GRASP for the Minimum Labelling *bi*-Connectivity of a labelled graph

José Andrés Moreno Pérez

Department of Computing and Systems Engineering
University of La Laguna, Tenerife, Spain
jamoreno AT ull.edu.es*
*Corresponding author

Sergio Consoli

ISTC/STLab, National Research Council (CNR), Catania, Italy
sergio.consoli AT istc.cnr.it

Abstract: A labelled or coloured undirected graph is a graph whose edges are labelled (or coloured). The Minimum Labelling Spanning Tree problem consists of, given a labelled undirected graph, to find the spanning tree with the minimum set of labels. This is a well combinatorial optimization problem that have been shown to be NP-hard. The problem can be formulated as to give the minimum number of labels that provide single connectivity among all the vertices of the graph. Here we consider the problem of finding the minimum set of labels that provide bi-connectivity among all the vertices of the graph. A graph is bi-connected if there are at least two disjoint paths joining every pair of vertices. We consider both bi-connectivity concept: the edge bi-connectivity where these paths cannot have a common edge and the vertex bi-connectivity where the paths cannot have a common vertex. We give an exact algorithm and a basic greedy constructive heuristic. Then a GRASP metaheuristic is given and their parameters anaylsed.

Keywords: xxxx

Reference to this paper should be made as follows: xxxx (xxxx) 'xxxx', *xxxx*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: AUTHOR PLEASE SUPPLY CAREER HISTORY OF NO MORE THAN 100 WORDS FOR EACH AUTHOR.

1 Introduction

The labelling optimization problems on network are defined on a labelled undirected or directed graph. Graph is labelled if each edge has a colour or label from a finite set of colours or labels; several edges can have the same label. The problems consist in selecting the optimal set of labels following some objective and subject to some constraints. Several labelling problems have appeared. The two most studied labelling problems are the labelling TSP ([35], [34], [48], [15], [31], [40], [32]) and the Labelling Spanning Trees ([3], [9],

[25], [44], [2], [7], [45], [46], [47])). Other labelling problems refer to paths [19], [4], bipartite subgraphs [29], matchings [30], [5], [31], Steiner trees [8], forests [4], and cuts [36], [37].

A best 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 or, equivalently, finding the minimum size set of labels that connects all the vertices of the graph [12]. Other labelling problems related to connection property has been proposed in the literature [36]. Several labeling problems have been derived from the MLST problem by relaxing the connectivity property. For instance, the kLSF problem consisting of finding the forest with minimum connected components using at most k different labels [14]. 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 disjoint paths. Since two paths can be considered that are disjoint if they have not a common vertex or a commom edge, there are to notions of bi-connectivity; the vertex bi-connectivity and the edge bi-connectivity. Therefore, an undirected graph is vertex 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. In the other hand, an undirected graph is edge bi-connected if any pair of vertices are joined by two edge-disjoint paths and, equivalently, if it remains connected after dropping any single edge. 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 or edge 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. The minimum labelling bi-connected problem consists of finding the minimum size set of labels that provides bi-connectivity. In a similar way we can extend the connectivity to the three-connectivity, the four-connectivity and so on.

The minimum labelling connectivity problems may be used to model many real-world situations where one aims to ensure some degree of connectivity among nodes by means of connections. For example, in telecommunication networks ([42]), there can be different types of communications media (such as optical fiber, coaxial cable, telephone line) or different companies to which the connections belong, or different transmission frequencies. It is then clear that we may be interested in optimizing this factor when we are considering the edges to be included in the solution of the problem that we are going to solve. In this context, a company would like to ensure the service to each terminal node by minimizing the cost (i.e., by minimizing the use of connections managed by other companies). However, in order to support node or link failure ...

This kind of problem, can be modeled as follows. The telecommunication network is represented by a graph where with each arc is assigned a label and each label denotes a different company that manages the arc. The aim of each company is to define a spanning tree of the graph that uses the minimum number of different labels. However, there could be situations where a budget constraint is involved, that is, there is a limit to the maximum number of different labels that can be used. In the other side may be interesting to optimizing the connectivity also in the presence of a limit in the number of transportation providers selected. This may reduce the construction cost and the overall complexity of the network (Hammadi and Ksouri, 2013). In these scenarios, a spanning tree would represent the optimum if we could find one that uses at most this limit number of labels. When such a spanning tree does not exist, then we are interested in finding a spanning forest, using at most k different labels, whose number of connected components is minimal (Cerulli et al.,

2014). In the other side, we can be interested, not only in connected networks, but also in networks that remains connected even in the event of a vertex or edge failure. Then we would consider the minimum labelling bi-connected problems.

Another example is given in multimodal transportation systems (Miller, 2006), where it is often required to guarantee a complete service between the terminal nodes of the network, without cycles, by using the minimum number of provider companies. Multimodal transportation networks can be represented by graphs where each edge is assigned a label, denoting a different company managing that link. In real-world situations we may be interested in

Besides applications in telecommunications network design ([41]) and data compression ([10]), such a model is very useful in multimodal transportation systems ([28]). A multimodal transportation network can be represented by a graph where each edge is assigned a label, denoting a different company managing that link. In this context, it is often desirable to provide a complete service between the nodes of the network, without cycles, by using the minimum number of companies ([43]). Thus, the aim is to find a spanning tree of the graph using the minimum number of labels. This spanning tree may reduce the construction cost and the overall complexity of the network. A practical example is given by multimodal transportation networks of large territories, from regions to states, or even continents, during humanitarian crisis events like, for example, volcanic eruptions, terrorist threats, floods, tsunamis, etc ([24]). In these very delicate crisis management situations, amongst different types of human intervention, it is also necessary to reorganize dynamically the entire transportation network of the damaged area, taking into account the upcoming inaccessible or forbidden zones, and guaranteeing a minimal working transport service among main cities, hospitals, airports, principal way outs, and others, with the minimum number of different transportation carriers and/or companies.

