A Constraint-based Local Search for Edge Disjoint Rooted Distance-Constrained Minimum Spanning Tree Problem

Alejandro Arbelaez, Deepak Mehta, Barry O'Sullivan, and Luis Quesada

Insight Centre for Data Analytics, University College Cork, Ireland
{alejandro.arbelaez,deepak.mehta,barry.osullivan,luis.quesada}@insight-centre.org

Abstract. Many network design problems arising in areas as diverse as VLSI circuit design, QoS routing, traffic engineering, and computational sustainability require clients to be connected to a facility under path-length constraints and budget limits. These problems can be modelled as Rooted Distance-Constrained Minimum Spanning-Tree Problem (RDCMST), which is NP-hard. An inherent feature of these networks is that they are vulnerable to a failure. Therefore, it is often important to ensure that all clients are connected to two or more facilities via edge-disjoint paths. We call this problem the Edge-disjoint RDCMST (ERDCMST). Previous works on RDCMST have focused on dedicated algorithms which are hard to extend with side constraints, and therefore these algorithms cannot be extended for solving ERDCMST. We present a constraint-based local search algorithm for which we present two efficient local move operators and an incremental way of maintaining objective function. Our local search algorithm can easily be extended and it is able to solve both problems. The effectiveness of our approach is demonstrated by experimenting with a set of problem instances taken from real-world passive optical network deployments in Ireland, the UK, and Italy. We compare our approach with existing exact and heuristic approaches. Results show that our approach is superior to both of the latter in terms of scalability and its anytime behaviour.

1 Introduction

Many network design problems arising in areas as diverse as VLSI circuit design, QoS routing, traffic engineering, and computational sustainability require clients to be connected to a facility under path-length constraints and budget limits. Here the length of the path can be interpreted as distance, delay, signal loss, etc. For example, in a multicast communication setting where a single node is broadcasting to a set of clients, it is important to restrict the path delays from the server to each client. In Long-Reach Passive Optical Networks (LR-PON) a metro-node is connected to the set of exchange-sites via optical fibres, the length of the fibre between an exchange-site and its metro-node is bounded due to signal loss, and the goal is to minimise the cost resulting from the total length of fibres [1]. In VLSI circuit design path delay is a function of maximum

interconnection path length while power consumption is a function of the total interconnection length [2]. In package shipment service guarantee constraints are expressed as restrictions on total travel time from an origin to a destination, and the organisation wants to minimise the transportation costs [3]. In wildlife conservation, which is an application from computational sustainability [4], the landscape connectivity is key to resilient wildlife populations in an increasingly fragmented habitat matrix where landscape connectivity is a function of the length of the path in terms of landscape resistance to animal movement.

Many of these network design problems can be modelled as Rooted Distance-Constrained Minimum Spanning-Tree Problem (RDCMST) [2] which is NP-hard. The objective is to find a minimum cost spanning tree with the additional constraint that the length of the path from a specified root-node (or facility) to any other node (client) must not exceed a given threshold. Many networks are complex systems that are vulnerable to a failure. A major fault occurrence would be a complete failure of the facility which would affect all the clients connected to the facility. Therefore it is important to provide network resilience.

In this paper we focus on the networks where all clients are required to be connected to two facilities via two edge-disjoint paths so that whenever a single facility fails or a single link fails all clients are still connected to at least one facility. We define this problem as the Edge-disjoint Rooted Distance-Constrained Minimum Spanning-Trees Problem (ERDCMST). Given a set of facilities and a set of clients such that each client is associated with two facilities, the problem is to find a set of distance-constrained spanning trees rooted from each facility with minimum total cost. Additionally, each client is connected to its two facilities via two edge-disjoint paths. Notice that if a same edge appears in two trees then it means there exists a client which is connected to its two facilities using the same edge. This would effectively mean that each pair of distance-bounded spanning trees would not be mutually disjoint in terms of edges. Certainly, ERDCMST is more complex than RDCMST as the former not only involves finding a set of distance-constrained spanning trees but also enforces that an edge between any pair of clients can only be used in at most one tree.

Previous works on RDCMST [5, 6] have focused on dedicated algorithms which are hard to extend with side constraints, and therefore these algorithms cannot be extended for solving ERDCMST. We present a constraint-based local search algorithm which can easily be extended to apply widely. We present two efficient local move operators, an incremental way of maintaining information required to checking constraints efficiently and an incremental way of maintaining cost of the assignment which are key elements for efficient local search algorithms. Our local search algorithm is able to solve both RDCMST and ERDCMST problems. The effectiveness of our approach is demonstrated by experimenting with a set of problem instances taken from real-world passive optical network deployments in Ireland, the UK, and Italy. We compare our approach with existing exact and heuristic approaches. Our results show that our approach is superior to both of the latter in terms of scalability and its anytime behaviour.

2 Formal Specification and Complexity

Let M be the set of facilities. Let E be the set of clients. Let $E_i \subseteq E$ be the set of clients that are associated with facility $m_i \in M$. We use N to denote the set of nodes, which is equal to $M \cup E$. We use T_i to denote the tree network associated with facility i. We also use $N_i \subseteq N = E_i \cup \{m_i\}$ to denote the set of nodes in T_i. Let λ be the maximum path-length from a facility to any of its clients.

