Useful names for vertices: An introduction to dynamic implicit informative labelling schemes

David Morgan

Department of Computing Science
University of Alberta
TR05-04

January 2005

Useful names for vertices: An introduction to dynamic implicit informative labelling schemes

David Morgan

Department of Computing Science, University of Alberta, Edmonton AB, Canada T6G 2E8.

Abstract

As defined by Peleg (Peleg, LNCS vol. 1893, 2000), an *informative labelling scheme* labels the vertices of a graph so that information can be deduced from the vertex labels. Peleg's paper on informative labelling schemes generalized the concepts introduced by Muller (Muller, Ph.D. thesis, Georgia Tech, 1988) and Kannan et al. (Kannan et al., SIAM J Disc Math, 1992) in which the adjacency of two vertices could be determined using only their labels. In general, the vertex labellings used in informative labelling schemes cannot be tweaked to accommodate small changes in the graph. Although Kannan et al., suggested the dynamic version of their problem as a direction for future research, only one paper has explored this topic. The paper of Brodal and Fagerberg (Brodal and Fagerberg, LNCS vol. 1663, 1999) devises a dynamic scheme for graphs of bounded arboricity, however, it lacks a formal statement of the dynamic problem.

Motivated by the necessity of such a formal statement, we define the notion of a *dynamic implicit informative labelling scheme*; discuss measures for judging the quality of such dynamic schemes; and show a connection between the existence of dynamic schemes and the recognition problem. To illustrate the concept of a dynamic implicit informative labelling scheme we develop a space-optimal dynamic implicit adjacency labelling scheme for *r-minoes*; as per the definition of Metelsky and Tyshkevich (Metelsky and Tyshkevich, SIAM J Disc Math, 2003), a graph is said to be an r-mino if none of its vertices belong to more than r maximal cliques.

1 Introduction

Consider a finite simple undirected graph $G = (V_G, E_G)$ with n vertices and m edges. Typically, we represent G using an adjacency matrix or a series of adjacency lists, labelling the vertices from 1 to n. These rudimentary labels serve only to distinguish between the vertices and do not tell us anything about the structure of G. In particular, the adjacency of any pair of vertices

must be determined from the adjacency matrix or the adjacency lists, both of which are usually maintained as global resources.

What if we could determine the adjacency of two vertices of G in a more local manner, that is, by using only the labels given to them? One way to do this is by labelling each vertex with a unique prelabel from $\{1, \ldots, n\}$, along with its corresponding row of the adjacency matrix whose indices are based on these prelabels. Given this labelling scheme, we can determine the adjacency of two vertices having prelabels v_1 and v_2 using only their labels by looking up the bit corresponding to v_2 in the row of the adjacency matrix found in the label of v_1, or vice versa. In this labelling scheme each vertex has a label of size $\Theta(n)$, the sum of the sizes of all the vertex labels is $\Theta(n^2)$, and adjacency queries can be handled in $\Theta(1)$ time. Another approach is to label each vertex with a unique prelabel from $\{1, \ldots, n\}$, along with a list of the prelabels of the vertices to which it is adjacent. Given this labelling scheme, we can determine the adjacency of two vertices having prelabels v_1 and v_2 using only their labels by determining if v_2 is in the adjacency list found in the label of v_1, or vice versa. In this labelling scheme a vertex v has a label of size $\Theta(\deg(v) \log n) \subseteq \mathrm{O}(n \log n)$, the sum of the sizes of all the vertex labels is $\Theta((m+n) \log n)$, and adjacency queries can be handled in $\mathrm{O}(\log n)$ time providing the adjacency lists are sorted. Upon extension to families of finite graphs these labelling schemes based upon adjacency matrices and adjacency lists are examples of informative (adjacency) labelling schemes as defined by Peleg [18].

Definition 1.1 (Peleg) *Consider a function $f(S, G)$ defined on sets of vertices S of fixed but arbitrary finite graphs G. An implicit f-labelling scheme of a family $\mathcal{G}$ of finite graphs is a pair (M, D) defined as follows.*

- *M is a vertex labelling algorithm whose input is a graph G in $\mathcal{G}$. Note that M need not be deterministic; accordingly, let $\mathcal{M}_G$ be the set of all vertex labellings of V_G which can be assigned by M.*

- *D is a polynomial time deterministic evaluation algorithm whose input is a set of vertex labels. Given any labelling L_G of V_G, let $L_{S,G}$ denote the subset of these labels corresponding to a subset S of V_G. For any graph G in $\mathcal{G}$ we define L_G to be (D, f)-correct if $D(L_{S,G}) = f(S, G)$ for every subset S of V_G on which f is defined. Given this definition, we require that M_G be (D, f)-correct for all G in $\mathcal{G}$ and for all M_G in $\mathcal{M}_G$. Note that D is a function of the labels only.*

Allowing sufficiently large labels we can create informative labelling schemes for any such function f, however, in doing so we may overlook

two desirable characteristics, namely, *space-optimality* and *balance*. If we define the size of a labelling of a graph to be the sum of the sizes of its vertex labels, then by a space-optimal f-labelling scheme of $\mathcal{G}$ we are referring to an f-labelling scheme which generates labellings of asymptotically smallest size over all f-labelling schemes of $\mathcal{G}$. By "asymptotically smallest size" we mean that the sizes of the graph labellings are considered asymptotically with respect to the number of vertices in the graph. If the size of a vertex labelling of a graph on n vertices is $\Theta(S)$, then by a balanced f-labelling scheme of $\mathcal{G}$ we are referring to an f-labelling scheme which generates vertex labels of size $O(\frac{S}{n})$, thus distributing the information about f across the graph. To date, balanced space-optimal informative labelling schemes have been developed for a variety of functions over certain graph classes, such as adjacency over interval graphs [11], distance over rings [17], and center of three vertices over trees [18].

