Linearity Is Strictly More Powerful Than Contiguity for Encoding Graphs

Christophe Crespelle\textsuperscript{1,2,6(✉)}, Tien-Nam Le\textsuperscript{3},
Kevin Perrot\textsuperscript{4,5}, and Thi Ha Duong Phan\textsuperscript{6}

\textsuperscript{1} Universit\`e Claude Bernard Lyon 1, Villeurbanne, France
\textsuperscript{2} CNRS, DANTE/INRIA, LIP UMR CNRS 5668, ENS de Lyon, Universit\`e de Lyon,
Lyon, France
christophe.crespelle@inria.fr
\textsuperscript{3} ENS de Lyon, Universit\`e de Lyon, Lyon, France
tien-nam.le@ens-lyon.fr
\textsuperscript{4} Universidad de Chile, DIM, CMM UMR CNRS 2807, Santiago, Chile
\textsuperscript{5} Aix-Marseille Universit\`e CNRS, LIF UMR 7279, 13288 Marseille, France
kevin.perrot@lif.univ-mrs.fr
\textsuperscript{6} Institute of Mathematics, Vietnam Academy of Science and Technology,
18 Hoang Quoc Viet, Hanoi, Vietnam
phanhaduong@math.ac.vn

Abstract. Linearity and contiguity are two parameters devoted to graph encoding. Linearity is a generalisation of contiguity in the sense that every encoding achieving contiguity \( k \) induces an encoding achieving linearity \( k \), both encoding having size \( \Theta(k.n) \), where \( n \) is the number of vertices of \( G \). In this paper, we prove that linearity is a strictly more powerful encoding than contiguity, i.e. there exists some graph family such that the linearity is asymptotically negligible in front of the contiguity. We prove this by answering an open question asking for the worst case linearity of a cograph on \( n \) vertices: we provide an \( O(\log n/\log \log n) \) upper bound which matches the previously known lower bound.

1 Introduction

One of the most widely used operation in graph algorithms is the \textit{neighbourhood query}: given a vertex \( x \) of a graph \( G \), one wants to obtain the list of neighbours of \( x \) in \( G \). The classical data structure that allows to do so is the adjacency lists. It stores a graph \( G \) in \( O(n + m) \) space, where \( n \) is the number of vertices of \( G \) and \( m \) its number of edges, and answers a neighbourhood query on any

This work was partially funded by a grant from Région Rhône-Alpes and by the delegation program of CNRS.
This work was partially funded by the Vietnam Institute for Advanced Study in Mathematics (VIASM) and by the Vietnam National Fondation for Science and Technology Developement (NAFOSTED).
This work was partially funded by Fondecyt Postdoctoral grant 3140527 and Núcleo Milenio Información y Coordinación en Redes (ACGO).

© Springer International Publishing Switzerland 2015
F. Dehne et al. (Eds.): WADS 2015, LNCS 9214, pp. 212–223, 2015.
DOI: 10.1007/978-3-319-21840-3_18
vertex $x$ in $O(d)$ time, where $d$ is the degree of vertex $x$. This time complexity is optimal, as long as one wants to produce the list of neighbours of $x$. On the other hand, in the last decades, huge amounts of data organized in the form of graphs or networks have appeared in many contexts such as genomic, biology, physics, linguistics, computer science, transportation and industry. In the same time, the need, for industrials and academics, to algorithmically treat this data in order to extract relevant information has grown in the same proportions. For these applications dealing with very large graphs, a space complexity of $O(n+m)$ is often very limiting. Therefore, as pointed out by [13], finding compact representations of a graph providing optimal time neighbourhood queries is a crucial issue in practice. Such representations allow to store the graph entirely in memory while preserving the complexity of algorithms using neighbourhood queries. The conjunction of these two advantages has great impact on the running time of algorithms managing large amount of data.

One possible way to store a graph $G$ in a very compact way and preserve the complexity of neighbourhood queries is to find an order $\sigma$ on the vertices of $G$ such that the neighbourhood of each vertex $x$ of $G$ is an interval in $\sigma$. In this way, one can store the order $\sigma$ on the vertices of $G$ and assign two pointers to each vertex: one toward its first neighbour in $\sigma$ and one toward its last neighbour in $\sigma$. Therefore, one can answer adjacency queries on vertex $x$ simply by listing the vertices appearing in $\sigma$ between its first and last pointer. It must be clear that such an order on the vertices of $G$ does not exist for all graphs $G$. Nevertheless, this idea turns out to be quite efficient in practice and some compression techniques are precisely based on it [1–4,11]: they try to find orders of the vertices that group the neighbourhoods together, as much as possible.

Then, a natural way to relax the constraints of the problem so that it admits a solution for a larger class of graphs is to allow the neighbourhood of each vertex to be split in at most $k$ intervals in order $\sigma$. The minimum value of $k$ which makes possible to encode the graph $G$ in this way is a parameter called contiguity [9] and denoted by $\text{cont}(G)$. Another natural way of generalization is to use at most $k$ orders $\sigma_1, \ldots, \sigma_k$ on the vertices of $G$ such that the neighbourhood of each vertex is the union of exactly one interval taken in each of the $k$ orders. This defines a parameter called the linearity of $G$ [6], denoted $\text{lin}(G)$. The additional flexibility offered by linearity (using $k$ orders instead of just 1) results in a greater power of encoding, in the sense that if a graph $G$ admits an encoding by contiguity $k$, using one linear order $\sigma$ and at most $k$ intervals for each vertex, it is straightforward to obtain an encoding of $G$ by linearity $k$: take $k$ copies of $\sigma$ and assign to each vertex one of its $k$ intervals in each of the $k$ copies of $\sigma$.

As one can expect, this greater power of encoding requires an extra cost: the size of an encoding by linearity $k$, which uses $k$ orders, is greater than the size of an encoding by contiguity $k$, which uses only 1 order. Nevertheless, very interestingly, the sizes of these two encodings are equivalent up to a multiplicative constant. Indeed, storing an encoding by contiguity $k$ requires to store a linear ordering of the $n$ vertices of $G$, i.e. a list of $n$ integers, and the bounds of each of the $k$ intervals for each vertex, i.e. $2kn$ integers, the total size of the encoding