Rooted Maximum Weight Connected Subgraphs with Balancing and Capacity Constraints

Ralf Borndörfer
Zuse Institute Berlin
Berlin, Germany
borndoerfer@zib.de

Stephan Schwartz
Zuse Institute Berlin
Berlin, Germany
schwartz@zib.de

William Surau
Zuse Institute Berlin
Berlin, Germany
surau@zib.de

ABSTRACT

Finding connected subgraphs of maximum weight subject to additional constraints on the subgraphs is a common (sub)problem in many applications. In this paper, we study the Maximum Weight Connected Subgraph Problem with a given root node and a lower and upper capacity constraint on the chosen subgraph. In addition, the nodes of the input graph are colored blue and red, and the chosen subgraph is required to be balanced regarding its cumulated blue and red weight. This problem arises as an essential subproblem in district planning applications. We show that the problem is NP-hard and give an integer programming formulation. By exploiting the capacity and balancing condition, we develop a powerful reduction technique that is able to significantly shrink the problem size. In addition, we propose a method to strengthen the LP relaxation of our formulation by identifying conflict pairs, i.e., nodes that cannot be both part of a chosen subgraph. Our computational study confirms the positive impact of the new preprocessing technique and of the proposed conflict cuts.

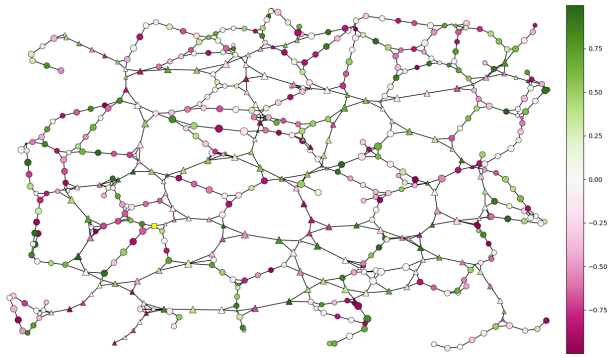

Figure 1: Exemplary BRCMWCS instance. The root node is depicted as a yellow square, circles represent nodes in V_b and triangles nodes in V_r, node sizes correspond to weights and colors to node profits.

1 INTRODUCTION

The Maximum Weight Connected Subgraph Problem (MWCS) is to find a node set of maximum cumulated weight that induces a connected subgraph in a given node-weighted graph. Popular variants of the MWCS include a given set of roots that have to be included in the chosen subgraph, and a capacitated (or budgeted) variant where additional node weights and lower and upper weight bounds for the chosen subgraph are specified. These variants occur in various applications, and have been the subject of a number of studies, see e.g. [2, 3, 11, 18].

In this paper, we study a variant of the rooted and capacitated MWCS where additional balancing constraints are imposed. Emerging from an application in district planning, the nodes are divided into blue and red nodes, and the chosen subgraph has to be balanced with respect to the color weights. The problem of balancing blue and red nodes in a connected subgraph has been studied, for instance, in [4, 20]. The combination of balanced, rooted, and capacitated MWCS, however, is new and interesting in its own right. It turns out that the combination of weight bounds and balancing constraints imposes a combinatorial structure that can be exploited to shrink the problem substantially and to derive logical implications on the shape of the subgraph.

The problem studied in this paper occurs as a subproblem when designing control districts for toll enforcement inspectors on motorways, see [6]. By a transition to the line graph of the motorway network, highway sections become nodes, whose lengths then correspond to node weights, and control districts are designed subject to lower and upper length bounds. In addition, the districts are desired to be homogeneous with respect to the traffic on its motorways, and homogeneity is measured as the cumulated difference to the median traffic of the district. Districts are generated dynamically in [6] and when fixing a root as potential median traffic node, all other nodes can be colored blue or red, representing motorways with less or, respectively, more traffic than the root node. Enforcing that the root has indeed the median traffic corresponds to the balancing condition of the BRCMWCS. The node profits stem from the duals of the restricted master problem, and are the only data that changes in each pricing round.