The rest of the paper is organized as follows. Section introduces the bi-connectivity concepts and the basic algorithms to test it. Then xxxx

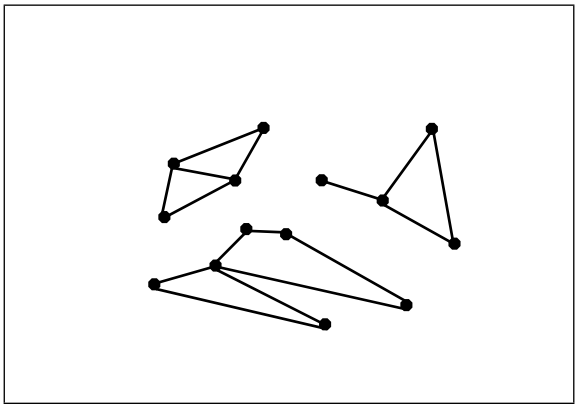
2 Bi-connectivity

An undirected graph $G = (V, E)$ is connected if and only if any pair of vertices are connected or joined by some path. The connected components of the graph are the maximal connected subgraphs; i.e. the connected subgraphs that are not included in another connected subgraph. The connected components of a graph constitute a partition of its set of vertices and of its set of edges. In the example of Figure 1 there is an undirected graph with 14 vertices, 16 edges and 3 connected components

A connected graph has only one connected component. A spanning tree is obtained from a connected graph by iteratively removing an edge from a cycle of the graph until no cycles exist. If the graph is not connected, the number of connected components gives a degree of the connectivity. If the number of connected components increases then the degree of connectivity decreases.

In the other hand, we can consider graphs with higher degree of connectivity. An undirected graph is (single) connected if and only if for any pair of vertices there is, at least, a path joining them. Then a graph is bi-connected if and only if there are, at least, two disjoint paths joining any pair of vertices. The idea of disjunction among paths can be taken by considering the paths as a sets of edges or also by considering the paths as a sets vertices. Then we say that graph G is edge bi-connected if and only if for any pair of vertices there

Figure 1 An undirected graph with 3 connected components



are, at least, two paths without a common edge. Analogously, we say that graph G is vertex bi-connected if and only if for any pair of vertices there are, at least, two paths without a common vertex, excluding the two joined vertices.

In the same way, for any integer k, a graph G is (edge/vertex) k-connected if for any pair of vertices there are at least k (edge/vertex) disjoint paths joining them. The (edge/vertex) connectivity of a connected graph is the maximum integer k such that the graph is (edge/vertex) k-connectivity. This concept of k-connectivity have been largely studied in graph theory [16] through standard graph theory books [20] up to very recent papers [22], [23], [33] related with the fault tolerance in sensor [26], [27], [49] and optical communication networks [1],[18].

The notion of k-connectivity can be extended for negative integers k if we consider that a graph with k connected components has connectivity $-k$. Then k-connectivity is extended to be an integer degree connectivity that ranges in the whole set of integer number except 0. Note that for negative integers, the edge and vertex connectivities are equivalent, and that the degrees of connectivity 1 and -1 are equivalent. In this paper we deal with the graphs with degree of connectivity equals 2 ($k = 2$), that is called the bi-connectivity.

The concept of articulation point or cut vertex is related with the vertex bi-connectivity, and the concept of bridge or cut edge is related with the edge bi-connectivity. A cut vertex or articulation point is a vertex whose removal disconnects the graph. A cut edge or bridge is an edge whose removal disconnects the graph. The vertex bi-connectivity is equivalent to the nonexistence of articulation point (cut vertex) and the edge bi-connectivity is equivalent to the nonexistence of bridge (cut edge). In the example of Figure 2 there are 7 articulation points represented by hollow vertices and the 4 bridges that are drawn as double lined edges.

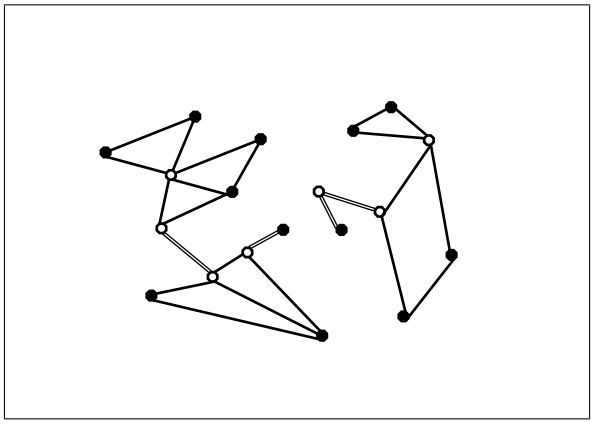
The connected components of an undirected graphs are its maximal connected subgraphs. SAnalogously, the maximal bi-connected subgraphs of a graph are its bi-connected components, usually called **blocks**. To differentiate between blocks corresponding to the edge bi-connectivity or to the vertex bi-connectivity we will call them edge-blocks or vertex-blocks; respectively. We have the following series of basic claims.

