Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Space-efficient acyclicity constraints a declarative pearl ☆

Taus Brock-Nannestad

Inria & LIX/École polytechnique, France

ARTICLE INFO

Article history:
Received 21 July 2016
Received in revised form 18 September 2017
Accepted 30 October 2017
Available online xxxx

Keywords:
Acyclicity
Graph embeddings
Turning numbers

ABSTRACT

Many constraints on graphs, e.g. the existence of a simple path between two vertices, or the connectedness of the subgraph induced by some selection of vertices, can be straightforwardly represented by means of a suitable *acyclicity* constraint. One method for encoding such a constraint in terms of simple, local constraints uses a 3-valued variable for each edge, and an $(N + 1)$-valued variable for each vertex, where N is the number of vertices in the entire graph. For graphs with many vertices, this can be somewhat inefficient in terms of space usage.

In this paper, we show how to refine this encoding into one that uses only a single bit of information, i.e. a 2-valued variable, per vertex, assuming the graph in question is planar. More generally, for graphs that are embeddable in genus g (i.e. on a torus with g handles), we show that $2g + 1$ bits per vertex suffices. We furthermore show how this same constraint can be used to encode *connectedness* constraints, and a variety of other graph-related constraints.

1. Introduction

In this paper, we aim to present an "encoding pearl" that shows how to encode various graph constraints in terms of an acyclicity constraint, and also how to decompose such an acyclicity constraint into a space-efficient (in terms of the combined sizes of the variable domains) collection of low-level constraints.

Our acyclicity constraint can be seen as a refinement of an intuitive, obviously correct, but space-inefficient constraint that prevents cycles by enforcing a total ordering on the vertices occurring in a path. Similar encodings for the Hamiltonian path problem can be seen in the work of Prestwich [6], and a refinement of that approach in Velev and Gao [11]. Moreover, a similar constraint is used to encode the Travelling Salesman problem as an integer linear program in Miller et al. [4]. To the best of our knowledge, the space-optimised constraint we present here is novel.

We will present our constraints both using a high-level, prose description, but also in terms of the more explicit language of *finite linear integer constraints*. We choose this as the target for our encoding because of its flexibility — we will freely make use of the fact it straightforwardly permits the encoding of e.g. conditionals and reified constraints.

In many cases, using a specialised acyclicity constraint, such as the one presented by Gebser et al. [2], will be more efficient for finding solutions to constraint satisfaction problems, as it can use domain-specific knowledge to propagate constraints in a way that a naïve encoding may not be able to. On the other hand, there are often large gains to be had from encoding a problem using high-level constraints into e.g. a boolean satisfiability (SAT) problem, as seen in the *Sugar* CSP solver [10].

☆ Extended version of "Space-efficient Planar Acyclicity Constraints — A Declarative Pearl" [1].

E-mail address: tausbn@gmail.com.

https://doi.org/10.1016/j.scico.2017.10.016
0167-6423/

An alternative approach to preventing cycles is Counterexample-Guided Abstraction Refinement (CEGAR), where the task of detecting and preventing cycles is not handled internally in the solver. Instead, if a purported solution contains cycles, these are excluded by adding further constraints, and the constraint solver is invoked once more. This approach, again in the setting of the Hamiltonian cycle problem, is presented by Soh et al. [8].

Given the above considerations, we make no claims about the *real-world* efficiency of the solution we present in this paper, and rather present it as a neat theoretical *curiosity*.

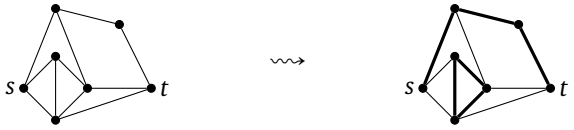
The remainder of the paper is structured as follows. In Section 2, we show how acyclicity plays a crucial role in encoding path constraints. In Section 3, we present a straightforward but space-inefficient encoding of such a constraint, in a subset of planar graphs which we call *grid graphs*, followed by a few improvements in Section 4. In Section 5, we present our optimised encoding, and we prove its correctness in Section 6. In Section 7, we show how to extend the results concerning grid graphs to general planar graphs without a loss in efficiency. In Sections 8 and 9 we extend the encoding to graphs embeddable in genus g. In Section 10, we consider other kinds of graph constraints, and show how these can also be encoded using our acyclicity constraint. We conclude in Section 11.

2. Making use of acyclicity

In this section, we will show exactly how an acyclicity constraint can be used to correctly enforce certain constraints on graphs. First, let us assume we are given some fixed graph G with two distinguished vertices s and t, and that we wish to select some subset of the edges so that they make up a single simple path from s and t. We will associate a variable to each edge of the graph, and say this variable has the value 1 if the edge is selected, and 0 otherwise. To ensure that the path is simple, we can add constraints that restrict the number of selected edges meeting a vertex as follows:

- Around s and t, we require that there is exactly one selected edge.
- For any other vertex v, we require that the number of selected edges around v is either 0 or 2.

These constraints already eliminate many incorrect selections, but unfortunately not all of them. The problem is that the above constraints ensure that the solution must be path-like around each vertex, but this is not enough:

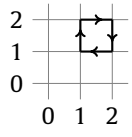


Any part of a cycle is *also* path-like around each vertex in the cycle, hence we may get spurious cycles with just the above constraints. At this point it should hopefully be clear that if we find some way of preventing cycles from appearing in our assignment of edges, we may ensure that the solution consists of only a single, simple path.

Before we present a way of encoding such an acyclicity constraint, we will briefly remark on two simplifying assumptions we can make in this setting. First of all, we may assume that the graph in question is simple, i.e. there is at most one edge between any two vertices, and no edges from a vertex to itself. This can be justified by noting that if we have selected two edges that have the same start and end vertices, then we have already created a cycle. Similarly, any edge that is not part of any cycle of the graph can also be disregarded for the purposes of enforcing acyclicity. Thus, in the remainder of this paper, we will assume that the underlying graph is simple and bridgeless.

3. A basic encoding of the acyclicity constraint

First, we'll describe an inefficient, but hopefully intuitive encoding of an acyclicity constraint. This presentation of the acyclicity constraint closely follows the work of Tamura [9]. To simplify the presentation, we'll only present the results in this section for a very simple and well-behaved graph. The graph we will consider has a vertex for each point $(i, j) \in \mathbb{Z} \times \mathbb{Z}$, and edges between any two vertices that are a unit distance apart. For the variables representing these paths, we will use $h_{i,j}$ for the horizontal edge extending rightwards from the vertex at (i, j), and $v_{i,j}$ for the edge extending vertically at this vertex. Furthermore, we will in certain situations rely on these edges having values that indicate not just whether they are selected, but also marking them with a specific direction. In this case, we will assume that the variables $h_{i,j}$ and $v_{i,j}$ may take on values from the set $\{-1, 0, 1\}$. A negative value indicates that the direction of the edge is to the left or down, and a positive value indicates that the direction of the edge is to the right or up. Thus,



is represented by $h_{1,1} = -1,\ v_{1,1} = 1,\ h_{1,2} = 1,\ v_{2,1} = -1$

To avoid having to consider truly infinite graphs, we will assume that all variables that are indexed by positions (i, j) take on the default value 0 on all but finitely many points. This essentially restricts us to working within a finite subgraph of the

Please cite this article in press as: T. Brock-Nannestad, Space-efficient acyclicity constraints a declarative pearl, Sci. Comput. Program. (2017), https://doi.org/10.1016/j.scico.2017.10.016