The seminal works of both Muller [15] and Kannan et al. [11] presented a narrower version of adjacency labelling schemes in the form of what Spinrad [21] defines as the the implicit representation problem. A family $\mathcal{G}$ of finite graphs is said to have an implicit representation if there is an adjacency labelling scheme for $\mathcal{G}$ such that the members on n vertices have vertex labels of size $O(\log n)$. To date, implicit representations have been found for many classes of graphs including trees, bounded degree graphs, planar graphs, and interval graphs. In his recent text on graph representation, Spinrad [21] has generalized the idea of an implicit representation by asking if families of graphs with $2^{\Theta(\phi(n))}$ members on n vertices have adjacency labelling schemes using $O(\frac{\phi(n)}{n})$ bits per vertex. We observe that an adjacency labelling scheme of a family of graphs provides a unique representation for each of the members of the family, so the number of bits required by an adjacency labelling scheme is at least the number of bits required to represent all of the members uniquely; in particular, a family of graphs with $2^{\Theta(\phi(n))}$ members on n vertices requires a labelling of size $\Theta(\phi(n))$ to uniquely represent each of the members on n vertices. Therefore, generalized implicit representations are balanced space-optimal adjacency labelling schemes. Note that the previously described adjacency labelling scheme devised from adjacency matrices is a generalized implicit representation, and hence a balanced space-optimal adjacency labelling scheme, for any family of graphs having $2^{\Theta(n^2)}$ members on n vertices. Such families include bipartite graphs, chordal graphs, and the class of all graphs.

The terms "generalized implicit representation" and "adjacency labelling scheme" used by Spinrad [21] and Peleg [18], respectively, do not precisely capture the essence of what they are intended to define when considered in

the context of one another. The term "generalized implicit representation" directly references the ability to determine the adjacency of two vertices *implicitly* from their labels, however, put in the wider context of informative labelling schemes, it is no longer evident that adjacency is the property on which we are being informed. Moreover, the term "generalized implicit representation" is used to capture the properties of space-optimality and balance, neither of which are evident from the term itself. On the other hand, the term "adjacency labelling scheme" makes evident the underlying interest in vertex adjacency, but overlooks the fact that the adjacency of two vertices can be determined *implicitly* from their labels. As such, what is defined by Peleg to be an "adjacency labelling scheme" might more accurately be called an "implicit adjacency labelling scheme"; similarly, what Spinrad defines to be a "generalized implicit representation" might more accurately be termed a "balanced space-optimal implicit adjacency labelling scheme". Although longer, these terms better capture the properties of what is being defined and in doing so offer a unified terminology for researchers unfamiliar with the subject area. We will use this new terminology throughout the remainder of this article, except when referring to earlier works in a historical context.

In many applications the underlying topology is constantly changing and we desire algorithms which can accommodate these changes without having to process the new topology from scratch. At present, algorithms for finding implicit adjacency labelling schemes are static; that is, if the graph provided to the algorithm is changed then the algorithm must process the new graph from scratch. The dynamic version of this problem was mentioned by Kannan et al. [11] in their original work on implicit representations, however, no formulation of the problem is attempted. At most, the authors suggest that the addition or deletion of a vertex or an edge should require only a "quick" update of the labels in order to obtain an implicit representation of the new graph. To date, the paper of Brodal and Fagerberg [8] stands as the only publication on this dynamic problem, however, their paper did not formally define the notion of a dynamic implicit adjacency labelling scheme. Specifically, they develop a dynamic implicit adjacency labelling scheme which handles the addition and deletion of single edges and vertices in graphs of bounded arboricity, providing the bounded arboricity is maintained. As a continuation of the work of Brodal and Fagerberg, there is a need for more formal discussion on dynamic implicit informative labelling schemes, as well as the development of dynamic implicit informative labelling schemes for more classes of graphs. In particular, algorithms developed for dynamic implicit informative labelling schemes should incorporate some form of error detection; that is, the algorithms should recognize when the modified

graph is no longer a member of the family under consideration. In section 2 we formally introduce the theory of dynamic implicit informative labelling schemes and in Section 3 we present dynamic implicit adjacency labelling schemes for r-minoes; as defined by Metelsky and Tyshkevich [14], a graph is an r-mino if none of its vertices belongs to more than r maximal cliques.

By further studying dynamic implicit adjacency labelling schemes we hope to expand the applicability of implicit labelling schemes to real world problems. In particular, implicit labelling schemes have direct applications to the efficiency of XML (Extensible Markup Language) search engines [12]. Web documents conforming to the XML standard can be viewed as a tree with nested nodes corresponding to individual words, phrases, or sections of the document. Using implicit informative labelling schemes, an XML search engine can assign labels to each of these nodes allowing relationships such as ancestor, parent, and sibling to be determined using only the labels of the nodes. This allows the search engine to answer web queries without repeatedly accessing the file itself. Moreover, by employing dynamic schemes the search engine will no longer have to recalculate the labels of the nodes when a small change is made to the XML document. Applications of implicit labelling schemes to communication networks have also been discussed in [12], [17], and [22].

2 Dynamic Implicit Adjacency Labelling Schemes

We begin by defining a dynamic implicit f-labelling scheme.