Claim 1: *An edge cannot belong to more than one edge-blocks.*

Claim 2: *A vertex cannot belong to more than one edge-blocks.*

[h]

Figure 2 Articulation points and Bridges

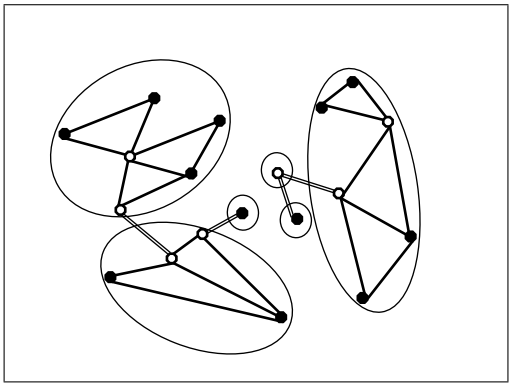


Claim 3: *The edge-blocks are disjoint as set of vertices.*

Claim 4: *The edge-blocks are disjoint as set of edges.*

In the example of Figure 3 there are 6 edge-blocks, each one of them is represented by an ellipse. Three of the blocks have a single vertex.

Figure 3 edge-blocks



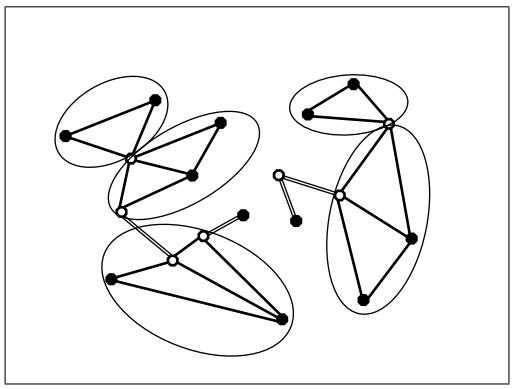
Claim 5: *The vertex-blocks are disjoint as set of edges.*

Claim 6: *A cut vertex belongs to more than one vertex-block.*

Claim 7: *The vertex-blocks are not necessarily disjoints in terms of sets of vertices.*

In the example of Figure 4 there is a graph with 8 vertex-blocks each one of them is represented by an ellipse. Three of these vertex-blocks have a single vertex.

Figure 4 Vertex Blocks



Claim 8: *The extremes of any bridge with degree greater than 1 is a cut vertex.*

Therefore, any vertex-block is included in an edge-blocks and every edge-block consists of one or more vertex blocks joined by cut vertices.

Claim 9: *Every edge that is not a bridge joints two vertices of the same vertex-block (and also of the edge-block).*

Claim 10: *Each bridge joints two vertices of different edge-blocks (and therefore of two different vertex-blocks).*

Therefore, every bridge is not in any vertex-block, neither in any edge-block.

Claim 11: *The set of vertex-blocks plus the set of bridges constitutes a partition of the whole set of edges.*

Then also the set of edge-blocks plus the set of bridges constitutes a partition of the whole set of edges.

Claim 12: *The set of edge-blocks also constitutes a partition of the set of vertices.*

However, the set of vertex-blocks does not constitute a partition of the set of vertices. Note that a cut vertex is in two different vertex blocks.

Therefore a vertex block corresponds to a set of edge and a edge-block to a set of vertices.

Claim 13: *A single vertex is a vertex-block if and only if it is an isolated vertex or it is the extreme of only edges that are bridges.*

Because of the above claim, edge-blocks constitutes a partition of the set of vertices. Then, we consider that any bridge is assumed to be a single vertex-block and we have, analogously, that vertex-blocks constitutes a partition of the set of vertices.

A clear view of the edge-blocks structure is easily obtained by painting each block with a colour (or enclosing each one in a big circle or ellipse) that can be joined by a single black edge (bridge) or not. The vertex-blocks structure is more difficult to visualize. When the size of the graph is low, a good view of the vertex-block structure is obtained by painting each block with a colour and then the cut vertices are drawn with big circle having several colours corresponding. When the size of the graph is high, we can shrink the blocks to represent them as big vertices of a new graph joined by bridges and articulation points as edges; however a vertex can be the articulation point of more than two blocks.

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 $l(e) \in \mathcal{L}$. Several labelling problems have been defined in the literature as referred above. Most of them are minimization problems consisting on determining the set of labels with the minimum possible size such that the corresponding graph has a given property. Given a subset of labels $L \subseteq \mathcal{L}$, the corresponding graph is $G(L) = (V, E(L))$ where $E(L) = \{e \in E | l(e) \in L\}$. Given a property P that can be verified or not for a subgraph of G, the corresponding labelling problem can be generally stated as follows. The goal of the problem is to find the set $L^* \subseteq \mathcal{L}$ that minimizes $|L|$, where $L \subseteq \mathcal{L}$ and $G(L)$ has property P.

The well-known minimum labelling spanning tree (MLST) problem introduced in [9] [25] is the labelling minimization problem corresponding to the standard connectivity property. The problem is originally formulated as consisting of finding the spanning tree T^* of the labelled graph with minimum number of different labels. To get the solution tree from the minimum size label set L^* of the (single) connectivity labelling problem, iteratively remove an edge from a cycle of the solution subgraph $G = (V, E(L^*))$ no cycles exist.

