Available online at www.sciencedirect.com
SCIENCE DIRECT
Discrete Applied Mathematics 154 (2006) 1380–1391
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

A combinatorial algorithm for weighted stable sets in bipartite graphs

Ulrich Faigle[a], Gereon Frahling[b]

[a]*ZAIK, Center for Applied Computer Science, University of Cologne, Cologne, Germany*
[b]*Heinz Nixdorf Institute, University of Paderborn, Paderborn, Germany*

Received 20 June 2003; received in revised form 19 July 2004; accepted 6 May 2005
Available online 25 January 2006

Abstract

Computing a maximum weighted stable set in a bipartite graph is considered well-solved and usually approached with preflow-push, Ford–Fulkerson or network simplex algorithms. We present a combinatorial algorithm for the problem that is not based on flows. Numerical tests suggest that this algorithm performs quite well in practice and is competitive with flow based algorithms especially in the case of dense graphs.

Keywords: Stable set; Bipartite graph; Tree solution

1. Introduction

The problem of finding an antichain of maximal weight in a (weighted) poset is (well-)known to be equivalent with the problem of computing a maximal weighted stable (or "independent") set of nodes in a bipartite graph. König and Egerváry's discovery that maximum size stable sets in bipartite graphs "behave nicely" and can be obtained from maximal matchings is a classical result in graph theory (and perhaps the root of combinatorial optimization) (see, e.g., [2,8]). Using a network flow formulation, the stable set problem is soluble in $O(n^{5/2})$ time in the unweighted case and $O(n^4)$ in the case of rational weights (where n denotes the number of nodes in the graph) (see, e.g., [1]).

We propose here a conceptually new algorithmic approach to the stable set problem that is *not* based on the formulation as a network flow problem. We are motivated by the algorithm of Lerchs and Grossmann [7] ("LG-algorithm") for the following problem: compute a maximal ideal (a.k.a. "down set") in a weighted poset (see the analysis of Hochbaum [5]) or, equivalently, a maximally weighted closure in a directed graph.

The LG-algorithm proceeds in a rather combinatorial way and, at first sight, does not seem to rely on flows in auxiliary graphs. Instead, it considers a spanning tree in the graph, solves the closure problem with respect to this spanning tree, adds and deletes edges to pass to a new spanning tree and finds a new optimal solution in that modified tree. Iterating this procedure, the algorithm finally arrives at a tree solution that is feasible in the original graph (and hence overall optimal).

E-mail addresses: faigle@zpr.uni-koeln.de (U. Faigle), frahling@upb.de (G. Frahling).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.05.037

The algorithm we present here for the maximum weight stable set problem in bipartite graphs follows the same philosophy. We consider a spanning tree in the graph and determine a maximal stable set relative to this tree. If this solution is not feasible for the original problem (i.e., not stable in the original bipartite graph), we find a restricting edge which has not yet been considered during the computation, add it to the tree, delete another edge and compute a solution in the modified tree. As we will show, the algorithm will end with a solution that is feasible (and optimal) for our original problem.

Hochbaum [5] was able to exhibit the LG-algorithm as a flow algorithm. In spite of the similarity of our algorithmic approach, however, we have not been able to interpret our algorithm as a network flow method.

In the unweighted case we can guarantee a computation time of $O(n^4)$ and in the case of integral weights bounded by K a time complexity of $O(K \cdot n^4)$. These theoretical worst case bounds are not as good as the ones known for flow algorithms. On the other hand, our numerical tests indicate that our simple and combinatorial algorithm may perform very well in practice. We have compared implementations of various state-of-the-art algorithms for the stable set problem with our algorithm and report in typical results in Section 4. Especially in the case of dense graphs our algorithm appears to be considerably faster than network algorithms.

We review in Section 2 the basic definitions of bipartite graphs as well as the network flow model for the problem so that the difference between our model and the standard approach becomes more clear. Our algorithm is described in Section 3.

2. Maximum weight stable sets in bipartite graphs

2.1. Definitions

A *bipartite graph* is a graph $G = (V, E)$ whose node set V can be partitioned into blocks $\mathscr{A}$ and $\mathscr{B}$ such that all edges have one endpoint in $\mathscr{A}$ and the other in $\mathscr{B}$.

A *stable set* in $G = (V, E)$ is a subset $M \subseteq V$ such that no edge $e \in E$ has both endpoints in M. (For example, both blocks $\mathscr{A}$ and $\mathscr{B}$ of the node partition of a bipartite graph are stable sets.)

Given a weight function $c : V \to \mathbb{R}_+$, we seek to maximize the total weight

$$c(M) := \sum_{v \in M} c(v)$$

over the collection of stable sets M of G (see Fig. 1).

2.2. Solving the problem with network flows

In order to obtain a network flow formulation of the stable set problem, one constructs an oriented auxiliary graph $G_F = (V_F, E_F)$ whose edges are oriented from $\mathscr{B}$ to $\mathscr{A}$ with additional nodes and edges in the following way:

$$V_F = V \cup \{s, t\},$$
$$E_F = E \cup \{(s, b) | b \in B\} \cup \{(a, t) | a \in A\},$$

where all original edges between $\mathscr{A}$ and $\mathscr{B}$ are considered to be directed from $\mathscr{B}$ to $\mathscr{A}$.

Next we define capacities $c(e)$ on all edges $e \in E_F$ as follows (see Fig. 2):

$$c(v, w) = \begin{cases} c(v) & \text{if } w = t, \\ c(w) & \text{if } v = s, \\ \infty & \text{otherwise.} \end{cases}$$

An (s, t)-*cut* in G_F is a partition of the node set V_F into two sets S and T with $s \in S$ and $t \in T$. Its *capacity* is defined as

$$c(S, T) = \sum_{v \in S, w \in T} c(v, w).$$