Definition 2.1 *Consider a function $f(S, G)$ defined on sets of vertices S of fixed but arbitrary finite graphs G. A dynamic implicit f-labelling scheme of a family $\mathcal{G}$ of finite graphs is a tuple (M, D, Δ, C) defined as follows.*

- *(M, D) is an implicit f-labelling scheme of $\mathcal{G}$.*
- *Δ is a set of functions which map graphs in $\mathcal{G}$ to other graphs.*
- *C is a polynomial time relabelling algorithm whose input is a pair (δ, L_G), where $\delta \in \Delta$, $G \in \mathcal{G}$, and L_G is a (D, f)-correct labelling of V_G from $\mathcal{L}_G$ (defined shortly); in particular, providing $\delta(G) \in \mathcal{G}$, C assigns a new (D, f)-correct labelling to $V_{\delta(G)}$ based upon the labelling L_G of G. Note that C need not be deterministic; accordingly, let $\mathcal{C}_{\delta, L_G}$ be the set of labellings of $V_{\delta(G)}$ which can be assigned by C on input (δ, L_G). For each G in $\mathcal{G}$ we define the family $\mathcal{L}_G$ of (D, f)-correct*

labellings of V_G by $L_G \in \mathcal{L}_G$ if and only if $L_G \in \mathcal{M}_G$ or there exists G^* in $\mathcal{G}$, δ in Δ, and L_{G^*} in $\mathcal{L}_{G^*}$ such that $\delta(G^*) = G$ and $L_G \in \mathcal{C}_{\delta, L_{G^*}}$.

Moreover, we say that the dynamic implicit f-labelling scheme is error-detecting if, given any input (δ, L_G), C is able to determine when $\delta(G) \notin \mathcal{G}$.

In a less formal context, C can be considered as the composition of algorithms required by the graph operations found in Δ. For instance, if Δ permitted the addition or deletion of any edge from a graph, we might consider C to be comprised of two algorithms, AddEdge(e,L_G) and DeleteEdge(e,L_G), which use a labelling L_G to relabel $G + e$ and $G - e$, respectively. Again, note that the algorithms AddEdge and DeleteEdge are provided input about the graph only in the form of vertex labels; in turn, these algorithms output labellings of the vertices of $G + e$ and $G - e$, respectively. Moreover, in practice we are not interested in maintaining a labelling for every graph in the family, rather, we use the labelling of a graph to determine a labelling of a slightly modified graph, discarding the labelling of the original graph in the process. In this sense we can omit the labelling from the input of the algorithms as these algorithms are directly modifying the labelling of the graph under consideration; that is, the above algorithms might be presented as AddEdge(e) and DeleteEdge(e).

We have seen how an implicit f-labelling scheme can be created for any function f when we allow sufficiently large labels; similarly, sufficiently weak choices of M, Δ, and C will result in a dynamic implicit f-labelling scheme for any function f. As a result, there are several ways in which one might judge the quality of a dynamic implicit f-labelling scheme. First of all, we might judge a dynamic scheme according to the time taken by C on input (δ, L_G) relative to the time taken to label $\delta(G)$ by the fastest labelling algorithm of a non-dynamic implicit f-labelling scheme. Specifically, the purpose of the dynamic scheme is to provide quick updates of the labels, thereby, if there is a non-dynamic scheme which can generate the labels in equal or better time, even from scratch, then there is no advantage gained by using the dynamic scheme. Secondly, since a dynamic implicit f-labelling scheme includes an implicit f-labelling scheme, we might also judge a dynamic scheme according to the size of the labels generated by M and C. For example, consider that the implicit adjacency labelling scheme developed using adjacency matrices can be further developed into a dynamic implicit adjacency labelling scheme. Since this dynamic scheme uses vertex labels of size $O(n)$, any other dynamic implicit adjacency labelling scheme using labels of size $\Omega(n)$ would only be advantageous if it permitted faster updates

of the labels than can be achieved using the dynamic scheme developed from adjacency matrices. Finally, we might judge a dynamic scheme according to the operations contained in Δ. Preferably, Δ will contain the addition and deletion of a single edge or vertex (along with the edges incident with this vertex) which are four fundamental dynamic graph operations. Moreover, using the operations found in Δ, we would like to be able to transform any member of $\mathcal{G}$ into any other member of $\mathcal{G}$ without escaping the class $\mathcal{G}$; however, this may require more than these four fundamental dynamic graph operations.

It should be noted that if $\mathcal{G}$ is a hereditary graph class then these fundamental graph operations are sufficient to transform any member of $\mathcal{G}$ into any other member of $\mathcal{G}$ without escaping the class $\mathcal{G}$; recall that a graph class is said to be hereditary if every vertex induced subgraph of a member of the class is also a member of the class. For each member G of $\mathcal{G}$ there is a sequence $S_G = \{G_0 = \emptyset, G_1, \ldots, G_{|V_G|-1}, G_{|V_G|} = G\}$ of members of $\mathcal{G}$ for which $G_{i-1} = G_i - v_i$, where v_i is a vertex of G_i and $1 \leq i \leq |V_G|$. Thereby, given $G^{(1)}, G^{(2)} \in \mathcal{G}$ we can construct $G^{(2)}$ from $G^{(1)}$ by using the vertex deleting algorithm to transform $G^{(1)}$ into $\emptyset$ via the members of $S_{G^{(1)}}$, then using the vertex adding algorithm to transform $\emptyset$ into $G^{(2)}$ via the members of $S_{G^{(2)}}$.

Continuing with the idea of transforming one graph into another, there is a connection between error-detecting dynamic implicit f-labelling schemes and the problem of recognizing whether a graph belongs to a certain family. Consider a family of graphs $\mathcal{G}$ for which there exists a dynamic implicit f-labelling scheme (M, D, Δ, C) and the recognition problem is polynomial on $\mathcal{G}$. If f allows us to determine the structure of a graph G from any labelling in $\mathcal{L}_G$, then on any input (δ, L_G) C can use f to determine the structure of G and, hence, the structure of $\delta(G)$, in polynomial time. In turn, C can apply a polynomial time recognition algorithm to determine if $\delta(G)$ is in $\mathcal{G}$; thereby, the dynamic implicit f-labelling scheme is error-detecting. Specifically, if f is the adjacency function then f can determine the structure of the graph.