If the objective is to obtain the minimum set of labels with, at most, a given number k of connected components we obtain a series of minimization problems corresponding the property of having at most k connected components; i.e., at least connectivity degree $-k$. The goal is to find the set $L^* \subseteq \mathcal{L}$ that minimizes $|L|$, where $L \subseteq \mathcal{L}$ and $G(L)$ has connectivity degree at least k. The problem corresponding to $k = 1$ is the MLSTP. An equivalent series of dual problems have been considered in literature known as Minimum Labeling Forest problem denoted by k-LSF problem [6], [14], [13]. The goal of the k-LSF problem in is to find the set $L^* \subseteq \mathcal{L}$ that minimizes the number c of connected components of $G(L)$ where $L \subseteq \mathcal{L}$ and L has at most k labels.

This series of problems extend the MLSTP by considered a relaxed connectivities; i.e., negative degree of connectivity. We also consider two new series of problems by strengthening the degree of connectivity. We have a series of minimization labeling problems for the edge k-connectivity and another series for the vertex k-connectivity. The **Minimum Labeling k-Connectivity Problem** consists of selecting the smallest set of labels such that the corresponding graph is k-connected. If the we use the vertex-connectivity or the edge-connectivity we obtain the Minimum Labeling edge k-Connectivity Problem and the Minimum Labeling edge k-Connectivity Problem, respectively. Note that if, for fixed integer k, the given graph is not k-connected then the corresponding minimum labeling

k-connectivity problem is not feasible since, even including all the labels, the corresponding graph is not k-connected.

In this paper we we deal with the Minimum Labelling bi-Connectivity Problems corresponding to $k = 2$ for both, the vertex-connectivity and for the edge-connectivity. Since the bi-connected maximal subgraphs are called blocks, we consider 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. In this formulation we can consider the edge and the vertex concetivities.

4 Solution algorithms

We extend the basic MVCA heuristic [25] and A^* exact algorithm [11] for the MLST problem to the MLSB problem. From the greedy constructive algorithm we derive a GRASP procedure.

5 Greeedy Constructive Heusristic

The Maximum Vertex Covering Algorithm (MVCA) is one of the first heuristics for the MLST problem [9], [25] and it is used as basic component of most metaheuristics and exact methods applied to the problem. It is a greedy constructive algorithm that selects the labels using the number of connected components as greedy function to be minimized. We adapt this greedy constructive algorithm (GCA) for the Minimum Labelling Spanning Block (MLSB) problem; both for edge-blocks and for vertex-blocks.

The GCA for the MLSB problem starts with the empty set of labels. Therefore, in the initial graph each vertex is a vertex-block, an edge-block and a connected component, and there is not bridges. The partial solution at each iteration of the Greedy Constructive Algorithm is constituted by the set of labels already selected and the corresponding graph. Then the algorithm iteratively adds a label using the greedy strategy.

The greedy function to be minimized in the MVCA for the MLSTP is the number of connected components in the resulting graph, since the purpose is to reach the whole single connectivity. For the MLSB problem, we can use as the greedy function to be minimized, the number of blocks in the resulting graph. The number of edge-blocks for the Minimum Labelling Spanning Edge-Block (MLSEB) and The number of vertex-blocks for the Minimum Labelling Spanning Vertex-Block (MLSVB). However, since the initial graph has no edge and the first edges to be added joint isolated vertices and do not provide blocks. Even in the case that the number of blocks do not increase, the edges (and labels) that reduces the number of connected components contributes to be closer to the bi-connectivity. Therefore we use as greedy function (to be minimized) the number of connected components plus the number of blocks (edge-blocks vertex blocks) in the resulting graph. Note that, at the first steps, the number of connected components would be relevant but not at the final steps, when the graphs becomes connected, or almost connected.

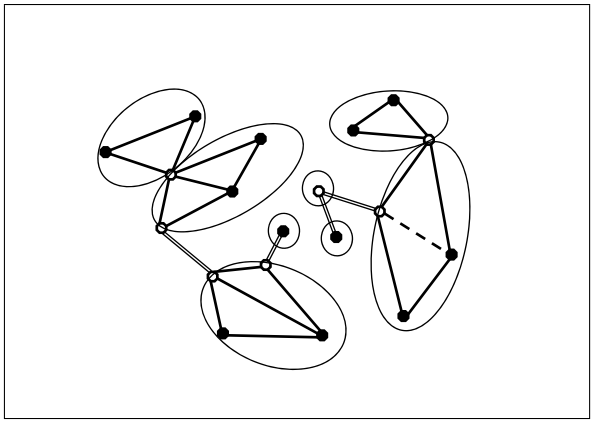
To know the number of edge-blocks and vertex-blocks when a label is added, like in the MVCA for the MLSTP, the corresponding edges are included one by one. The key

procedure is the updating of the set of bridges, the set of blocks (edge-blocks and vertex-blocks) and the set components when a single edge is added to the graph. Consequently, the resulting subgraph is updated by adding the edges associated to the selected label. The algorithm stops when a single block is obtained or when no label to be selected can reduce the number of blocks. Note that if the graph G is not bi-connected we cannot get a number of blocks smaller than the number of blocks of G adding any set of labels. However, if G is bi-connected then the output graph $G(L)$ is also bi-connected.

The ADD procedure get the new numbers of blocks when a single edge is added to the subgraph. Assume that we already know the bridges, connected components and blocks (edge-blocks or vertex-blocks) of the graph and we add a new edge. The new number of blocks of the resulting graph is computed as follows.