Rooted Distance-Constrained Minimum Spanning-Tree Problem (RDCMST) Given a facility $m_i \in M$, the set of clients E_i, a set of feasible links $L_i \subseteq N_i^2$, two real numbers, a cost c_l and a distance d_l for each link $l \in L_i$, and a real number λ, the RDCMST is to find a spanning tree T_i with minimum total cost such that the length of the path from the facility m_i to any $e_j \in E_i$ is not greater than λ.

Edge-Disjoint Rooted Distance-Constrained Minimum Spanning-Trees Problem (ERDCMST). Given a set of facilities M, a set of clients E, a set of feasible links $L \subseteq N^2$, two real numbers, a cost c_l and a distance d_l for each link $l \in L$, an association of clients with two facilities $\pi : E \rightarrow M^2$, and a real number λ, the ERDCMST is to find a spanning tree T_i for each facility m_i such that:

1. The length of the unique path from the facility m_i to any of its clients is not greater than λ.
2. For each client e_k, the two paths connecting e_k to m_i and m_j, where $\pi(e_k) = \langle m_i, m_j \rangle$, are edge disjoint.
3. The sum of the costs of the edges in all the spanning trees is minimum.

Figure 1 shows an example with two facilities F_1 and F_2 and N={a, b, c, d, e, f}, black (reps. gray) edge denote the set of edges used to reach F_1 (reap. F_2), the minimum valid value for λ is 12 and the total cost of the solution is 46 for this illustrative example. The indicated solution satisfies the length constraint (i.e., the distance from the facilities to any node is less or equal to λ=12), and the paths connecting the set of nodes to the facilities are edge disjoint. For instance a solution with lower cost would be replacing gray edge $\langle F_2, a \rangle$ and $\langle F_2, b \rangle$ for a gray edge $\langle a, b \rangle$, but this solution would not be edge disjoint, because edges $\langle a, b \rangle$ and $\langle b, c \rangle$ would be used for nodes b and c to reach the two facilities.

Complexity. ERDCMST involves finding a rooted distance-bounded spanning tree for every facility whose total cost is minimum. This problem is known to be NP-complete [2].

3 Constraint Optimisation Formulation

In this section we present a constraint optimisation formulation of ERDCMST. Without loss of generality, in the formulation of the problem we assume full connectivity in the graph, and non-existing links can be added in L by associating

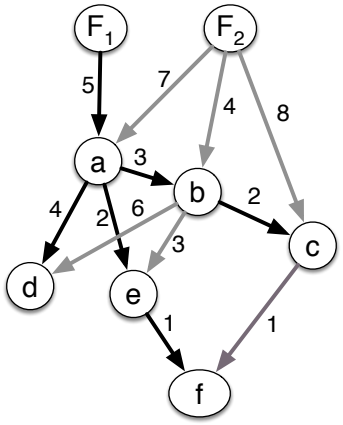


Fig. 1. Example of an instance of the ERDCMST problem. λ=12

them with very large distances (w.r.t. to λ). The model only relies in integer variables and all constraints are linear, and therefore it can be encoded directly into existing MIP solvers.

Variables.

- Let x^i_{jk} be a Boolean variable that denotes whether an arc between clients $j \in E_i$ and $k \in E_i$ of facility $i \in M$ is selected or not. Each arc (i, j) has an associated cost c_{ij}[1].
- Let f^i_j be a variable that denotes the length of the path from the facility i to its client j.
- Let b^i_j be a variable that denotes the maximum length of the path from the client j to any client in the tree of facility i that is acting as a leaf-node.

We remark that the partial order enforced by f or b help to rule out cycles in the solution.

Constraints. Each client associated with each facility has only one incoming arc:

$$\forall_{m_i \in M} \forall_{e_k \in E_i} : \quad \sum_{e_j \in N_i} x^i_{jk} = 1$$

Each facility is connected to at least one of its clients:

$$\forall_{m_i \in M} : \quad \sum_{e_j \in E_i} x^i_{ij} \geq 1$$

The total number of arcs in any tree T_i is equal to $|E_i|$:

$$\forall_{m_i \in M} : \quad \sum_{e_j \in N_i} \sum_{e_k \in E_i, e_j \neq e_k} x^i_{jk} = |E_i|$$

[1] In this paper we assume that the cost is symmetrical, i.e., the c_{ij}=c_{ji}.