The contributions and the structure of this paper are as follows. In Section 2 we formalize the problem, review the literature, and present an IP formulation. In Section 3 we propose preprocessing methods that can drastically reduce the problem size. Section 4 is concerned with conflict pairs, i.e., pairs of vertices that cannot be both part of a feasible solution. Adding the resulting inequalities to the IP can considerably strengthen the LP relaxation. We conclude with a computational study in Section 5 that shows the strong effect of the preprocessing and the potential of the conflict cuts.

2 THE PROBLEM

For the *Balanced, Rooted, and Capacitated Maximum Weight Connected Subgraph Problem (BRCMWCS)*, we consider a graph $G = (V, E)$ with a bipartition of the nodes $V = V_b \dot\cup V_r$ into blue and red nodes, node weights $w \in \mathbb{R}^V_{\geq 0}$ and profits $p \in \mathbb{R}^V$, as well as numbers $0 \leq W_L \leq W_U$, $\Delta \geq 0$, and a root node $r \in V$. The BRCMWCS is to find a tree $T = (V_T, E_T)$ of weight $w(V_T) \in [W_L, W_U]$, such that $r \in V_T$ and $|w(V_T \cap V_b) - w(V_T \cap V_r)| \leq \Delta$, and such that $p(V_T)$ is maximized. Here and in the following, we use the short notation $w(V') := \sum_{v \in V'} w_v$ for a subset $V' \subseteq V$, and, accordingly, for $p(V')$.

Complexity. The BRCMWCS is NP-hard and the reduction can come from multiple angles. For instance, even without the balancing and capacity constraints, i.e., by setting Δ and W_U sufficiently large, a reduction from the rooted MWCS (which is NP-hard due to [2]) is possible. On the other hand, since the balanced connected subgraph problem is NP-hard (see [4]), a reduction is also possible without the root or capacity constraints. Note that if the rooted case could be solved in polynomial time, the unrooted version can be solved in polynomial time by considering every vertex as potential root node. Finally, the capacity contraints alone make the problem NP-hard. A reduction from the number partition problem is possible by considering a star graph where the leaf weights are the numbers of the partition instance and the center has weight 0. By setting $W_L = W_U$ to half of the sum of all weights, we see that the BRCMWCS is NP-hard even without the root or balancing condition.

2.1 Related Work

The MWCS is a classical optimization problem with close relations to the family of Steiner tree problems, see e.g. [12, 19, 22] for transformations and context. A number of papers from the last decade study preprocessing techniques [13, 22, 23], exact approaches [1, 14, 21], or applications [3, 8, 9] for the MWCS. The standard preprocessing approaches, however, do not carry over to the budgeted variant, since the bounds might require the inclusion of nodes with negative profit or the exclusion of nodes with positive profit in the optimal solution. Our paper adresses exactly this situation.

The budgeted MWCS with a lower bound has been considered in [15, 17, 18], but the nodes bear unit weights and $W_L = W_U = k$, i.e., the goal is a maximum weight connected subgraph with exactly k nodes. While Hochbaum and Pathria [15] propose a dynamic program that finds the optimal solution on trees and a $\frac{1}{k}$-approximation on general graphs, the authors of [18] reduce the problem to the single-rooted case which is heuristically solved in [17].

The rooted and budgeted MWCS has been studied in [2, 11], but only with an upper weight bound, i.e., $W_L = 0$. Both of these works focus on the comparison of different connectivity formulations. The results of Dilkina and Gomes [11] indicate that a single-commodity flow is best if the upper weight bound is impractically large, while in the other case, a formulation based on arc separation is preferred. Álvarez-Miranda et al. [2] propose a node separator formulation, and find that this formulation is favorable for denser graphs, compared to the arc separation. Comparisons of different connectivity formulations for the pricing problem in [6] strongly suggest that a single-commodity flow is best suited for the BRCMWCS.