- First, if the edge joins two vertices of the same vertex-block, then the bridges, connected components, vertex-blocks and edge-blocks do not change. The new edge is added to this vertex blocks and also to the edge-blocks that includes it (see Figure 5).

Figure 5 New edge inside a vertex-block



- Second, if the edge joins two vertices of different connected components, then these two components are joined into a single connected component including all the blocks of both components, and the new edge becomes a bridge for the new graph (see Figure 6). Note that each (vertex or edge) block is in only one connected component, so every component has its own blocks.

- Finally, if the edge joins two vertices of different vertex-blocks into the same component, we have the most complicated case. In this case several blocks of this connected component are joined into a single block. The rest of the structure in the other connected components (bridges and blocks) do not change. The blocks joined are those traversed by the shortest path joining the two extreme of the added edge. This path is obtained by a simple BFS described later.

  Namely, the edge-blocks of the vertices traversed by this path are joined in a single edge-block (see Figure 7).

Figure 6 New edge joining two connected components

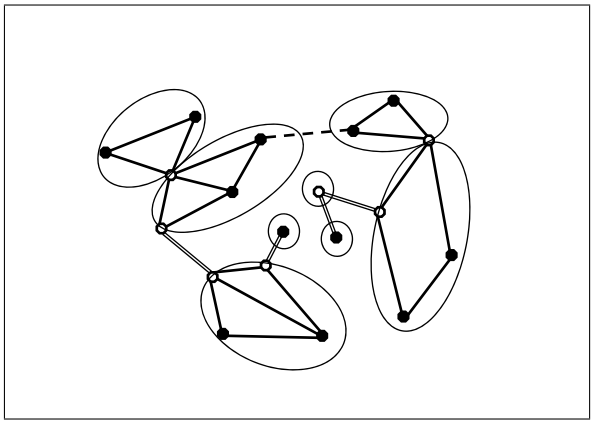
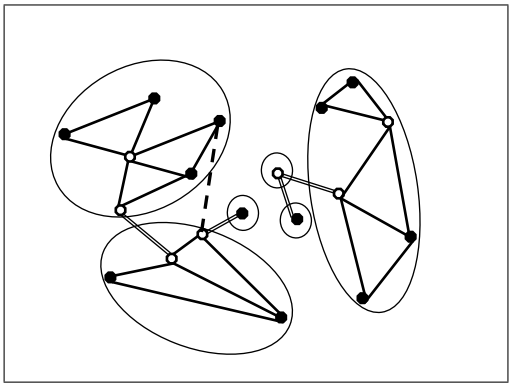


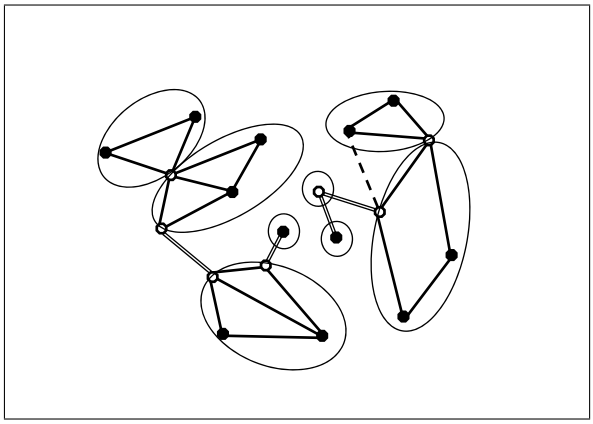
Figure 7 New edge joining two edge-blocks



Analogously, the vertex-blocks of the edges traversed by this path are joined in a single vertex-block. If the two extremes of the added edge are in the same edge-block (and therefore also the corresponding vertex-blocks are also included in this edge-block), this BFS path is inside this edge-block and then the number of edge-blocks does not change (see again Figure 7). However, if the two extremes are in different edge-blocks, the new vertex blocks is constituted by the joining of the bridges and vertex-blocks traversed by the BFS path (see Figure 8).

The bridges traversed by the BFS path become interior edges to the new block they have to be eliminated from the set of bridges. The cut-vertices traversed by this path become interior vertices to the new block they have to be eliminated from the set of cut-points. The number of edge-blocks and vertex-blocks decrease in the number of times that the BFS path changes from a block to another one in its trajectory. The number of edge-blocks and vertex-blocks decrease in the number of times that the BFS path changes from a block to another one in its trajectory. To compute it, we take into account the edge-blocks of the

Figure 8 New edge joining two vertex-blocks of the same edge-block



vertices traversed by the path and the vertex-blocks of the edges traversed by the path, for the number of edge-blocks or vertex-blocks respectively.

The updating procedure is describe in 1.

Input: A undirected subgraph $G = (V, E)$, with V as set of vertices, E as set of edges and a new edge $e = [i, j] \notin E$. The number of connected components Num_CComps, of edge-blocks $Num\_EBlocks$ and of vertex-blocks $Num\_VBlocks$ are known.
Output: The new subgraph $G = (V, E + \{[i, j]\})$, with the corresponding values for Num_CComps, $Num\_EBlocks$ and $Num\_VBlocks$.
begin
  if *i and j are in the same connected component*
  then
    - $Num\_CComps \leftarrow Num\_CComps - 1$,
    - $Num\_EBlocks \leftarrow Num\_EBlocks$ and
    - $Num\_VBlocks \leftarrow Num\_VBlocks$
  end
  else
    Apply the BFS to get the shortest path p from i to j.
    Let $Tr\_EBlocks$ be the number of different edge-blocks of the vertices in p.
    Let $Tr\_VBlocks$ be the number of different vertex-blocks of the edges in p.
    - $Num\_CComps \leftarrow Num\_CComps$,
    - $Num\_EBlocks \leftarrow Num\_EBlocks - Tr\_EBlocks + 1$ and
    - $Num\_VBlocks \leftarrow Num\_VBlocks - Tr\_VBlocks + 1$
  end
  Add edge $[i, j]$ to the subgraph.
end