If there is an arc from $e_j \in E_i$ to $e_k \in E_i$ then the length of the path from m_i to e_k is r equal to the sum of the lengths from m_i to e_j plus the length between e_j and e_k:

$$\forall_{m_i \in M} \forall_{\{e_j, e_k\} \in N_i} : \quad x^i_{jk} = 1 \Rightarrow f^i_k = f^i_j + d_{jk}$$

If there is an arc from $e_j \in E_i$ to $e_k \in E_i$ then the length of the path from e_j to any leaf-node through e_k is greater than or equal to the sum of the lengths from e_k to any of its leaf-node plus the length between e_j and e_k:

$$\forall_{m_i \in M} \forall_{\{e_j, e_k\} \in N_i} : \quad x^i_{jk} = 1 \Rightarrow b^i_j \geq b^i_k + d_{jk}$$

At any node in the tree the length of the path from a facility m_i to a client e_j and the length of the path from e_j to the farthest client on the same path should be less than λ:

$$\forall_{m_i \in M} \forall_{e_j \in N_i} : \quad f^i_j + b^i_j \leq \lambda$$

If m_i and $m_{i'}$ are the facilities of the client j, and if there exists any path in the subnetwork associated with facility i that includes the arc $\langle e_j, e_k \rangle$, then facility i' cannot use the same arc. Therefore, we enforce the following constraint:

$$\forall_{\{m_i, m_{i'}\} \in M} \forall_{\{e_j, e_k\} \in E_i \cap E_{i'}} : x^i_{jk} + x^{i'}_{jk} \leq 1$$

Objective. The objective is to minimize the total cost:

$$\min \sum_{m_i \in M} \sum_{\{e_j, e_k\} \in E_i} c_{jk} \cdot x^i_{jk}$$

4 Iterated Constraint-based Local Search

The Iterated Constraint-based Local Search (ICBLS) [7, 8] framework depicted in Algorithm 1 comprises two phases. First, in a local search phase, the algorithm improves the current solution, little by little, by performing small changes. Generally speaking, it employs a move operator in order to move from one solution to another in the hope of improving the value of the objective function. Second, in the perturbation phase, the algorithm perturbs the incumbent solution (s^*) in order to escape from difficult regions of the search (e.g., a local minima). Finally, the acceptance criterion decides whether to update s^* or not. To this end, with a probability 5% s'^* will be chosen, and the better one otherwise.

Our algorithm starts with a given initial solution where all clients are able to reach their facilities while satisfying all constraints (i.e., the upper bound in the length and disjointness). We use the trivial solution of connecting all clients directly to their respective facilities. We switch from the local search phase to perturbation when a local minima is observed; in the perturbation phase we perform a given number of random moves (20 in this paper). The stopping criteria is either a timeout or a given number of iterations.

Algorithm 1 Iterated Constraint-Based Local Search

1: s_0 := Initial Solution
2: s^* := ConstraintBasedLocalSearch(s_0)
3: **repeat**
4: s' := Perturbation(s^*)
5: s'^* := ConstraintBasedLocalSearch(s')
6: s^* := AcceptanceCriterion(s^*, s'^*)
7: **until** No stopping criterion is met

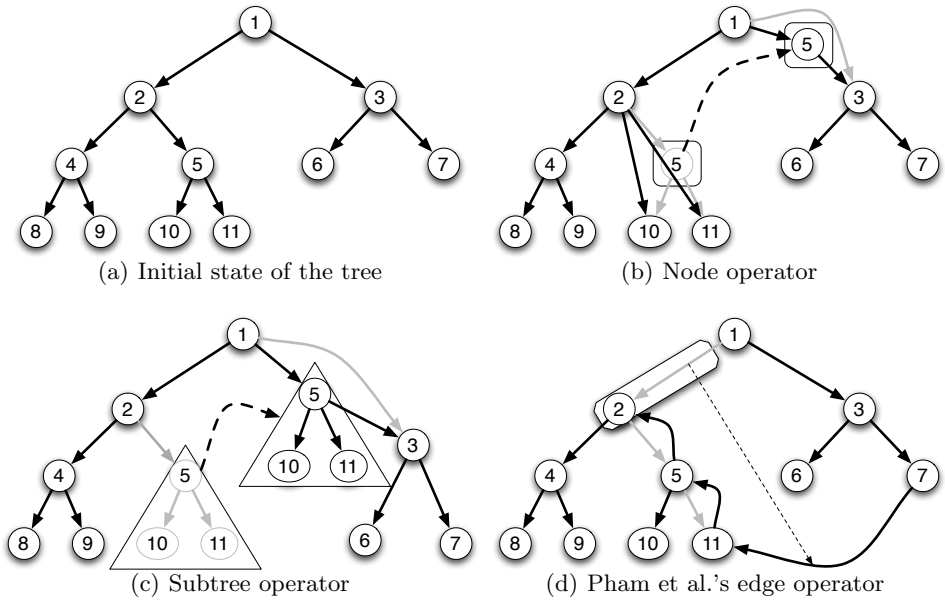


Fig. 2. Move operators

4.1 Move-Operators

In this section we propose *node* and *subtree* move operators. We use T_i to denote the tree associated with facility i. An edge between two clients e_p and e_q is denoted by $\langle e_p, e_q \rangle$.

Node operator (Figure 2(b)) moves a given node e_i from the current location to another in the tree. As a result of this, all successors of e_i will be directly connected to the predecessor node of e_i. e_i can be placed as a new successor for another node or in the middle of an existing arc in the tree.