On the other hand, it is more complicated to develop recognition from error-detection. Consider a family of graphs $\mathcal{G}$ for which there exists an error-detecting dynamic implicit f-labelling scheme (M, D, Δ, C). If for any graph G in $\mathcal{G}$

- there exists a graph G^* in $\mathcal{G}$ for which we can determine in polynomial time a polynomial length sequence $S_G = \{G_0 = G^*, G_1, \ldots, G_{k-1}, G_k = G\}$ of members of $\mathcal{G}$, as well as a polynomial length sequence $G^{\Delta} = \{\delta_0, \delta_1, \ldots, \delta_{k-1}\}$ of members of Δ such that $\delta_i(G_i) = G_{i+1}$, for $0 \leq$

$i \leq k-1$

- and there exists a polynomial time function for determining a labelling of G^* which belongs to $\mathcal{L}_{G^*}$

then the recognition problem is polynomial on $\mathcal{G}$. The reason being that we can determine S_G, G^{Δ}, and the labelling for G^* in polynomial time; then transform the labelling of G^* into a labelling for G_{k-1} using a polynomial number of calls of the polynomial time algorithm C, namely $\{C_0, C_1, \ldots, C_{k-1}\}$, where $C_0 = C(\delta_0, L_{G^*})$ and $C_i = C(\delta_i, C_{i-1})$, for $1 \leq i \leq k-1$; and finally resolve the membership of G in $\mathcal{G}$ according to the action of C when it attempts to determine a labelling of G from the labelling of G_{k-1}. If $G \in \mathcal{G}$ then C will determine a labelling of G, otherwise, it will output that $G \notin \mathcal{G}$ since it is an error-detecting algorithm. For example, consider a hereditary graph class with an error-detecting dynamic implicit f-labelling scheme whose graph operation set includes the addition of vertices (along with incident edges). For each member G of the class there is a sequence $S_G = \{G_0 = \emptyset, G_1, \ldots, G_{|V_G|-1}, G_{|V_G|} = G\}$ of members of $\mathcal{G}$ for which $G_{i-1} = G_i - v_i$, where v_i is a vertex of G_i and $1 \leq i \leq |V_G|$. We have a polynomial time labelling for $\emptyset$ and a means to achieve $G_{|V_G|-1}$ from $\emptyset$ using operations in Δ, thereby, the recognition problem is polynomial for the hereditary class.

Given that the algorithms which change the labellings are functions of the change and the labelling only, the vertex labels used in dynamic implicit labelling schemes must contain sufficient information to allow algorithms to update the labellings. In general, the labels used in implicit adjacency labelling schemes do not contain enough information to be used in dynamic implicit adjacency labelling schemes, however, the implicit adjacency labelling schemes of some classes are inherently dynamic. For instance, consider the following implicit adjacency labelling scheme for trees. Let T be a tree on n vertices. We arbitrarily assign to T a root and give each vertex a unique prelabel from $\{1, \ldots, n\}$. We now obtain an implicit adjacency labelling scheme for T by giving each vertex v of T the label (prelabel(v), prelabel(parent(v))). The adjacency of two vertices v_1 and v_2 can be determined using only their labels by a polynomial time algorithm D which checks if prelabel(v_1) = prelabel(parent(v_2)) or prelabel(v_2) = prelabel(parent(v_1)). Moreover, each label is of size $O(\log n)$, thus making the scheme space-optimal and balanced as the number of trees on n vertices is $2^{\Theta(n \log n)}$. If a new vertex is added to a tree such that the resulting graph is still a tree, then its only neighbour is its parent. Therefore, we can give it the label (prelabel, prelabel of parent) so that the labelling is

still (D,*adjacency*)-correct on the new graph. If a vertex is deleted such that the resulting graph is still a tree, then the vertex must have been a pendant vertex. Therefore, it was not the parent of any other vertex and so deleting the vertex keeps the labelling (D,*adjacency*)-correct on the remaining tree. Although the maintenance of this space-optimal implicit adjacency labelling scheme for trees seems straightforward, there are some underlying shortcomings. One such problem is that when a vertex is added and given a prelabel there must be some way of determining an acceptably small unused prelabel to assign to it. Another such problem is that it is possible to delete too many vertices causing the remaining prelabels to ruin the space-optimality of the labelling. In the work on r-minoes presented in Section 3 we make assumptions which eliminate these problems.

As mentioned, the only work on dynamic implicit adjacency labelling schemes is by Brodal and Fagerberg [8] who develop such schemes for graphs of bounded arboricity. Fundamental to their work is the relationship between arboricity and outdegree orientations where, in particular, a graph with arboricity c has an outdegree-c orientation. Their scheme maintains an outneighbourhood list for each vertex v, denoted by adj[v], where, most importantly, their algorithms include a mechanism to handle outdegree lists which get too big. On a graph with n vertices and arboricity bounded by c, Brodal and Fagerberg's representation supports adjacency testing in $O(c)$ time, edge insertions in $O(1)$ time, and edge deletions in $O(c + \log n)$ time. We present their algorithms for handling the addition and deletion of a single edge from a graph of bounded arboricity c in Figure 1. Unfortunately, these algorithms are built on the assumption that the changes to the graph do not cause its arboricity to exceed c. In their article, Brodal and Fagerberg do describe modified algorithms which can handle unspecified arboricities, however, this results in increased time bounds.