Algorithm 1: Adding a new edge

Note that the inclusion of a single edge may reduce the number of edge-blocks and the number vertex-blocks by different amounts). It also can reduce the number of connected components but exactly by one. However, the adding of a single edge cannot reduce both, the number of blocks and the number of components. Note that this single increase of the set of edges can keep the number of blocks and number of components with the same value. The inclusion of an edge increases by one the number of bridges if it becomes a bridge and can reduced the number of bridges if this edge joins two blocks of the same component.

The number of bridges decreases exactly by the number of bridge included in the BFS path. However, the adding of a single label instead may reduce the number of blocks (both kind of blocks), the number of connected components, both or neither of them, and in different amounts. It can also increases or decreases the number of bridges in different amounts.

The GCA algorithm is describe in 2.

Input: A labelled undirected graph $G = (V, E)$, with V as set of vertices, E as set of edges, and labelling $\ell : E \to \mathcal{L}$
Output: A set of labels $L \subseteq calL$ such that $B(L) = B(\mathcal{L})$;
Notation:
- Let L denote the subset of used labels; i.e., $L \subseteq \mathcal{L}$;
- Let $G(L) = (V, E(L))$ be the subgraph of the edges with labels in L; i.e., where $E(L) = \{e \in E : \ell(e) \in L\}$;
- Let $C(L)$ be the number of connected components of $G(L)$;
- Let $B(L)$ be the number of blocks of $G(L)$;
Initialization:
- Let $L \leftarrow \emptyset$ be the initially empty set of used labels,
- Let $B(L) = C(L) = n$ where n is the number of vertices of G,
begin
  while $B(L) > B(\mathcal{L})$ **do**
    Select the unused label $l \notin L$ that minimizes $B(L \cup \{l\}) + C(L \cup \{l\})$;
    Add label l to the set of used labels: $L \leftarrow L \cup \{l\}$;
    Update $G(L) = (V, E(L))$ and $B(L)$;
  end
  Take $L^* = L$ as the optimal solution.
end
Greedy constructive algorithm

Algorithm 2: Greedy Constructive Algorithm for the MLSB problem

5.1 Exact method

The exact approach for the MLST problem [11] is adapted to solve the both MLSB problems. The procedure works in the space of sets of labels searching for the minimum size feasible solution. The method is based on an A^* or backtracking procedure to test the subsets of $\mathcal{L}$. 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 exact method is a subroutine that determines if the graph $G(L)$ has less blocks than $G(L^*)$. In the case that the whole graph is bi-connected, this procedure determines wether $G(L)$ is (edge/vertex) bi-connected or not. For example a Depth First Search (DFS) determine, in addition to the connected components, the bridges and the articulation vertices [39]. Given the bridges and articulation points we determine the edge-blocks and vertex-blocks. The number of sets tested, and therefore the running time, can be shortened by pruning the search tree using simple rules.

Graph traversal algorithms [39] 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 [21] to determine if a graph is bi-connected and, in the negative case, to obtain its blocks. Their DFS finds the cut vertices and bridges of the graph as follows. The algorithm traverse the vertices and edges of the graph using the depth-first strategy and determining a so-called directed DFS-tree. The algorithm uses the depth of every vertex and the lowest depth of neighbors of all its descendants in the DFS tree, its lowpoint. The root of the DFS tree is a cut vertex if it has more than one outgoing tree edges. 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. They are 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 this vertex was not visited before v by DFS, then v is a cut vertex. The lowpoint of v can be computed after visiting all descendants of v as the minimum of the depth of v, the depth of all neighbors of v and the lowpoint of all children of v in the depth-first-search tree. Moreover, DFS traverse the edges of each vertex block consecutively. Therefore, the vertex blocks are obtained using a stack to keep track of the edges being traversed. A bridge is a back edge that constitute a vertex-blocks. The edge-blocks are obtained by joining the blocks joined by cut vertices.

The exact algorithm proposed for the MLSBP is a $A*$ or backtracking procedure that tests the subsets of L. It performs a branch and prune procedure in the partial solution space based on a recursive procedure *Test* that attempts to find a better solution from the current incomplete solution. The main program that solves the MLSB problem calls the *Test* procedure with an empty set of labels. The details are specified in Algorithm 3. In order to reduce the number of tests, we select the Greedy Solution L^* obtained by the GCA in the initial step. The running time of the exact method based on A^* grows exponentially, but if either the problem size or the optimal number of labels is small, we could get the exact solution in a reasonable running time. It is valid for both edge and vertex connectivity.