Subtree operator (Figure 2(c)) moves a given node e_i and the subtree emanating from e_i from the current location to another in the tree. As a result of this, the predecessor of e_i is not connected to e_i, and all successors of e_i are still directly

connected to e_i. e_i can be placed as a new successor for another node or in the middle of an existing arc.

Edge Operator (Figure2(d)). In this paper we limit our attention to moving a node or a complete subtree. [9] proposed to move edges in the context of the Constrained Optimum Path problems. Pham et al. move operator (Figure 2(d)) chooses an edge in the tree and finds another location for it without breaking the flow.

4.2 Operations and Complexities

We first present the complexities of node and subtree operators as they share similar features. For an efficient implementation of the move operators, it is necessary to maintain f_j^i (the length of the path from facility i to client j) and b_j^i (length of the path from e_j down to the farthest leaf associated with it in tree T_i) for each client e_i associated with each facility m_i. This information will be used to maintain the path-length constraint. Let e_{p_j} be the immediate predecessor of e_j and let S_j be the set of immediate successors of e_j in T_i. Table 1 summarises the complexities of the move operators. In this table n denotes the maximum number of clients associated with a single facility.

Table 1. Complexities of different operations

	Node	Subtree	Arc
Delete	$O(n)$	$O(n)$	$O(1)$
Feasibility	$O(1)$	$O(1)$	$O(n)$
Move	$O(n)$	$O(n)$	$O(n)$
Best move	$O(n)$	$O(n)$	$O(n^3)$

Delete. Deleting a node e_j from T_i requires a linear complexity with respect to the number of clients of facility m_i. For both operators, it is necessary to update $b_{j'}^i$ for all the nodes j' in the path from the facility m_i to client e_{p_j} in T_i. In addition, the node operator updates $f_{j'}^i$ for all nodes j' in the subtree emanating from e_i. After deleting a node e_j or a subtree emanating from e_j, the objective function is updated as follows:

$$obj = obj - c_{j,p_j}$$

Furthermore, the node operator needs to add to the objective function the cost of disconnecting each successor element of e_j and reconnecting them to e_{p_j}.

$$obj = obj + \sum_{k \in S_j} (c_{k,p_j} - c_{kj})$$

Feasibility. Checking feasibility for a move can be performed in constant time by using f_j^i and b_j^i. If e_j is inserted between an arc $\langle e_p, e_q \rangle$ then we check the following:

$$f_p^i + c_{pj} + c_{jq} + b_q^i < \lambda$$

If e_p is a leaf-node in the tree and e_j is placed as its successor then the following is checked:

$$f_p^i + c_{pj} + b_j^i < \lambda$$

Move. A move can be performed in linear time. We recall that this move operator might replace an existing arc $\langle e_p, e_q \rangle$ with two new arcs $\langle e_p, e_j \rangle$ and $\langle e_j, e_q \rangle$. This operation requires to update f_j^i for all nodes in the emanating tree of e_j, and b_j^i in all nodes in the path from the facility acting as a root node down to the new location of e_j. The objective function must be updated as follows:

$$obj = obj + c_{pj} + c_{jq} - c_{pq}$$

Best Move. Selecting the best move involves traversing all clients associated with the facility and selecting the one with the maximum reduction in the objective function.

Now we switch our attention to the edge operator. This operator does not benefit by using b_j^i. The reason is that moving a given edge from one location to another might require changing the direction of a certain number of edges in the tree as shown in Figure 2(d). Deleting an edge requires constant complexity, this operation generates two separated subtrees and no data structures need to be updated. Checking the feasibility of adding an edge $\langle e_{p'}, e_{q'} \rangle$ to connect two subtrees requires linear complexity. It is necessary to actually traverse the new tree to obtain the distance from $e_{q'}$ to the farthest leaf in tree of facility i. Performing a move requires a linear complexity, and it involves updating f_j^i for the new emanating tree of $e_{q'}$. And performing the best move requires a cubic time complexity, the number of possible moves is n^2 (total number of possible edges for connecting the two subtrees) and for each possible move it is necessary to check feasibility. Due to the high complexity ($O(n^3)$) of the edge operator to complete a move, hereafter we limit our attention to the node and subtree operators.

Disjointness. To ensure disjointness among spanning-trees we maintain a $2 \cdot |E|$ Matrix, where $|E|$ represents the number of clients. For every client, there are two integers indicating the predecessor in the primary and secondary facilities, these two numbers must always be different.

We also maintain a graph which we call *facility connectivity graph*, denoted by fcg, where the vertices represent the trees associated with the facilities and an edge between a pair of trees represent that a change in one tree can affect the change in another tree. An edge between two vertices in fcg is added if the facilities share at least 2 clients. Notice that if two facilities do not share at least 2 clients then they are independent from disjointness point of view.