For the *Balanced Connected Subgraph* Problem (BCS) we are given a graph $G = (V_b \dot\cup V_r, E)$ with nodes colored either blue or red, and seek a maximum-cardinality subgraph that contains an equal number of blue and red nodes. The problem was introduced in [4] and shown to be NP-hard, even on planar, bipartite, and chordal graph, or when a single root node is specified. When the graph is a tree, however, Bhore et al. [4] give a labelling algorithm to solve the BCS in time $O(|V|^4)$. Kobayashi et al. [16] improve the runtime to $O(|V|^2)$ by running a dynamic program on a transformed rooted binary tree with possibly additional uncolored nodes. The authors also briefly study a weighted version of BCS and give complexity results for special graph classes. Further complexity and inapproximability results as well as polynomial-time algorithms for the BCS on other special graph classes are provided in [4, 5, 10, 20]. We are not aware of any preprocessing or cutting plane approaches to the BCS.

2.2 IP Formulation for BRCMWCS

We use a flow formulation to ensure the connectivity of the chosen subgraph T. The idea is to construct a flow that emerges at the root node. Each node of the chosen subgraph consumes one unit of flow, while all other nodes satisfy flow conservation. To this end, we consider the bidirected version $D = (V, A)$ of G and introduce the variables

- $y_v \in \{0, 1\}$ for $v \in V$ indicating whether $v \in T$,
- $x_a \geq 0$ for $a \in A$ specifying the flow on arc $a \in A$.

The IP formulation of the BRCMWCS can then be stated as follows.

$$\max_{x,\, y} \quad \sum_{v \in V} p_v\, y_v \tag{1a}$$

$$\text{s.t.} \quad W_L \leq \sum_{v \in V} w_v\, y_v \leq W_U \tag{1b}$$

$$\sum_{v \in V_r} w_v\, y_v - \sum_{v \in V_b} w_v\, y_v \leq \Delta \tag{1c}$$

$$\sum_{v \in V_b} w_v\, y_v - \sum_{v \in V_r} w_v\, y_v \leq \Delta \tag{1d}$$

$$y_r = 1 \tag{1e}$$

$$\sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = y_v \qquad \forall v \in V \setminus \{r\} \tag{1f}$$

$$x_{uv} \leq M\, y_v \qquad \forall (u, v) \in A \tag{1g}$$

$$x_a \geq 0 \qquad \forall a \in A \tag{1h}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{1i}$$

The objective (1a) is to maximize the profit of the chosen subgraph. The budget constraints are specified in (1b), and the balancing constraints in (1c) and (1d). The flow conservation is ensured by equalities (1f). Inequalities (1g) are necessary to activate every node that is used by the flow. The use of a big M parameter is bad for the LP relaxation, but necessary. It should be chosen as small as possible, and following [24], $M = \max_{V' \subseteq V} \{|V'| : w(V') \leq W_U\} - 1$ is a valid choice that can be computed with a greedy algorithm.

3 PREPROCESSING

Computational studies show that preprocessing methods for the MWCS generally have a huge impact on the solution time [13, 22, 23]. While the general methods for the MWCS do not carry over to the BRCMWCS, we propose an effective approach that significantly reduces the problem size and computation times for our problem.

The capacity constraints in combination with the balancing condition allow for different reduction techniques for the BRCMWCS. In particular, we can derive the following weight range for the chosen blue vertices $V_T \cap V_b$:

$$\frac{W_L - \Delta}{2} \leq w(V_T \cap V_b) \leq \min\left(\frac{W_U + \Delta}{2}, w(V_r) + \Delta\right). \quad (2)$$

The weight range for the red color class is defined accordingly, and we denote the respective upper weight bounds by W_U^b and W_U^r. A method to discard nodes based on these color weight ranges was proposed in [6]: For each color, we determine the color radius, i.e., the set of nodes that can be reached from the root on a path that satisfies the upper weight bound of the according color. The color radius can be determined with a single shortest path tree computation in the bidirected version of G using arc weights

$$\tilde{w}^b_{(u,v)} := \begin{cases} w_v & , \text{if } v \in V_b, \\ 0 & , \text{else,} \end{cases}$$

and $\tilde{w}^r$ defined accordingly. Every node that is outside of the color radius cannot be part of any feasible solution and can be removed.