3 Error-Detecting Dynamic Implicit Adjacency Labelling Schemes for r-minoes

In the remainder of this work we consider error-detecting dynamic implicit adjacency labelling schemes for a series of graph classes known as r-minoes. As mentioned in Section 1, Metelsky and Tyshkevich [14] define a graph to be an r-mino if none of its vertices belongs to more than r maximal cliques; this notion of an r-mino is an extension of the idea of a domino, as defined by Kloks et al. [13], in which each vertex belongs to at most two maximal cliques. In their work, Metelsky and Tyshkevich show that the class of r-

```
INSERT((u, v))
 1  adj[u] ← adj[u] ∪ {v}
 2  if |adj[u]| = 4c + 1
 3     then S ← {u}
 4          while S ≠ 0
 5          do w ← Pop(S)
 6             for x ∈ adj[w]
 7             do adj[x] ← adj[x] ∪ {w}
 8                if |adj[x]| = 4c + 1
 9                   then Push(S, x)
10             adj[w] ← ∅

DELETE((u, v))
1  adj[u] ← adj[u] \ {v}
2  adj[v] ← adj[v] \ {u}
```

Figure 1: Algorithms for dynamic implicit representations of graphs of bounded arboricity c.

minoes is the same as the class of line graphs of Helly hypergraphs with rank at most r; recall that the line graph of a hypergraph $H = (V, \mathcal{E})$ is the graph $L(H) = (\mathcal{E}, E')$ for which $ee' \in E'$ if and only if $e \neq e'$ and $e \cap e' \neq \emptyset$; that the hypergraph H is said to satisfy the Helly property if every pairwise intersecting subset $\mathcal{E}'$ of $\mathcal{E}$ is such that $\bigcap_{e \in \mathcal{E}} e \neq \emptyset$; and that the rank of H is the value $\max_{e \in \mathcal{E}} \{|e|\}$.

The challenge in creating dynamic implicit adjacency labelling schemes lies in storing sufficient information in the vertex labels to deduce, at least partially, the structure of the graph in an efficient manner. Without knowing the structure of the graph we cannot determine the structure of the new graph and, in turn, we cannot determine the labels of the new graph. If the label of a vertex v were to contain only the indices of the maximal cliques in which it is contained then it would be impossible to deduce the neighbourhood of v without checking the label of every vertex u in the graph to see if u shared a common maximal clique with v. To overcome this problem we use the vertex labels to maintain a circular (doubly) linked list structure of the vertices in each maximal clique. Not only will a vertex label contain a listing of all the maximal cliques in which it is contained, but for each of

these cliques it will contain the name of the previous and next vertex in the circular linked list for that clique. Using these circular liked lists, we can determine all the vertices in a maximal clique by examining only the labels of the vertices in that clique and, moreover, we will be able to determine the neighbourhood of a vertex by examining only the labels of the vertices in the neighbourhood. Given the labels of two vertices v_1 and v_2 of the graph an evaluation algorithm can determine their adjacency in polynomial time by checking if they are in a common maximal clique. It should also be noted that the labelling algorithm can determine the maximal cliques, and hence the vertex labels, in polynomial time providing r is polynomial in n; we will discuss this in more detail later. Specifically, the labels of our dynamic scheme will contain the following information.

pre: Each vertex is assigned a unique prelabel from $\{1, \ldots, n\}$; *pre* is the prelabel of the vertex.

pre.cliquesin: The number of maximal cliques in which *pre* is contained.

pre.cl: A list of triples containing information on the maximal cliques in which *pre* is contained. Each member $pre.cl_i$ is a triple of the form $(num, nx, prev)$ where *num* is the number assigned to the clique; *nx* is the prelabel of the next vertex after *pre* in the circular doubly linked list of the vertices in clique *num*; and *prev* is the prelabel of the vertex before *pre* in the circular doubly linked list. The index i ranges from 1 to *cliquesin*.

In particular, the label of a vertex is (*pre*: *pre.cliquesin*; *pre.cl*). As an example of this labelling consider the graph presented in Figure 2.

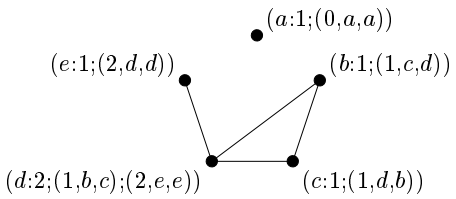

Figure 2: A labelling of a graph obtained using our labelling scheme for 2-minoes (dominoes).

Consider an r-mino which, through some series of modifications, now contains exactly n vertices. If $l(string)$ denotes the size of the representation

of *string* then the size of a label is

$$\begin{aligned}
& l(pre) + l(pre.cliquesin) + \sum_{i=1}^{pre.cliquesin} l(pre.cl_i) \\
\in\ & \mathrm{O}\Big(l(pre) + \log(r) + r\big(l(pre.cl_{i^*})\big)\Big) \\
=\ & \mathrm{O}\Big(l(pre) + \log(r) + r\big(l(pre.cl_{i^*}.num) + l(pre.cl_{i^*}.nx)\big)\Big) \\
=\ & \mathrm{O}\Big(l(pre) + \log(r) + r\big(\log(rn) + l(pre.cl_{i^*}.nx)\big)\Big) \\
=\ & \mathrm{O}\Big(l(pre) + \log(n) + l(pre.cl_{i^*}.nx)\Big),
\end{aligned}$$

where i^* is the value of i for which $l(pre.cl_i)$ is a maximum.