[6] also proposed two move operators for the RDCMST. *Edge-Replace* is a special case of our subtree operator, which removes an arc in the tree, and finds the cheapest arc to reconnect the two sub-trees. *Component-Renew* removes an arc in the tree, nodes that are separated from the root node are sequentially added in the tree using Prim's algorithm, nodes that violate the length constraint are added in the tree using a pre-computed route from the root node to any node in the tree. It is worth noticing that the *Component-Renew* operator cannot be applied to DRDCMST as the pre-computed route from the root node to a give node might not be available due to the disjoint constraint.

4.3 Algorithm

Algorithm 2 ConstraintBasedLocalSearch (*move-op*,*sol*)

1: $\{T_1 \dots T_n\} \leftarrow sol$
2: $list \leftarrow \{(m_i, e_j) | m_i \in M \wedge e_j \in E_i\}$
3: $fcg \leftarrow \{(m_i, m_j) || N_i \cap N_j| \geq 2\}$
4: **while** $list \neq \emptyset$ **do**
5: Select (m_i, e_j) randomly from $list$
6: Delete e_j from T_i and update T_i
7: $Best \leftarrow \{(e_{p_j}, S_j)\}$
8: $cost \leftarrow \infty$
9: **for** (e_q, S) in $Locations(move\text{-}op, T_i)$ **do**
10: **if** FeasibleMove(*move-op*, $(e_q, S), e_j$) **then**
11: $cost' \leftarrow$ CostMove((e_q, S), e_i)
12: **if** $cost' < cost$ **then**
13: $Best \leftarrow \{(e_q, S)\}$
14: $cost \leftarrow cost'$
15: **else if** $cost' = cost$ **then**
16: $Best \leftarrow Best \cup \{(e_q, S)\}$
17: **end if**
18: **end if**
19: **end for**
20: Select $(e_{q'}, S')$ randomly from $Best$
21: **if** $e_{q'} \neq e_{p_j} \vee S' \neq S_j$ **then**
22: $list \leftarrow list \cup \{(m_k, e) | (m_i, m_k) \in fcg \wedge e \in N_k \cap (S_j \cup \{e_{p_j}\})\}$
23: **end if**
24: $list \leftarrow list - \{(m_i, e_j)\}$
25: $T_i \leftarrow$Move(T_i,*move-op*, $(e_{q'}, S')$, e_j)
26: **end while**
27: return $\{T_1 \dots T_n\}$

The pseudo-code for constraint-based local search is depicted in Algorithm 2. It starts by selecting and removing a client e_j of a facility m_i randomly from Tree T_i (Lines 5 and 6) and performs a move by using one of the move operators as described before. Here the move-operator, which is itself a function, is passed

as a parameter. In each iteration of the algorithm (Lines 9-19), we identify the best location for e_j. A location in a tree T_i is defined by (e_q, S) where e_q denotes the parent of e_j and S denotes the set of successors of e_j after the move is performed. Broadly speaking, there are two options for the new location: (1) Breaking an arc $\langle e_p, e_q \rangle$ and inserting e_j in between them such that the parent node of e_j would be e_p and the successor set would be singleton, i.e., $S = \{e_q\}$; (2) Adding a new arc $\langle e_p, e_j \rangle$ in the tree in which case the parent of e_j is e_p and $S = \emptyset$. *Locations* returns all the locations relevant w.r.t. a given move-operator (Line 9). Line 10 verifies that the new move is not breaking any constraint and *CostMove* returns the cost of applying such move using a given move operator. (Line 11).

If the cost of new location is the best so far then the best set of candidates is reiniliased to that location (Lines 12-14). If the cost is same as the best known cost so far then the best set of candidates is updated by adding that location (Lines 15-16). The new location for a given node is randomly selected from the best candidates if there is more than one (Lines 20).

Instead of verifying that the local minima is reached by exhaustively checking all moves for all clients of all facilities, we maintain a list of pairs of clients and facilities. The list is initialised in Line 2 with all pairs of facilities and clients. In each iteration a pair of facility and client, (m_i, e_j) is selected and removed from the list (Line 24). When *list* is empty the algorithm reaches a local minima. If an improvement is observed then one simple way to ensure the correctness of local minima is to populate *list* with all pairs of facilities and clients. However, this can be very expensive in terms of time. We instead exploit the *facility connectivity graph* (*fcg*) where an edge between two facilities is added (in Line 3) if they share at least two clients. If an edge between m_i and m_k exists in *fcg* then it means that if there is a change in T_i then it might be possible to make another change in T_k such that the total cost can be improved. Consequently, we only need to consider clients of affected facilities. This mechanism helps in reducing the time significantly by reducing the number of useless moves. We further strengthen the condition for populating *list* by observing the fact that the set of clients that can be affected in a facility m_k is a subset of $\{e_{p_j}\} \cup S_j$. The reason is all the other clients of m_k were not subject to any constraint from T_i (Line 22).

Algorithm 2 can tackle both RDCMST and ERDCMST. For the former, there would be only one facility. In addition, *FeasibleMove* will only check path-length constraint.