Here, we propose a complementary preprocessing approach to discard even more nodes. The *bicolor radius* is the set of nodes that can be reached from the root on a path that satisfies the upper weight bound of both color classes. Unfortunately, the bicolor radius cannot be computed via a shortest path tree. In fact, it is essentially a constrained shortest path problem to determine if a specified node is inside the bicolor radius. We solve the problem with a Bellman-Ford-like labelling algorithm. The approach is detailed in Algorithm 1 and uses two-dimensional arc weights and node labels (for the blue and red cumulated weight, respectively). The upper weight bounds, given as a pair $\ell_{\max} = (W_U^b, W_U^r)$, are part of the input. We use the usual notion of domination, i.e., label ℓ_1 dominates label ℓ_2 if each weight in ℓ_1 is less than or equal to the corresponding weight in ℓ_2 and if at least one of the inequalities is strict. The algorithm is closely related to constrained shortest path labelling approaches and runs in pseudo-polynomial time.

The effect of the bicolor radius preprocessing can be substantial, and goes far beyond the single color radii. For the instance depicted in Figure 2, the single color radii cannot exclude a single node. The bicolor radius, on the other hand, is able to eliminate all gray nodes, essentially eliminating half of the graph.

4 CONFLICT PAIRS

In this section, we propose a method to identify and make use of conflict pairs in the BRCMWCS. A conflict pair consists of two nodes $u, v \in V$ that cannot be both part of a feasible solution, i.e., $u \in T \implies v \notin T$ for any feasible solution T. For any conflict pair (u, v) we can potentially tighten the LP relaxation of (1) with the inequality $y_u + y_v \leq 1$.

Algorithm 1: BicolorRadius

Input: $G = (V_b \dot\cup V_r, E), w, r, \ell_{\max}$
Output: set of nodes within the bicolor radius

Consider arc weights ω in the bidirected version of G with $\omega_{(u,v)} \leftarrow \left(\tilde{w}^b_{(u,v)}, \tilde{w}^r_{(u,v)}\right)$;
$\ell_r \leftarrow \begin{cases} (w_r, 0) & , \text{if } r \in V_b, \\ (0, w_r) & , \text{else} \end{cases}$
$labels[v] \leftarrow \begin{cases} \{\ell_r\} & , \text{if } v = r, \\ \varnothing & , \text{else} \end{cases}$;
$L \leftarrow \{(r, \ell_r)\}$;
repeat
  $L' \leftarrow \varnothing$;
  for $(u, \ell_u) \in L$ **do**
    for *each neighbor* v *of* u *in* G **do**
      $\ell' \leftarrow \ell_u + \omega_{(u,v)}$;
      if ℓ' *respects* $\ell_{\max}$ *and is not dominated by any label at* v **then**
        Add ℓ' to $labels[v]$;
        Add (v, ℓ') to L' ;
        Remove dominated labels in $labels[v]$;
  $L \leftarrow L'$;
until $L = \varnothing$;
return *all nodes with at least one label* ;

The idea of analyzing conflicting pairs (or even larger sets), and deriving stronger constraints has been proven to be very useful in different set packing problems, such as knapsack or matching problems. The proposed idea also translates to the rooted and budgeted MWCS, i.e., without the colors and balancing condition.

4.1 Finding Conflict Pairs

In order to identify conflict pairs, we can make use of the upper capacity bounds introduced in Section 3 again. If for two nodes u

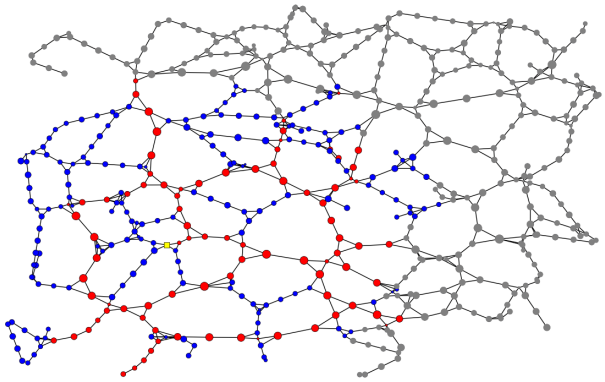

Figure 2: Effect of bicolor radius preprocessing. The root node is depicted as a yellow square, the gray nodes are outside of the bicolor radius and can be removed.