Observe that if the graph had been obtained by the deletion of vertices then it is possible that the largest prelabel of a vertex might be larger than n; as such, let the largest prelabel of a vertex in the graph be L. Thereby, $l(pre)$ and $l(pre.cl_{i^*}.nx)$ are $\mathrm{O}(\log L)$. If $L \in \mathrm{O}(n)$, which we assume hereafter, then the size of a vertex label reduces to $\mathrm{O}(\log(n))$. That is, the graph is represented using $\mathrm{O}(n \log(n))$ bits. The space-optimality of this labelling can be established using an argument found in Spinrad [21] (p. 18) to develop a lower bound on the number of r-minoes on n vertices. Consider a graph with $\frac{n}{2}$ disjoint edges, each of which has one endpoint in $\{1, \ldots, \frac{n}{2}\}$ and the other in $\{\frac{n}{2}+1, \ldots, n\}$. There are $(\frac{n}{2})!$ such graphs, each of which is a member of our family. Yet,

$$(\frac{n}{2})! > \frac{(\frac{n}{2})!}{(\frac{n}{4})!} > (\frac{n}{4})^{\frac{n}{4}} = 2^{\frac{n}{4}\log(\frac{n}{4})} \in \Omega(2^{n\log(n)}),$$

so there are $\Omega(2^{n\log(n)})$ r-minoes on n vertices. Therefore, any labelling which uniquely represents each member of this class, as ours does, requires $\Omega(n\log(n))$ bits thereby proving the space-optimality of our dynamic labelling scheme when $r \in \mathrm{O}(1)$.

3.1 Algorithms used in the dynamic scheme

In the remainder of this work we discuss the graph operations included in our dynamic scheme. In particular, we accommodate the addition or deletion of a vertex (along with its incident edges) and the addition or deletion of an edge.

3.1.1 Deleting a vertex from the graph

One action that we allow on the r-mino is the deletion of a vertex along with its incident edges; the algorithm DeleteVertex found in Figure 3 can be used to maintain the vertex labels in this situation. Letting *pre* be the prelabel of the vertex of the graph to be deleted, DeleteVertex first removes *pre* from each maximal clique in which it is contained. If any of the residual cliques are empty, or no longer maximal, then it removes all references to these residual cliques and frees the clique number for future use using FreeClique. Before finishing, DeleteVertex frees the prelabel *pre* for future use. If $maxcl(pre)$ is the size of the largest maximal clique containing *pre*, then DeleteVertex runs in $O(r^2 \cdot maxcl(pre))$ time. Moreover, DeleteVertex is error-detecting because the vertex induced subgraph of any r-mino is also an r-mino.

3.1.2 Adding a vertex to the graph

Another action we allow is the addition of a vertex along with its incident edges; the algorithm AddVertex found in Figure 4 can be used to maintain the vertex labels in this situation. Letting *pre* be the prelabel of the vertex to be added, AddVertex examines the maximal cliques that contain members of $N(pre)$, the open neighbourhood of *pre*, to determine the maximal cliques to which *pre* belongs. For each maximal clique containing a member of $N(pre)$ we consider the subclique of vertices in $N(pre)$. If the vertices of this subclique are not contained within another maximal clique which has yet to be explored then we will use this subclique to create a maximal clique containing *pre*; otherwise, we will wait until we encounter the same subclique later. Specifically, if the subclique is a maximal clique then we simply add *pre* to the subclique; otherwise, we form a new maximal clique from the subclique and *pre*. If $maxcl(pre)$ is the size of the largest maximal clique containing *pre*, then AddVertex runs in $O(r^3 \cdot maxcl(pre) \cdot |N(pre)|)$ time. Moreover, whenever a vertex is placed in a new clique we check that it is not in more than r maximal cliques, thus making the algorithm AddVertex error-detecting.

Since the class of r-minoes is hereditary, the labelling algorithm associated with the dynamic scheme can iteratively run AddVertex, beginning with the empty graph, in order to determine an initial labelling of any member in the class. Such an algorithm would run in polynomial time, as required by the definition of a dynamic implicit informative labelling scheme.

3.1.3 Deleting an edge from the graph

The third action we allow is the deletion of an edge; the algorithm DeleteEdge found in Figure 5 can be used to maintain the vertex labels in this situation. Letting u and v be the prelabels of the endpoints of the edge to be deleted, DeleteEdge examines the maximal cliques that contain both u and v. For each of these maximal cliques, *clnum*, we need to consider the possibilities of *clnum* $-\{u\}$ and *clnum* $-\{v\}$ being maximal cliques. Specifically, if the vertices in these cliques share another common clique besides *clnum* then we do not add any new cliques; however, if these vertices do not share another common clique then we add a new clique. If $maxcl(u, v)$ denotes the size of the largest maximal clique containing both u and v then DeleteVertex runs in $O(r^2 \cdot maxcl(u, v))$ time. Moreover, whenever a vertex is placed in a new clique we check that it is not in more than r maximal cliques, thus making the algorithm DeleteEdge error-detecting. We present the algorithm at a higher level than we did AddVertex as many of the details can be implemented using similar constructs found in AddVertex.

3.1.4 Adding an edge to the graph

The final action we allow is the addition of an edge; the algorithm AddEdge found in Figure 6 can be used to maintain the vertex labels in this situation. Letting u and v be the prelabels of the endpoints of the edge to be added, AddEdge examines the maximal cliques that contain v. For each maximal clique containing v we consider the subclique of vertices in $N(u)$. If the vertices of this subclique are not contained within another maximal clique which has yet to be explored then we will use this subclique to create a maximal clique containing u; otherwise, we will wait until we encounter the same subclique later. Specifically, if the subclique is a maximal clique then we simply add u to the subclique; otherwise, we form a new maximal clique from the subclique and u. The algorithm AddEdge is very similar to AddVertex. If $maxcl(u)$ and $maxcl(v)$ are the sizes of the largest maximal cliques containing u and v respectively, then DeleteVertex runs in $O(r^3 \cdot max\{maxcl(u), maxcl(v)\})$ time. Moreover, whenever a vertex is placed in a new clique we check that it is not in more than r maximal cliques, thus making the algorithm AddEdge error-detecting. As a final comment, we note the similarity between the algorithm AddEdge and the algorithm AddVertex.