5 Long-Reach Passive Optical Networks

We now describe a real-world application whose instances are used for evaluating our approach. Long-Reach Passive Optical Networks (LR-PONs) provides an economically viable solution for fibre-to-the-home network architectures [1]. In LR-PON fibres are distributed from the Metro-Nodes (MNs) to the Exchange-Sites (ESs) through cables that forms a tree distribution network. A major fault occurrence in LR-PON would be a complete failure of the MN, which could

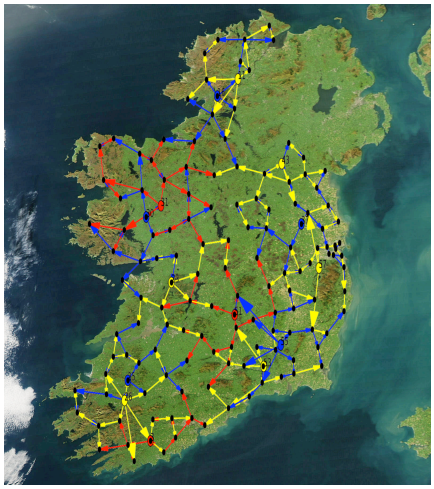


Fig. 3. Example of a LR-PON network for Ireland where each exchange-site is connected to two metro-nodes through disjoint paths. In the plot the subnetwork of each metro node is associated with a colour. Two subnetworks may have the same colour if they do not share nodes.

affect tens of thousands of customers. The dual homing protection mechanism for LR-PON enables customers to be connected to two MNs, so that whenever a single MN fails all customers are still connected to a back-up. Notice that the paths from an ES to its two MNs cannot contain the same link. Otherwise, this would void the purpose of having two MNs. Given as association of MNs with ESs the problem is to determine the routes of cables such that there are two edge-disjoint paths from an ES to its two MNs, the length of each path is below threshold and the total cable length required for connecting each ES to two MNs is minimised. Notice that here metro-nodes are facilities and exchange-sites are clients.

To evaluate the performance of the proposed local search algorithm and the move operators we use three datasets corresponding to real networks from three EU countries: Ireland with 1121 ESs, the UK with 5393 ESs, and Italy with 10708 ESs. For each dataset we use [10] to identify the position of the MNs and computed four instances for each country. Ireland with 18, 20, 22, and 24 MNs; the UK with 75, 80, 85, and 90 MNs; and Italy with 140, 150, 160, and 170 MNs. The LS algorithm starts with the trivial solution which consists in connecting all ESs (clients) to their MNs (facility). A solution obtained using our approach for Ireland is shown in Figure 3.

6 Empirical Evaluation

6.1 Experimental Protocol

All the experiments were performed on a 4-node cluster, each node features 2 Intel Xeon E5430 processors at 2.66Ghz and 12 GB of RAM memory.

In addition to the proposed MIP and local search approaches, we also considered BKRUS [2], a well known heuristic for solving RDCMST when the cost function and the delay function in RDCMST are equivalent and follow the Triangle inequality property (i.e., given three points a, b, and c then $d(a,b)+d(b,c) > d(a,c)$). The overall complexity of BKRUS is cubic in terms of number of nodes, which hinders its scalability when the bound on the path-length is tight. Although there are approaches that are better than BKRUS [5, 6], our aim is not to claim superiority over the RDCMST approaches since we are solving a more general problem for which these approaches would not be applicable. Instead, our aim is to show that although our approach is not specialised for RDCMST, it still provides very good quality solutions.

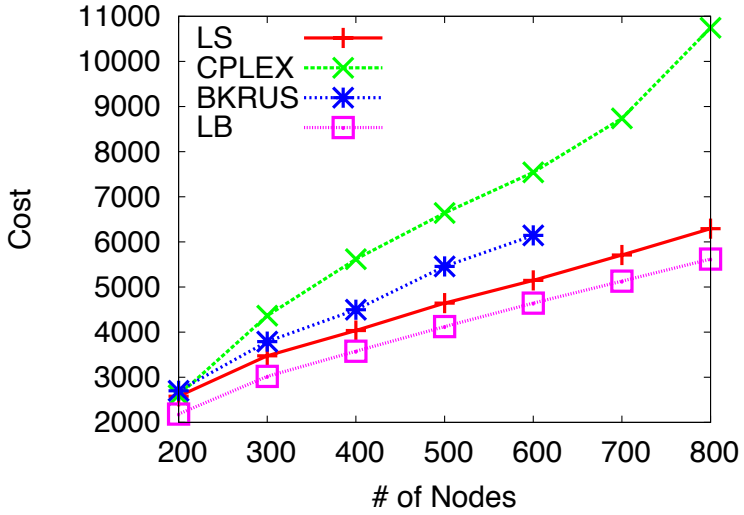
In particular in this paper we consider the following three experimental scenarios.

- *RDCMST*: In this scenario we report results on a set of real-life instances coming from our industrial partner in Ireland, each instance contains $|E| \in \{200, 300, \ldots, 800\}$ nodes. In this dataset the minimum valid value for λ is 415.
- *ERDCMST*: In this case we consider the following two scenarios:
  - *Real-life*: We consider real-life instances coming from our industrial partners with real networks in Ireland, the UK, and Italy detailed in the previous section.
  - *Random*: We generated 10 random instances extracted from the previous real-life network in Ireland. Each instance is generated by using 18 facilities and for each facility we randomly select $|E| \in \{100, 200, \ldots, 1000\}$ nodes.