Algorithm 2: SteinerTreeConflictPairs

Input: $D = (V, A)$, r, $\tilde{w} \in \mathbb{R}^A_{\geq 0}$, $w_{\max}$
Output: set C of conflict pairs

```
1  C ← ∅ ;
2  len ← all pairs shortest path lengths in D w.r.t. w̃ ;
3  for all node pairs u, v ∈ V \ {r} do
4      for c ∈ V do
5          weight ← len[(r, c)] + len[(c, u)] + len[(c, v)] ;
6          if weight ≤ w_max then
7              // there is a feasible Steiner tree
8              Go to line 3 and check the next node pair ;
9      Add (u, v) to C ;
10 return C ;
```

and v and for every tree $T = (V_T, E_T)$ containing r, u, and v, we know that $w(V_T) > W_U$ or $w(V_T \cap V_b) > W_U^b$ or $w(V_T \cap V_r) > W_U^r$, then (u, v) is a conflict pair. Deciding if there exists a tree connecting a given set of terminal vertices at some cost, is a Steiner tree problem. Fortunately, an elegant combinatorial approach is known for the Steiner tree problem with three terminals. It is based on the insight that any Steiner tree with three terminals is a union of paths from a common center node (that is possibly a terminal itself) to the terminals.

Building on this, Algorithm 2 describes our procedure to identify conflict pairs. As input, we use the bidirected version D of G, the root node r, and one of the following three weight combinations:

- $\tilde{w}$ with $\tilde{w}_{(u,v)} = w_v$, $w_{\max} = W_U - w_r$,
- $\tilde{w}^b$, $w_{\max} = W_U^b - w_r \cdot \mathbf{1}_{V_b}(r)$,
- $\tilde{w}^r$, $w_{\max} = W_U^r - w_r \cdot \mathbf{1}_{V_r}(r)$,

where $\mathbf{1}$ is the indicator function. The union of the three respective return values of Algorithm 2 constitutes our set of conflict pairs. The proposed algorithm runs in time $O(|V|^3)$, and for all tested instances, it is able to identify a large number of conflict pairs. In fact, in most cases there are so many conflict pairs that some strategy is needed to exploit this information. We will discuss two approaches: Deriving large conflict sets and identifying essential conflicts.

4.2 The Conflict Graph

The individual conflict pair inequalities $y_u + y_v \leq 1$ can be too weak to effectively strengthen the LP relaxation. We construct the *conflict graph* on the vertex set V by introducing an edge for every conflict pair. Analogously to the set packing problem (see [7] for details), we can derive stronger inequalities from the conflict graph: Given an odd cycle C of length $2k + 1$ in the conflict graph, the inequality $\sum_{v \in C} y_v \leq k$ is known to be stronger than the ordinary conflict pair inequalities. These odd-cycle cuts, however, usually do not help the optimization process. The situation is different when considering a clique C in the conflict graph. The clique cuts $\sum_{v \in C} y_v \leq 1$ are known to be often beneficial for the LP relaxation.

We experimented with including clique cuts to formulation (1). Since the number of cliques is even much larger than the number of conflict pairs, we determined an edge covering with cliques, and only added the respective clique cuts. These additional inequalities, however, showed to have a negative effect on the solution time in our tests. A typical conflict clique together with an optimal solution for the BRCMWCS instance is depicted in Figure 3. One can see that the conflicting nodes are quite far apart and located at the boundary of the graph. Depending on the node profits of the instance, such clique cuts are rarely violated by the optimal LP relaxations. Hence, we shift our focus to identifying more meaningful conflict pairs.

4.3 Essential Conflicts

Our experiments showed that the addition of all conflict cuts results in significantly longer solution times. The same holds true if only violated cuts are added dynamically to the program. Hence, it is necessary to identify essential conflicts that actually facilitate the solution process. Such conflicts presumably involve nodes that are closer to the root node or have a high profit.

In order to specify these nodes, we define a scoring function that assigns a value from the interval $[-1, 1]$ to every vertex. A positive profit as well as a small ratio between the shortest r-v-path length and W_U increase the score of node v. Essential conflicts are then defined as all conflict pairs between nodes with a positive score. Figure 4 shows the node scores and all resulting essential conflicts for an exemplary instance. Observe that the conflict sets are now much more central. Again, we determine an edge clique cover of the essential conflict graph and only added the respective clique cuts.

