On the Minimum Labelling Spanning *bi*-Connected Subgraph problem

José Andrés Moreno Pérez[1], Sergio Consoli[2]

[1] Department of Computing Engineering, Universidad de La Laguna, Tenerife, Spain
jamoreno@ull.edu.es

[2] ISTC/STLab, National Research Council (CNR), Catania, Italy
sergio.consoli@istc.cnr.it

Abstract

We introduce the minimum labelling spanning bi-connected subgraph problem replacing connectivity by bi-connectivity in the well known Minimum Labelling Spanning Tree problem. A graph is bi-connected if, for every two vertices, there are, at least, two vertex-disjoint paths joining them. The problem consists in finding the spanning bi-connected subgraph or block with minimum set of labels. We adapt the exact method of the minimum labelling spanning tree problem to solve this problem and the basic greedy constructive heuristic, the Maximum Vertex Covering algorithm. This procedure is a basic component in the application of metaheuristics to solve the problem.

1 Introduction

A labelled undirected graph is a graph where each edge has a label from a finite set of labels. A well known problem in labelled graphs is the Minimum Labelling Spanning Tree (MLST) problem consisting of finding the spanning tree that uses the minimum set of labels [3]. Other labelling problems has been proposed in the literature, some derived from the MLST problem. One of the most recently studied is the kLSF problem where the purpose is to find an optimal forest, therefore relaxing the connectivity property [4]. We propose, in the opposite direction, to deal with bi-connectivity instead of connectivity. An undirected graph is bi-connected if any pair of vertices are joined by two vertex-disjoint paths and, equivalently, if it remains connected after dropping any single vertex. The bi-connected graphs or networks have important applications in robustness of transport, communication and social networks. This condition guarantees that such networks remain connected in the event of a node failure. The shortest bi-connected network joining a set of vertices is a cycle or a ring passing throws all of them. For this reason the first computer networks in the 80's were based on rings.

2 Bi-connectivity

An undirected graph $G = (V, E)$ is connected if and only if for any pair of vertices there is, at least, a joining path. The connected components of a graph are the maximal connected subgraphs. The connected components of a graph constitute a partition of its set of vertices and edges. A connected graph has only one connected component. A spanning tree is obtained from a connected graph by iteratively removing and edge from a cycle of the graph until no cycles exist.

In the other hand, an undirected graph G is bi-connected if and only if for any pair of vertices there are, at least, two vertex-disjoint joining paths (i.e.; two paths without a common vertex, excluding the two joined vertices). A cut vertex is a vertex whose removal disconnects the graph. The bi-connectivity is equivalent to the nonexistence of cut vertex. An isolated vertex is assumed to be bi-connected. The bi-connected components or *blocks* of a graph G are the maximal bi-connected subgraphs of G. A cut vertex belongs to more than one block; therefore the blocks are not disjoints in terms of sets of vertices. On the other side, the blocks are disjoint as set of edges, but an edge could not be in any block. An edge that joints two different blocks is a bridge; a bridge is not in any block. An edge that is not a bridge joins two vertices of the same block. The set of bridges and blocks (as set of edges) constitutes a partition of the set of edges.

3 Formulation of the problem

A labelled undirected graph $(G, \mathcal{L}) = (V, E, \mathcal{L})$ consists of an undirected graph $G = (V, E)$ and a finite set of labels $\mathcal{L}$ where every edge $e \in E$ has an unique label $l(e) \in \mathcal{L}$. The well-known minimum labelling spanning tree (MLST) problem introduced in [1, 6] consists of finding the spanning tree T^* of the labelled graph with minimum number of different labels. Given a spanning tree $T = (V, F)$ of G, let $L(T) = \{l(e)|e \in F\})$. The MLST is the spanning tree that minimizes $|L(T)|$. The problem can be alternatively formulated terms of set of labels. Given a subset of labels $L \subseteq \mathcal{L}$, the corresponding graph is $G(L) = (V, E(L))$ where $E(L) = \{e \in E|l(e) \in L\}$. The problem is to determine the set of labels such that the corresponding graph is connected and has the minimum possible size. That is the goal is to find the set $L^* \subseteq \mathcal{L}$ that minimizes $|L|$, where $L \subseteq \mathcal{L}$ and $G(L)$ is connected. To get the MLST from $G(L^*)$, iteratively remove an edge from a cycle until no cycles exist.

We now introduce the Minimum Labelling Spanning Block (MLSB) problem, which consists of finding the spanning block B^* having the minimum number of distinct labels. Alternatively, the problem is to find the set of labels such that the corresponding graph is bi-connected with minimum number of labels. It is to find the set $L^* \subseteq \mathcal{L}$ that minimizes $|L|$, where $L \subseteq \mathcal{L}$ and $G(L)$ is bi-connected.

4 Solution algorithms

The exact approach for the MLST problem [2] is adapted to solve the MLSB problem. The method works in the space of sets of labels searching for the minimum size feasible solution. The algorithm performs a branch and prune procedure in the partial solution space based on a recursive procedure that attempts to find an smaller solution from the current incomplete solution. The main program of the exact method calls the recursive procedure with an empty set of labels, and iteratively stores the smallest feasible solution to date, say L^*. The key procedure of the method is a subroutine that determines if the graph $G(L)$ is connected or not. To adapt the method to the MLSB problem, this procedure is replaced by a subroutine to determine if $G(L)$ is bi-connected or not. For example a Depth First Search (DFS) can be used to determine connectivity and bi-connectivity of the graph. The number of sets tested, and therefore the running time, can be shortened by pruning the search tree using simple rules.