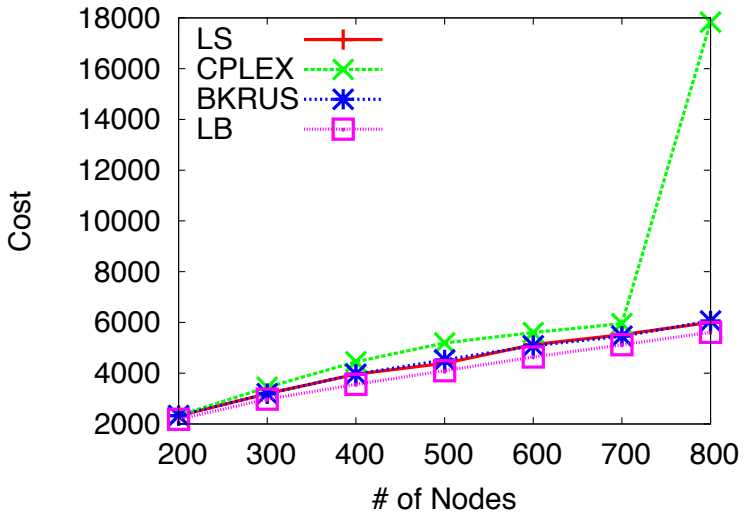
6.2 Experimental results

Figures 4(a) and 4(b) report results of the RDCMST problem for the three approaches: the local search approach using the subtree move-operator (LS), CPLEX, BKRUS, and the lower bound of the solution computed using CPLEX (LB). For each experiment we increase λ from 415 to 623, and used a timelimit of one hour. When λ is not playing an important role (i.e., λ=623) the problem is close to the unbounded minimum spanning tree, and except for CPLEX with $|E|$=800, the three approaches report near-optimal values. However, for tight values of λ (i.e., 415) the problem becomes more challenging and BKRUS is unable to solve instances with $|E|$>700, and the gap for CPLEX (w.r.t. the lower bound) considerably increases as the number of nodes increases. On the other hand, for the difficult case (λ=415) LS reports better upper bounds than CPLEX and BKRUS in all 8 instances, and the quality of the solution w.r.t. the lower bound does not degrade with the problem size.

Table 2 reports results for the random set of instances of the ERDCMST problem, here we depict the median value across 11 independent executions of the node and subtree operators; the best solution obtained with CPLEX; the best solution obtained with CPLEX using the solution of the first execution of



(a) λ=415



(b) λ=623

Fig. 4. RDCMST results with different limits on path-length

LS (subtree operator); and the best known LBs for each instance obtained with CPLEX using a larger time limit. The time limit for each local search experiment was set to 30 minutes, and 4 hours for CPLEX-based approaches.

In these experiments we observe that the subtree operator generally outperforms the node operator, we attribute this to the fact that moving a complete subtree helps to maintain the structure of the tree in a single iteration of the algorithm. The node operator might eventually reconstruct the structure, how-

Table 2. Results for the small-sized instances of ERDCMST problem where $|M| = 18$ and λ=67

$\|E\|$	LS (Subtree)	LS (Node)	CPLEX	LS+CPLEX	LB
100	**4674**	**4674**	**4674**	**4674**	4674
200	6966	6988	**6962**	**6962**	6962
300	8419	8575	**8404**	**8404**	8152
400	9728	10008	9728	**9721**	9329
500	**11203**	11672	11318	**11203**	10298
600	**11885**	12559	12276	11924	10517
700	13148	13981	13812	**13140**	11485
800	14040	15133	15118	**13977**	12402
900	14770	16098	16438	**14839**	12860
1000	**15962**	17479	18174	16009	13943

ever, more iterations would be required. CPLEX-based approaches report the optimal solution for 100 and 200 clients, while the median execution of the local search approaches reported the optimal solution for 100 clients, and the subtree operator reached the optimal solution in 5 out of the 11 executions for 200 clients. After $|E|$=500 LS dominates the performance for a margin ranging between 1% ($|E|$=500) to 12% ($|E|$=900). Moreover, cplex+LS was only able to improve the average performance of the subtree operator in a very small factor (up to 0.4% for $|E|$=800) after running the solver for 4 hours. We also experimented with instances with $|E|$<100 and $|E|$>800. In the first case the three algorithms reported similar results. In the second case only LS was able to provide good quality solutions with a Gap of 10% w.r.t. the LB, while BKRUS reported timeouts and CPLEX a Gap of about 59%.