5 COMPUTATIONAL RESULTS

In our computational study, we evaluate the impact of the bicolor preprocessing and the conflict pair cuts. We ran the experiments on machines equipped with Intel Xeon E3-1234 CPUs with 3.7GHz and 32GB RAM. The code is written in Python 3.6 and to solve IPs Gurobi 9.1 is used. The instances stem from a transit network generator used in [6]. These networks are edge weighted with a length and a traffic value. We transform the transit networks into node weighted graphs. The weight of a node represents the length of the corresponding edge. As a root we chose a random node and

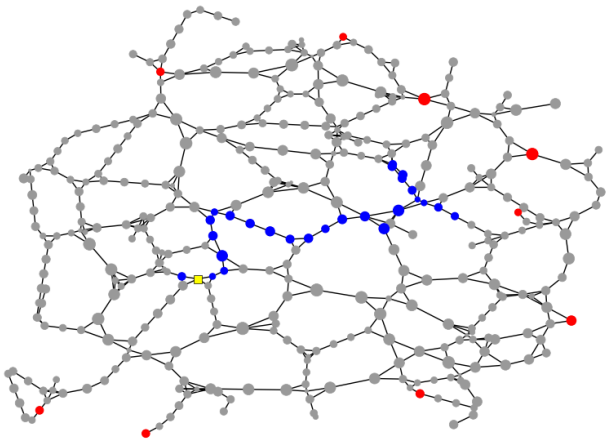

Figure 3: The nodes of an exemplary conflict clique are colored red, an optimal solution is colored blue.

Table 1: Reduction effect of the preprocessing and numbers of (essential) conflict pairs for instances grouped with respect to the four different weight bounds `W1`-`W4`.

group	*rp*	*cp*1	*ecp*1	*cp*2	*ecp*2
`W1`	227.7	13624.1	110.0	30426.5	146.6
`W2`	95.2	11132.1	103.0	46410.6	175.9
`W3`	25.3	4033.5	56.7	38420.7	146.0
`W4`	3.7	156.1	6.8	18688.0	78.2

the color of a node depends on whether the traffic value is higher or lower than the traffic at the root. Finally, Δ is set to the weight of the root node.

We constructed 5 such network instances where the number of nodes ranges from 563 to 569 and the number of edges from 845 to 885. For each network, we consider 9 different profits: 3 are actual reduced costs from the pricing problem from [6], 3 are distributed uniformly at random in the interval $[-1, 1]$, and 3 are based on the normal distribution to thin out extreme profits. For one instance per profit class, we also used a random partition into red and blue vertices. Finally, we consider 4 different upper weight bounds that lie between 40% and 75% of the graph diameter, and are motivated by our application. The lower bounds were set to half of the upper bounds but showed no effect in any computation. Altogether we have $5 \cdot 12 \cdot 4 = 240$ test instances.

The base variant of the study is the formulation without preprocessing and conflict cuts, which is denoted by `00`. To assess the impact of the proposed preprocessing, we consider the variant `10`. For a broader perspective, we consider two sets of conflict pairs generated by Algorithm 2: The set *cp*1 is the union of the return values for the input $\tilde{w}^b$ and $\tilde{w}^r$. The set *cp*2, on the other hand, is the combined output of all three weight combinations. The essential conflict sets *ecp*1 and *ecp*2 are formed by the presented scoring function. To evaluate the impact of the conflict pairs, we consider the variant `11` where, in addition to the preprocessing, conflict cuts

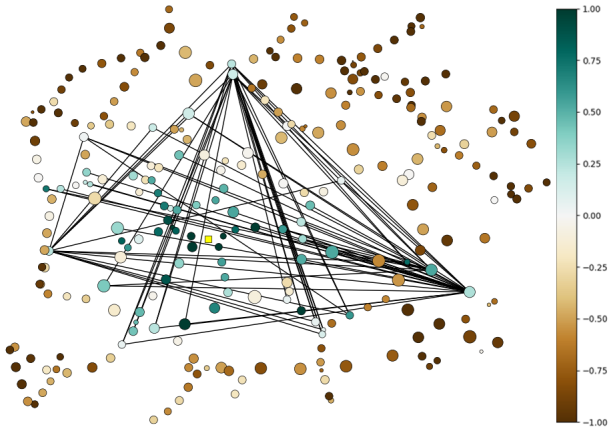

Figure 4: Essential conflicts and node scores of an exemplary instance.

