koeppl. Reconstructing species-based dynamics from reduced stochastic rule-based models. In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 1–15. Winter Simulation Conference, 2012. doi: 10.1109/WSC.2012.6465241.

[10] R. C. Read and D. G. Cornell. The graph isomorphism disease. *Journal of graph theory*, 1(4): 339–363, 1977. doi: 10.1002/jgt.3190010410.

[11] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X. doi: 10.1145/230514.571645.

A Site Graphs and Hopcroft's Original Algorithm

In the literature site graphs are typically defined as expressions in a language, which is natural when considering simulation and analysis of rule-based models. We give a more direct definition suitable for our purposes. Site graphs in the literature often include *internal states* of sites, representing e.g. post-translational modification. We omit internal states but they are straightforward to encode.

Definition 16. Let $(\Sigma_{\mathrm{p}}, \preceq_{\mathrm{p}})$ and $(\Sigma_{\mathrm{s}}, \preceq_{\mathrm{s}})$ be given, disjoint linearly ordered sets of protein and site names, respectively. Then $\mathscr{SG}$ is the set of all *site graphs* $S = (V, E, \phi_{\mathrm{p}})$ satisfying:

- $V = \{1, \dots, k\}$ is a set of vertices.
- $E \subseteq \binom{V \times \Sigma_{\mathrm{s}}}{2}$ is a set of site-labelled, undirected edges satisfying that $\forall e, e' \in E. e \neq e' \rightarrow e \cap e' = \emptyset$.
- $\phi_{\mathrm{p}} : V \rightarrow \Sigma_{\mathrm{p}}$ is a vertex (protein) naming.

Note the key condition on edges that a given site can occur at most once within a vertex.

Definition 17. Let $S, S' \in \mathscr{SG}$. A *site graph isomorphism* is a bijective function $\sigma : V_S \rightarrow V_{S'}$ satisfying:

1. $\forall v_1, v_2 \in V_S. [\{(v_1, s_1), (v_2, s_2)\} \in E_S \Leftrightarrow \{(\sigma(v_1), s_1), (\sigma(v_2), s_2)\} \in E_{S'}]$.
2. $\forall v \in V_S. \phi_{\mathrm{p}S}(v) = \phi_{\mathrm{p}S'}(\sigma(v))$.

The first condition states that edges and edge site names are preserved by the isomorphism, and the second condition states that protein names are preserved. We next show how to encode site graphs into coloured graphs. Let $(\Sigma_{\mathrm{p}}, \preceq_{\mathrm{p}})$ and $(\Sigma_{\mathrm{s}}, \preceq_{\mathrm{s}})$ be given. The aim is to construct a linearly ordered edge colour set $(\Sigma, \prec)$ and a total function $\rho : \mathscr{SG}(\Sigma_{\mathrm{p}}, \Sigma_{\mathrm{s}}) \rightarrow \mathscr{CG}(\Sigma)$ which preserves isomorphism. We define the colour set as $\Sigma \stackrel{\Delta}{\simeq} \mathscr{P}(\Sigma_{\mathrm{s}} \times \Sigma_{\mathrm{s}}) \cup \Sigma_{\mathrm{p}}$, and the linear order on Σ as $c \prec c'$ iff one of the following conditions hold:

- $c \in \Sigma_{\mathrm{p}} \wedge c' \notin \Sigma_{\mathrm{p}}$ or
- $c, c' \in \Sigma_{\mathrm{p}} \wedge c \preceq_{\mathrm{p}} c'$ or
- $c, c' \in \mathscr{P}(\Sigma_{\mathrm{s}} \times \Sigma_{\mathrm{s}}) \wedge \mathrm{Sort}_{\preceq_{\mathrm{s}}}(c) \preceq_{\mathrm{s}} \mathrm{Sort}_{\preceq_{\mathrm{s}}} c'$

In the latter case we assume the $\preceq_{\mathrm{s}}$ relation extended lexicographically to pairs and lists as usual. Let $S \stackrel{\Delta}{\simeq} (V, E, \phi_{\mathrm{p}})$ be a given site graph. We then define $\rho(S) \stackrel{\Delta}{\simeq} (V', E', \phi')$ where:

1. $V' \stackrel{\Delta}{\simeq} V$.
2. $E' \stackrel{\Delta}{\simeq} E'_1 \cup E'_2$ where:
   (a) $E'_1 \stackrel{\Delta}{\simeq} \{(v, v') \mid \exists s, s'. s \preceq_{\mathrm{s}} s' \wedge (s, s') = \min_{\preceq_{\mathrm{s}}} \{(s, s') \mid \{(v, s), (v', s')\} \in E\}$
   (b) $E'_2 \stackrel{\Delta}{\simeq} \{(v, v) \mid v \in V\}$
3. $\phi'(v, v') \stackrel{\Delta}{\simeq} C'_1 \cup C'_2$ where
   (a) $C'_1 \stackrel{\Delta}{\simeq} \{(s, s') \mid \{(v, s), (v', s')\} \in E\}$
   (b) $C'_2 \stackrel{\Delta}{\simeq} \begin{cases} \{\phi_{\mathrm{p}}(v)\} & \text{if } v = v' \\ \emptyset & \text{otherwise} \end{cases}$

The encoding does not affect vertices. Note that site graphs can have multiple unordered edges between nodes while coloured graphs have at most one, ordered edge. The direction of this one edge is determined in Step 2*a* from the site ordering of the least pair of sites, where the least pair of sites is determined from the extension of the site ordering to pairs. All vertices have self-loops (step 2*b*) which are used to encode vertex colour as edge colour in step 3*b*. Step 3a assigns a colour to an edge as the union colours of each edge between the edge in the site graph; the ordering of colours follows that assigned to the edge.

Proposition 18. *The coding function ρ satisfies the following for all $S, S' \in \mathrm{Dom}(\rho)$:*

1. ***Injective:*** $\rho(S) = \rho(S') \Rightarrow S = S'$.
2. ***Respects isomorphism:*** $S \simeq S' \Leftrightarrow \rho(S) \simeq \rho(S')$.
3. ***Linear size:*** $|\rho(S)| = O(|S|)$.
4. ***Linear time computable:*** ρ *is computable in* $O(|S|)$.

It follows immediately that the coding function together with a canonical labeller for coloured graphs can be used to define a canonical labeller for site graphs as follows.

Corollary 19. *Given a canonical labeller $L : \prod G \in \mathscr{CG}\,.\,[G]_{\simeq}$ on coloured graphs running in $O(f(|G|)) \geq O(|G|)$ time, the function $L^*(S) \overset{\Delta}{\simeq} \rho^{-1}(L(\rho(S)))$ is an $O(f(|G|))$ time canonical labeller on site graphs.*

Algorithm 5: A version of Hopcroft's original algorithm adapted from Algorithm 4 in [6].

input : a graph G
output: the relation ρ_G

1 $A_G/\theta \leftarrow \{V_G, V_G^{\mathbf{u}}\}$
2 $L \leftarrow \emptyset$
3 **foreach** $x \in \Sigma_G$ **do**
4 add $(V_G^{\mathbf{u}}, x)$ to L
5 **while** $L \neq \emptyset$ **do**
6 remove a pair (Q, x) from L
7 **foreach** $P \in obj(Q, x, \theta)$ **do**
8 replace P with $P_{Q,x}$ and $P^{Q,x}$ in A_G/θ
9 **foreach** P *just refined* **do**
10 **foreach** $x' \in \Sigma_G$ **do**
11 **if** $(P, x') \in L$ **then**
12 replace (P, x') with $(P_{Q,x}, x')$ and $(P^{Q,x}, x')$ in L
13 **else**
14 AddBetter($(P_{Q,x}, x')$, $(P^{Q,x}, x')$, x', L)
15 **return** θ
16 AddBetter(P,Q,x,L):
17 **if** $|P| < |Q|$ **then**
18 add (P, x) to beginning of L
19 **else**
20 add (Q, x) to beginning of L