5.2 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedure (GRASP) is a well known metaheuristic method that combine the power of greedy heuristics, randomisation, and local search [17]. It is a multi-start two-phase metaheuristic: consisting of a "construction phase" and a "local search phase". The construction phase builds a solution using a greedy randomized procedure, whose randomness allows solutions in different areas of the solution space to be obtained. Each solution is randomly produced step-by-step by uniformly adding one new element from a candidate list (RCL_α: restricted candidate list of length α) to the current

Input: A labelled undirected graph $G = (V, E)$, with V as set of vertices, E as set of edges, and labelling $\ell : E \to \mathcal{L}$
Output: set of labels $L \subseteq calL$ such that $B(L) = B(\mathcal{L})$;
Initialization:
- Let $L \leftarrow \emptyset$ be the initially empty set of used labels;
- Let $B(L) = n$ the initial number of blocks that is the number of vertices n;
- Let $L^* \leftarrow L$ be the current best set of used labels;
- Let $b_0 = B(\mathcal{L})$ the number of blocks of G;
begin
  Call $Test(L)$;
end

Procedure Test(L):
if $|L| < |L^*|$ **then**
  Update $B(L)$;
  if $B(L) = b_0$ **then**
    Move $L^* \leftarrow L$;
  else if $|L| < |L^*| - 1$ **then**
    foreach $l \notin L$ **do**
      Try to add label l: Test($L \cup \{l\}$);
    end
  end
end

Exact method for the MLSB problem

solution. Subsequently, the local search phase is applied to try to improve the current best solution. It is repeated by including it in a multistart framework until the stop condition is met. [38].

Input: A labelled undirected graph $G = (V, E)$, with V as set of vertices, E as set of edges, and labelling $\ell : E \to \mathcal{L}$
Output: set of labels $L \subseteq calL$ such that $B(L) = B(\mathcal{L})$;
Initialisation:
- Let $L \leftarrow \emptyset$ be the initially empty set of used labels;
- Let $B(L) = n$ the initial number of blocks that is the number of vertices n;
- Let $L^* \leftarrow \mathcal{L}$ be the current best set of used labels;
begin
  repeat
    Set $L \leftarrow \emptyset$;
    Construction phase(L);
    Local search(L);
    if $|L| < |L^*|$ **then**
      Move $L^* \leftarrow L$;
    end
  until *termination conditions*;
end

GRASP for the MLSB problem

Procedure Construction phase(L)***:***
Input: A labelled undirected graph $G = (V, E)$, with V as set of vertices, E as set of edges, and labelling $\ell : E \to \mathcal{L}$
Output: A set of labels $L \subseteq calL$;
while $B(L) > B(\mathcal{L})$ **do**
  Set $RCL \leftarrow \{l \notin L\}$ minimizing $B(L \cup \{l\})$;
  Select at random a label $l \in RCL$;
  Add label l to the set of used labels: $L \leftarrow L \cup \{l\}$;
  Update $G(L) = (V, E(L))$ and $B(L)$;
end
Procedure *Construction phase*$(\cdot)$

THESE DATA ARE NOT THE TRUE DATA

6 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. Andreassen B. Helvik. Fault tolerance in optical networks; a study of electronic in-and egress interconnections in torus topologies. In *Proc. 9th Conf. Opt. Network Design Model*, 2005.

[2] T. Brüggemann, J. Monnot, and G. J. Woeginger. Local search for the minimum label spanning tree problem with bounded colour classes. *Operations Research Letters*, 31:195–201, 2003.

[3] H. Broersma and X. Li.

[4] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 345–353, 2000.

[5] Francesco Carrabs, Raffaele Cerulli, and Monica Gentili. The labeled maximum matching problem. *Computers & Operations Research*, 36(6):1859 – 1871, 2009.

[6] R. Cerulli, A. Fink, M. Gentili, and A. Raiconi. The k-labeled spanning forest problem. *Procedia - Social and Behavioral Sciences*, 108:153–163, 2014.

[7] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. In B. L. Golden, S. Raghavan, and E. A. Wasil, editors, *The Next Wave on Computing, Optimization, and Decision Technologies*, pages 93–106. Springer-Verlag, New York, 2005.

[8] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Extensions of the minimum labelling spanning tree problem. *Journal of Telecommunications and Information Technology*, 4:39–45, 2006.

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

[10] A. M. Chwatal, G. R. Raidl, and O. Dietzel. Compressing fingerprint templates by solving an extended minimum label spanning tree problem. *Proceedings of the 7th Metaheuristics International Conference (MIC 2007)*, 2007. Montreal, Canada.

[11] 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.

[12] 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.

[13] S. Consoli, N. Mladenović, and J. A. Moreno-Pérez. Comparison of metaheuristics for the k-labelled spanning forest problem. *International Transactions in Operational Research*, page to appear, 2016.

[14] 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.

[15] B. Couëtouxa, L. Gourvès, J. Monnota, and O.A. Telelis. Labeled traveling salesman problems: Complexity and approximation. volume 7, pages 74–85. 2010.

Table 1 Computational results, (*max-CPU-time* for heuristics = 1000 ms)

Parameters			**Average objective function values**		
n	ℓ	d	EXACT	CGA	GRASP
20	20	0.8	*2.4*	2.4	2.4
		0.5	*3.1*	3.2	3.1
		0.2	*6.7*	6.7	6.7
30	30	0.8	*2.8*	2.8	2.8
		0.5	*3.7*	3.7	3.7
		0.2	*7.4*	7.4	7.4
40	40	0.8	*2.9*	2.9	2.9
		0.5	*3.7*	3.7	3.7
		0.2	*7.4*	7.6	7.4
50	50	0.8	*3*	3	3
		0.5	*4*	4	4.1
		0.2	*8.6*	8.6	8.6
TOTAL:			*55.7*	56	55.8

Parameters			**Computational times (milliseconds)**		
n	ℓ	d	EXACT	MGA	GRASP
20	20	0.8	*0*	0	15.6
		0.5	*0*	1.6	22
		0.2	*11*	3.1	23.4
30	30	0.8	*0*	3	9.4
		0.5	*0*	3.1	26.5
		0.2	*138*	4.7	45.4
40	40	0.8	*2*	6.3	12.5
		0.5	*3.2*	7.9	28.2
		0.2	$100.2 \cdot 10^3$	10.8	120.3
50	50	0.8	*3.1*	17.1	21.8
		0.5	*21.9*	20.2	531.3
		0.2	$66.3 \cdot 10^3$	17.2	93.6
TOTAL:			$166.7 \cdot 10^3$	95	950

[16] P. Erdös and A. Rényi. On the strength of connectedness of random graphs. *Acta Mathematica Hungarica*, 12(1):261–267, 1961.