Table 2: Comparison of average computation times (in seconds) for different variants and instance groups.

group	`00`	`10`	`11`	`12`	`1*`
`small`	42.3	22.7	23.8	23.9	21.6
`medium`	255.8	170.3	171.3	176.2	149.8
`large`	1996.7	1775.9	1527.6	1478.1	1188.0

to the set *ecp*1 are added, and the variant `12` defined accordingly with the set *ecp*2.

Table 1 shows the impact of the preprocessing and the number of generated (essential) conflict pairs. The instances are partitioned into four groups corresponding to the upper weight bounds. We report on the average numbers of removed nodes by the preprocessing and of (essential) conflict pairs for the two described variants. Overall, we observe that with increasing upper weight bound, the effect of the preprocessing and the number of conflict pairs decrease. Also observe that most conflict pairs are found with respect to $\tilde{w}$, and that the reduction to essential conflict pairs is significant in either case. Finding a decent subset of conflict pairs that support the solution process is a big challenge. We proposed a scoring function to determine a set of essential conflicts. While our computational results show that this approach is already beneficial in many instances, we like to point out that the potential of the conflict cuts is even larger. To this end, we consider another hypothetical variant, `1*`, that assumes for every instance the preferred cuts from *ecp*1 or *ecp*2. Consequently, the variant `1*` is attributed the minimum runtime of `11` and `12` for each instance. This variant essentially simulates a superior way to determine essential conflicts.

In order to compare the variants, we partition the 240 instances into three groups based on the magnitude of the computation time for the base variant: `small` (123 instances), `medium` (91 instances), and `large` (26 instances). All instances and instance-wise computational results are available in an online supplement[1] to this article. Table 2 contains the cumulated results. We report the geometric mean of the runtimes in seconds for the respective groups and variants. We opt for the geometric instead of the arithmetic mean to lessen the impact of outliers. In particular, there is one large instance that profits immensely from the additional cuts (speed-up factor 40).

We can observe that the preprocessing clearly helps to reduce the average computation times. While there are also instances where the preprocessing has a smaller or even a negative effect, the positive impact of the proposed method prevails in all groups. In terms of the conflict cuts, the impact is not so obvious. On average, for the small and medium instances, the cuts together with the preprocessing are slightly worse than the preprocessing alone. For the large instances, most of the positive impact admittedly stems from the single outlier instance. Nevertheless, a smaller positive effect remains when excluding this instance.

The average values, however, do not paint the whole picture. On many instances, one of the conflict cut variants is significantly better than variant `10`, on many other instances, the reverse is true. Interestingly, the variants `11` and `12` are often complementary, i.e.,

[1] https://github.com/stephanschwartz/brcmwcs

one set of essential cuts is much more favorable than the other. In order to assess the potential of the conflict cuts, variant 1⋆ serves as an oracle that always chooses the better of the two essential conflict sets. Indeed, we find that on average, this variant performs best in all groups. Furthermore, the positive impact increases for harder instances.

When considering the different variants over all instance classes, there is no clear indication on which classes some variant performs better than another. We cannot observe that a certain variant is dominant on a specific network instance. The same holds true for a specific profit class or weight bound. Even for the 15 instances that have a sibling only varying in node colors, the performance of the variants is vastly different for the pairs. Thus, it remains open in which cases the single variants are best. As one can expect, however, the computation time increases with larger (upper) weight bounds, since the number of feasible subgraphs drastically increases.

6 CONCLUSIONS AND OUTLOOK

In this paper, we studied a variant of the Maximum Weight Connected Subgraph problem that arises as a subproblem in a districting application. A distinct feature of the problem is that the chosen subgraph has to meet a given cumulated weight range and is required to have a similar weight of red and blue nodes. We use these features to develop a preprocessing method that is able to considerably reduce the average problem size and computation time.