Table 3. Results for Ireland, UK and Italy

Country	$\|M\|$	Subtree	CPLEX	LB	Gap-Subtree	Gap-CPLEX
	18	**17107**	26787	14809	13.43	44.71
Ireland	20	**16819**	83746	14845	11.73	82.27
$\|E\|$=1121	22	**16711**	79919	14990	10.29	81.24
	24	**16163**	26918	14570	9.85	45.87
	75	**65377**	285014	54720	16.30	80.80
UK	80	**64565**	301190	54975	14.85	81.74
$\|E\|$=5393	85	**63517**	281546	55035	13.35	80.45
	90	**62163**	220041	55087	11.38	74.96
	140	**89418**	–	76457	14.49	–
Italy	150	**88255**	–	76479	13.34	–
$\|E\|$=10708	160	**88336**	–	76794	13.06	–
	170	**87405**	–	77013	11.88	–

Now, we move our attention to Table 3 where we report results for real ERDCMST instances. In this case, we used a time limit of one hour for LS (using the subtree move operator), and four hours for cplex. As it can be observed, LS dominates the performance in all these experiments, and once again the solution quality of LS does not degrade with the problem size. Indeed, the Gap w.r.t. to the LB for local search varies from 9% to 13% for Ireland, 11% to 16% for the UK, and 12% to 14% for Italy. CPLEX ran out of memory when solving instances from Italy, and we report '–' since no valid solutions were obtained. For the UK instances CPLEX also ran out of memory before the time limit.

7 Conclusions and Future Work

We have presented an efficient local search algorithm for solving Edge Disjoint Rooted Distance-Constrained Minimum Spanning-Trees problem. We presented two novel move operators along with their complexities and an incremental evaluation of the neighbourhood and the objective function. Any problem involving tree structures could benefit from these ideas and the techniques presented make sense for a constraint-based local search framework where this type of incrementally is needed for network design problems. The effectiveness of our approach is demonstrated by experimenting with a set of problem instances taken from real-world long-reach passive optical network deployments in Ireland, the UK, and Italy. We compare our approach with a MIP-based exact approach and a spanning tree-based heuristic approach. Results show that our approach is superior in terms of scalability and its anytime behaviour.

In future we would like to extend ERDCMST with the notion of optional nodes, since this extension is a common requirement in several applications of ERDCMST. Effectively this means that we would compute for every facility a Minimum Steiner Tree where all clients are covered but the path to them may follow some optional nodes. We also plan to make our constraint-based local search approach parallel in two different ways. First, we intend to study the performance of the portfolio approach [11] where multiple copies of the algorithm compete and cooperate to solve a given problem instance, and second we intend to exploit a decomposition of the problem into smaller sub-problems that can be solved in parallel. Preliminary results have been presented in [12][2].

Acknowledgments

This work was supported by DISCUS (FP7 Grant Agreement 318137), and Science Foundation Ireland (SF) Grant No. 10/CE/I1853. The Insight Centre for Data Analytics is also supported by SFI under Grant Number SFI/12/RC/2289.

[2] We remark that [12] has been presented in a workshop without formal proceedings

References

1. Payne, D.B.: FTTP deployment options and economic challenges. In: Proceedings of the 36th European Conference and Exhibition on Optical Communication (ECOC 2009). (2009)
2. Oh, J., Pyo, I., Pedram, M.: Constructing minimal spanning/steiner trees with bounded path length. Integration **22**(1-2) (1997) 137–163
3. Ruthmair, M., Raidl, G.R.: A kruskal-based heuristic for the rooted delay-constrained minimum spanning tree problem. In: Computer Aided Systems Theory-EUROCAST 2009. Springer (2009) 713–720
4. Gomes, C.P.: Computational sustainability: Computational methods for a sustainableenvironment, economy, and society. The Bridge **39**(4) (2009) 5–13
5. Leitner, M., Ruthmair, M., Raidl, G.R.: Stabilized branch-and-price for the rooted delay-constrained steiner tree problem. In Pahl, J., Reiners, T., Vo, S., eds.: INOC. Volume 6701 of Lecture Notes in Computer Science., Springer (2011) 124–138
6. Ruthmair, M., Raidl, G.R.: Variable neighborhood search and ant colony optimization for the rooted delay-constrained minimum spanning tree problem. In Schaefer, R., Cotta, C., Kolodziej, J., Rudolph, G., eds.: PPSN (2). Volume 6239 of Lecture Notes in Computer Science., Springer (2010) 391–400
7. Hoos, H., Stütze, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
8. Hentenryck, P.V., Michel, L.: Constraint-based local search. The MIT Press (2009)
9. Pham, Q.D., Deville, Y., Hentenryck, P.V.: Ls(graph): a constraint-based local search for constraint optimization on trees and paths. Constraints **17**(4) (2012) 357–408
10. Ruffini, M., Mehta, D., O'Sullivan, B., Quesada, L., Doyle, L., Payne, D.B.: Deployment strategies for protected long-reach pon. Journal of Optical Communications and Networking **4** (2012) 118–129
11. Arbelaez, A., Codognet, P.: From sequential to parallel local search for SAT. In: 13th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP'13). (2013) 157–168
12. Arbelaez, A., Mehta, D., O'Sullivan, B., Quesada, L.: A constraint-based parallel local search for disjoint rooted distance-constrained minimum spanning tree problem. In: Workshop on Parallel Methods for Search & Optimization. (2014)