[17] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[18] S. Sundara Vadivelu G. Ramesh. A reliable and fault-tolerant routing for optical wdm networks. *International Journal of Computer Science and Information Security*, 6(2):48 – 54, 2009.

[19] G.J. Woeginger S. Zhang H. Broersma, X. Li. Paths and cycles in colored graphs. *Australasian Journal on Combinatorics*, 31, 2005.

[20] F. Harary. *Graph Theory*. Addison-Wesley, 1994.

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

[22] V. Gligor J. Zhao, O. Yağan. k-connectivity in random key graphs with unreliable links. *IEEE Transactions on Information Theory*, 61(7):3810 – 3836, 2015.

[23] V. Gligor J. Zhao, O. Yağan. k-connectivity of random key graphs. *arXiv:1502.00400 [physics.soc-ph]*, 12(1):261–267, 2015.

[24] J. Koźlak. Learning in cooperating agents environment as a method of solving transport problems and limiting the effects of crisis situations. In Y. Shi, G. D. Van Albada, J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the 7th International Conference on Computational Science (ICCS 2007), LNCS*, volume 4488, pages 872–879. Springer-Verlag, Heidelberg, Germany, 2007.

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

[26] Muriel Médard and Steven S. Lumetta. *Network Reliability and Fault Tolerance*. John Wiley & Sons, Inc., 2003.

[27] Deepankar Medhi. *Network Reliability and Fault-Tolerance*. John Wiley & Sons, Inc., 2007.

[28] J. S. Miller. *Multimodal statewide transportation planning: A survey of state practices*. Transportation Research Board, National Research Council, Virginia, US, 2006.

[29] J. Monnot. The labeled perfect matching in bipartite graphs. *Information Processing Letters*, 96(3):81–88, 2005.

[30] J. Monnot. A note on the hardness results for the labeled perfect matching problems in bipartite graphs. *RAIRO-Operations Research*, 42(3):315–324, 2008.

[31] I. Kanj M.R. Fellows, J. Guo. The parameterized complexity of some minimum label problems. *Journal of Computer and System Sciences*, 76.

[32] G. Laporte N. Jozefowiez and F. Semet. A branch-and-cut algorithm for the minimum labeling hamiltonian cycle problem and two variants. *Computers and Operations Research*, 38:1534–1542, 2011.

[33] D. Poole. On the strength of connectedness of a random hypergraph. *The Electronic Journal of Combinatorics*, 22(1):Paper #P1.69, 2015.

[34] A.P. Punnen. Erratum: Traveling salesman problem under categorization. *Operations Research Letters*, 14.

[35] A.P. Punnen. Traveling salesman problem under categorization. *Operations Research Letters*, 12.

[36] D. Segev R. Hassin, J. Monnot. Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14(4):437–453, 2007.

[37] D. Segev R. Hassin, J. Monnot. The complexity of bottleneck labeled graph problems. In *in: Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 4769 of *Lecture Notes in Computer Science*, pages 328–340, 2007.

[38] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedure. In F. Glover and G. Kochenberger, editors, *Handbook of metaheuristics*, pages 219–249. Kluwer Academic Publishers, Norwell, MA, 2003.

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

[40] J. Silberholz, A. Raiconi, R. Cerulli, M. Gentili, B. Golden, and S. Chen. Comparison of heuristics for the colourful travelling salesman problem. *International Journal of Metaheuristics*, 2(2):141–173, 2013.

[41] A. S. Tanenbaum. *Computer networks*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[42] A. S. Tanenbaum and D. Wetherall. *Computer Networks: Pearson New International Edition*. Pearson Education, United Kingdom, 2013.

[43] R. Van-Nes. *Design of multimodal transport networks: A hierarchical approach*. Delft University Press, The Netherlands, 2002.

[44] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84:99–101, 2002.

[45] Golden B. Xiong, Y. and E. Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 9(1):55–60, 2005.

[46] Golden B. Xiong, Y. and E. Wasil. Worst-case behavior of the mvca heuristic for the minimum labeling spanning tree problem. *Operations Research Letters*, 33(1):77–80, 2005.

[47] Golden B. Xiong, Y. and E. Wasil. Improved heuristics for the minimum labelling spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 10(6):700–703, 2006.

[48] Y. Xiong, B. Golden, and E. Wasil. The colorful traveling salesman problem. In E. K. Baker A. Joseph, A. Mehrotra, and M. A. Trick, editors, *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, volume 37 of *Operations Research/Computer Science Interfaces Series*, pages 115–123. Springer-Verlag, 2007.

[49] Ting Yuan and Shiyong Zhang. Secure fault tolerance in wireless sensor networks. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*, pages 477–482, July 2008.