3.2 Maximal cliques, edge clique covers, and intersection representations

We observe that the set of maximal cliques of a graph G constitutes an *edge clique cover* which is a set of complete subgraphs of G which cover E_G. Therefore, an r-mino has an edge clique cover in which each vertex is contained in at most r of these cliques. The converse of this statement is not true; for example, consider the graph G shown in Figure 7. The vertex v is contained in $\frac{|V_G|+1}{2} \in \mathrm{O}(|V_G|)$ maximal cliques, however, the edge clique cover $\{\{v, a_1, \ldots, a_k\}, \{v, b_1, \ldots, b_k\}, \{a_1, b_1\}, \ldots, \{a_k, b_k\}\}$ is such that each vertex is contained in at most two cliques.

Additionally, there exists a dual relationship between edge clique covers and intersection models of graphs. In particular, an intersection model of a graph can be obtained from an edge clique cover by representing each vertex by the set of cliques in the edge clique cover to which it belongs; similarly, an edge clique cover can be obtained by associating a clique with each element used in the intersection model, thereby, a vertex will be contained in a clique of the edge clique cover if the set corresponding to the vertex contains the element associated with that clique. Therefore, an r-mino has an intersection model in which each vertex is represented by a set of size at most r. As with our previous observations regarding edge clique covers, the converse of this statement does not hold.

4 Conclusion

Over the last fifteen years the study of informative labelling schemes has evolved through the efforts of several authors, including Muller [15], Kannan et al. [11], Peleg [18], and Spinrad [21]. In this work we formalize the idea of a dynamic implicit informative labelling scheme and present dynamic schemes for graph classes known as r-minoes. In particular, the graph operations which are permitted in this dynamic scheme for graphs are the addition or deletion of a vertex (and its incident edges) and the addition or deletion of an edge. In developing this dynamic scheme we have employed a technique in which information about the neighbourhood of a vertex is incorporated into the vertex labels via circular linked list structures so as to distribute information about a maximal clique across the labellings of the vertices in that maximal clique.

Future research on dynamic implicit informative labelling schemes will reveal dynamic schemes for additional classes of graphs. By studying these dynamic schemes we will increase the applicability of informative labelling

schemes to real world problems such as internet search engines and communication networks [12, 17, 22].

References

[1] S. Abiteboul, H. Kaplan, and T. Milo, *Compact labeling schemes for ancestor queries*, Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, D.C., USA), 2001, pp. 547–556.

[2] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, *Nearest common ancestors: A survey and a new distributed algorithm*, Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (Winnipeg, Canada), 2002, pp. 258–264.

[3] S. Alstrup and T. Rauhe, *Small induced-universal graphs and compact implicit graph representations*, 43rd Annual Symposium on Foundations of Computer Science (Vancouver, Canada), IEEE, 2002, pp. 53–62.

[4] S. R. Arikati, A. Maheshwari, and C. Zaroliagis, *Efficient computation of implicit representations of sparse graphs*, Discrete Applied Mathematics **78** (1997), no. 1, 1–16.

[5] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, *Compact distributed data structures for adaptive routing*, Proceedings of the 21st Annual ACM Symposium on Theory of Computing (Seattle, USA), 1989, pp. 467–478.

[6] M. A. Breuer, *Coding the vertexes of a graph*, IEEE Transactions on Information Theory **12** (1966), 148–153.

[7] M. A. Breuer and J. Folkman, *An unexpected result in coding the vertices of a graph*, Journal of Mathematical Analysis and Applications **20** (1967), 583–600.

[8] G. S. Brodal and R. Fagerberg, *Dynamic representation of sparse graphs*, Algorithms and Data Structures, Proceedings of the 6th International Workshop (Vancouver, Canada), Lecture Notes in Computer Science, vol. 1663, Springer-Verlag, 1999, pp. 342–351.

[9] W. R. Franklin, *Compressing elevation data*, Advances in Spatial Databases, Proceedings of the 4th International Symposium (Portland,

USA), Lecture Notes in Computer Science, vol. 951, Springer-Verlag, 1995, pp. 385–404.

[10] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz, *Distance labeling in graphs*, Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, D.C., USA), 2001, pp. 210–219.

[11] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, SIAM Journal on Discrete Mathematics **5** (1992), no. 4, 596–603.

[12] H. Kaplan and T. Milo, *Short and simple labels for small distances and other functions*, Algorithms and Data Structures, Proceedings of the 7^{th} International Workshop (Providence, USA), Lecture Notes in Computer Science, vol. 2125, Springer-Verlag, 2001, pp. 246–257.

[13] T. Kloks, D. Kratsch, and H. Müller, *Dominoes*, Graph Theoretic Concepts in Computer Science, Proceedings of the 20^{th} International Workshop (Herrsching, Germany), Lecture Notes in Computer Science, vol. 903, Springer-Verlag, 1995, pp. 106–120.

[14] Y. Metelsky and R. Tyshkevich, *Line graphs of Helly hypergraphs*, SIAM Journal of Discrete Mathematics **16** (2003), no. 3, 438–448.

[15] J. H. Muller, *Local structure in graph classes*, Ph.D. thesis, Georgia Institute of Technology, March 1988.

[16] M. Naor, *Succinct representation of general unlabeled graphs*, Discrete Applied Mathematics **28** (1990), 303–307.

[17] D. Peleg, *Proximity-preserving labeling schemes and their applications*, Graph Theoretic Concepts in Computer Science, Proceedings of the 25^{th} International Workshop (Ascona, Switzerland), Lecture Notes in Computer Science, vol. 1665, Springer-Verlag, 1999, pp. 30–41.