Graph traversal algorithms [7] use the edges to traverse the graph and visit its vertices. Each time a vertex v is visited, the out going edges $[v, w]$ are included in a list C of candidate edges to be traversed. When an edge $[v, w]$ is included in the list it is oriented from v to w. The algorithm starts with an empty list C and visiting a vertex. At each step, the algorithm selects an edge $[v, w]$ from C to be traversed. If w is an unvisited then edge $[v, w]$ is traversed and w visited. Otherwise, $[v, w]$ is a back edge and not traversed. Traversal algorithm stops when C becomes empty. The traversed edges constituted a tree rooted at the starting vertex. Depth First Search (DFS) and Breath First Search (BFS) are strategies to implement the traversal algorithm. DFS selects the edges from C in LIFO (Last-In-First-Out) order and BFS do it in FIFO order. DFS manage list C as a stack and BFS as a queue.

DFS and BFS are linear in the number of edges and are the basis for many graph-related algorithms, including topological sorts, planarity, and connectivity testing. Both may be used to determine if a graph is connected and, in the negative case, to obtain its connected components. For this purpose, start the traversal iteratively from a non visited vertex, until all the vertices are then visited. The vertices of each connected component are those visited in each run of the algorithm. The number of connected components is the number of runs. BFS if often used to determine the shortest path from the root to the rest of the vertices. DFS was extended in [5] to determine if a graph is bi-connected and, in the negative case, to obtain its blocks. DFS finds the cut vertices of the graph as follows. The root of the DFS tree is a cut vertex if it has more than one outgoing tree edge. A vertex v, which is not the root, is a cut vertex if it has a child w such that no back edge starting in the subtree of w reaches an ancestor of v. It is determined during the execution of the DFS by using the order in which the vertices are visited. The DFS algorithm recursively get the first visited node that is reachable from v by using a directed path having at most one back edge; i.e., the last edge of the path. If that vertex was not visited before v by DFS, then v is a cut

vertex. DFS traverse the edges of each block consecutively. Therefore, the blocks are obtained using a stack to keep track of the edges being traversed.

The Maximum Vertex Covering Algorithm (MVCA) is one of the first heuristic for MLST problem [1, 6] and is used as basic component of many metaheuristics applied to the problem. It is a greedy constructive algorithm that selects the labels using the number of connected components as greedy function to minimize. The MVCA for the MLSB problem starts with the empty set of labels, and then each vertex is a block and a connected component. Then the algorithm iteratively adds a label, selected by using the number of blocks plus the number of connected components as greedy function to minimize. To know the number of blocks and components when a label is added, the corresponding edges are included one by one. Note that each block is in only one connected component, so every component has its own blocks. If the components and blocks are known when an edge is added, the new number of blocks and components of the resulting graph are computed as follows. First, if the edge joins two vertices of the same block, then the components and blocks do not change. Second, if the edge joins two vertices of different components, then they are joined into a single connected component including the blocks of both components, and the added edge becomes a bridge for the new graph. Finally, if the edge joins two vertices of different blocks into the same component, several blocks of this component are joined into a single block but the components does not change. These are the blocks that are traversed by the shortest path joining the two extreme of the added edge. This path is obtained by a BFS.

The partial solution at each iteration of the MVCA is constituted by the set of labels already selected. Then the label that most reduces the number of blocks and components at that step is selected to be included in the partial solution. Note that the inclusion of an edge may reduce the number of blocks (by different amounts), the number of connected components (exactly by one), or neither of the two cases; but never both. The adding of a label instead may reduce the number of blocks, the number of connected components, both or neither of them, and in different amounts.

5 Summary and Outlook

We introduce the minimum labelling spanning block (MLSB) problem and adapt to it the exact solution method and MVCA used for the MLST problem. In the incoming future we plan to implement and compare other successful metaheuristics derived from the MLST literature. In order to consider real-world applications, we will extend the problem by considering that an edge can have associated more than one label, where each label may represent a different company in a transportation network perspective, or a different communication mode, frequency, or wavelength in telecommunication networks, and that each pair of nodes can be connected by one or more such multi-labeled connecting edges.

References

[1] R. S. Chang and S. J. Leu. The minimum labelling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.

[2] S. Consoli, K. Darby-Dowman, N. Mladenović, and J. A. Moreno-Pérez. Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*, 196(2):440–449, 2009.

[3] S. Consoli, N. Mladenović, and J. A. Moreno-Pérez. Solving the minimum labelling spanning tree problem by intelligent optimization. *Applied Soft Computing*, 28:440–452, 2015.

[4] S. Consoli and J. A. Moreno-Pérez. Variable neighbourhood search for the *k*-labelled spanning forest problem. *Electronic Notes in Discrete Mathematics*, 47:29–36, 2015.

[5] J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[6] S. O. Krumke and H. C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[7] R. Sedgewick and K. Wayne. *Algorithms (4th edition)*. Addison-Wesley, 2011.