[18] ———, *Informative labeling schemes for graphs*, Mathematical Foundations of Computer Science 2000, Proceedings of the 25^{th} International Symposium (Bratislava, Slovakia), Lecture Notes in Computer Science, vol. 1893, Springer-Verlag, 2000, pp. 579–588.

[19] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal **28** (1985), 5–8.

[20] E. R. Scheinerman, *Local representations using very short labels*, Discrete Mathematics **203** (1999), 287–290.

[21] J. Spinrad, *Efficient graph representation*, Fields Institute Monographs, AMS, Providence, 2003.

[22] M. Thorup and U. Zwick, *Compact routing schemes*, Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures (Heraklion, Greece), 2001, pp. 1–10.

DeleteVertex(pre)
1 **for** $i \leftarrow 1$ **to** $pre.cliquesin$ **do**
2 **if** $pre.cl_i.nx = pre$ or the vertices of clique $pre.cl_i.num$, other than pre, exist in a common clique besides $pre.cl_i.num$ **then**
3 FreeClique($pre.cl_i.num, pre$)
4 **else** RemoveFromClique($pre.cl_i.num, pre$)
5 FreeVertex(pre)

FreeClique($cliquenum, start$)
1 $x \leftarrow start$
2 $y \leftarrow$ the next vertex after x in clique $cliquenum$
3 RemoveFromClique($cliquenum, x$)
4 **while** $y \neq start$ **do**
5 $x \leftarrow y$
6 $y \leftarrow$ the next vertex after x in clique $cliquenum$
7 RemoveFromClique($cliquenum, x$)
8 free $cliquenum$ so that it can be used as the name of a clique in the future

RemoveFromClique($cliquenum, x$)
1 $y \leftarrow$ the next vertex after x in clique $cliquenum$
2 $z \leftarrow$ the vertex before x in clique $cliquenum$
3 **if** $x = y$ **then**
4 remove the reference to clique $cliquenum$ in the label of x
5 **else** remove x from between z and y in clique $cliquenum$
6 $x.cliquesin \leftarrow x.cliquesin - 1$

FreeVertex(v)
1 delete the label of the vertex with prelabel v and free the prelabel v for future use

Figure 3: The algorithm DeleteVertex which updates the labels when a vertex (along with its incident edges) is deleted from the r-mino.

```
AddVertex(N(pre))
 1   pre ← GetPrelabel
 2   toprocess ← the maximal cliques containing members of N(pre)
 3   for each vertex v in N(pre) do
 4       for i ← 1 to v.cliquesin do
 5           toprocess ← toprocess −{v.cl_i.num}
 6           cliquelist ← toprocess
 7           I ← Nil
 8           outside ← 0
 9           for each vertex w in clique v.cl_i.num do
10               if w ∈ N(pre) then
11                   cliquelist ← the maximal cliques in cliquelist that con-
tain w
12                   Push(I, w)
13               else outside ← 1
14           if cliquelist = ∅ then
15               if outside = 0 then
16                   AddToClique(v, pre, v.cl_i.nx, v.cl_i.num)
17               else Push(I, pre)
18                   MakeNewClique(I)

GetPrelabel()
 1   return an unused prelabel for the new vertex

AddToClique(z, x, y, cliquenum)
 1   x.cliquesin ← x.cliquesin +1
 2   CheckRCliques(x)
 3   insert x between z and y in clique cliquenum

CheckRCliques(x)
 1   if x.cliquesin > r then
 2       error "the new graph is no longer an r-mino"

MakeNewClique(S)
 1   cliquenum ← GetCliqueNumber
 2   start ← Pop(S)
 3   x ← start
 4   AddToClique(x, x, x, cliquenum)
 5   while S ≠ Nil do
 6       z ← x
 7       x ← Pop(S)
 8       AddToClique(z, x, start, cliquenum)
```

Figure 4: The algorithm AddVertex which updates the labels when a vertex (along with its incident edges) is added to the r-mino.

```
DeleteEdge(u, v)
 1  cliques ← the numbers of the maximal cliques common to both u
and v
 2  for each maximal clique clnum in cliques do
 3      I ← Nil
 4      for each vertex w in clique clnum other than u and v do
 5          Push(I, w)
 6      FreeClique(clnum, u)
 7      Push(I, v)
 8      if the vertices in I do not share a common clique then
 9          MakeNewClique(I)
10      Pop(I)
11      Push(I, u)
12      if the vertices in I do not share a common clique then
13          MakeNewClique(I)
```

Figure 5: The algorithm DeleteEdge which updates the labels when an edge is deleted from the r-mino.

```
AddEdge(u, v)
 1  toprocess ← the maximal cliques containing v
 2  for i ← 1 to v.cliquesin do
 3      toprocess ← toprocess −{v.cl_i}
 4      cliquelist ← toprocess
 5      I ← Nil
 6      outside ← 0
 7      for each vertex w in clique v.cl_i do
 8          if w ∈ N(u) then
 9              cliquelist ← the maximal cliques in cliquelist that contain
w
10              Push(I, w)
11          else outside ← 1
12      if cliquelist = ∅ then
13          if outside = 0 then
14              AddToClique(v, pre, v.cl_i.nx, v.cl_i.num)
15          else Push(I, pre)
16              MakeNewClique(I)
```

Figure 6: The algorithm AddEdge which updates the labels when an edge is added to the r-mino.

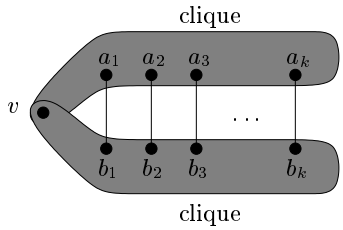

Figure 7: A graph G with an edge clique cover in which each vertex is contained in at most two cliques, but also with a vertex contained in $O(|V_G|)$ maximal cliques.