In addition, we propose a method to identify node pairs that cannot be both part of a feasible solution. By considering the emerging conflict graph, we can strengthen the packing condition. The main problem, however, is to identify an appropriate subset of conflict pairs to add. To this end, we introduce a node scoring function that favors nodes close to the root and nodes with positive profit. Employing this scoring function, we generate two different sets of essential conflicts that are added to the IP formulation. Our computational results show that each of the sets individually has a positive impact on a number of instances. If we artificially choose the better set for each instance, the advantage of the conflict cuts becomes obvious. This suggests that the conflict cut approach is well suited for this problem, and that future work should aim to discover which conflict sets are essential for a given instance.

REFERENCES

[1] Eduardo Álvarez-Miranda, Ivana Ljubić, and Petra Mutzel. 2013. The maximum weight connected subgraph problem. In *Facets of Combinatorial Optimization*. Springer, 245–270.

[2] Eduardo Álvarez-Miranda, Ivana Ljubić, and Petra Mutzel. 2013. The rooted maximum node-weight connected subgraph problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 300–315.

[3] Christina Backes, Alexander Rurainski, Gunnar W Klau, Oliver Müller, Daniel Stöckel, Andreas Gerasch, Jan Küntzer, Daniela Maisel, Nicole Ludwig, Matthias Hein, Andreas Keller, Helmut Burtscher, Michael Kaufmann, Eckart Meese, and Hans-Peter Lenhof. 2012. An integer linear programming approach for finding deregulated subgraphs in regulatory networks. *Nucleic acids research* 40, 6 (2012), e43.

[4] Sujoy Bhore, Sourav Chakraborty, Satyabrata Jana, Joseph SB Mitchell, Supantha Pandit, and Sasanka Roy. 2019. The balanced connected subgraph problem. In *Conference on Algorithms and Discrete Applied Mathematics*. Springer, 201–215.

[5] Sujoy Bhore, Satyabrata Jana, Supantha Pandit, and Sasanka Roy. 2019. Balanced connected subgraph problem in geometric intersection graphs. In *International Conference on Combinatorial Optimization and Applications*. Springer, 56–68.

[6] Ralf Borndörfer, Stephan Schwartz, and William Surau. 2021. *Vertex Covering with Capacitated Trees*. Technical Report 21–25. Zuse Institute Berlin.

[7] Ralf Borndörfer and Robert Weismantel. 2000. Set packing relaxations of some integer programs. *Mathematical Programming* 88, 3 (2000), 425–450.

[8] Rodolfo Carvajal, Miguel Constantino, Marcos Goycoolea, Juan Pablo Vielma, and Andrés Weintraub. 2013. Imposing connectivity constraints in forest planning models. *Operations Research* 61, 4 (2013), 824–836.

[9] Chao-Yeh Chen and Kristen Grauman. 2012. Efficient activity detection with max-subgraph search. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1274–1281.

[10] Benoit Darties, Rodolphe Giroudeau, König Jean-Claude, and Valentin Pollet. 2019. The Balanced Connected Subgraph Problem: Complexity Results in Bounded-Degree and Bounded-Diameter Graphs. In *International Conference on Combinatorial Optimization and Applications*. Springer, 449–460.

[11] Bistra Dilkina and Carla P Gomes. 2010. Solving connected subgraph problems in wildlife conservation. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 102–116.

[12] Marcus T Dittrich, Gunnar W Klau, Andreas Rosenwald, Thomas Dandekar, and Tobias Müller. 2008. Identifying functional modules in protein–protein interaction networks: an integrated exact approach. *Bioinformatics* 24, 13 (2008), 223–231.

[13] Mohammed El-Kebir and Gunnar W Klau. 2014. Solving the maximum-weight connected subgraph problem to optimality. 11th DIMACS Implementation Challenge Workshop. (2014).

[14] Matteo Fischetti, Markus Leitner, Ivana Ljubić, Martin Luipersbeck, Michele Monaci, Max Resch, Domenico Salvagnin, and Markus Sinnl. 2017. Thinning out Steiner trees: a node-based model for uniform edge costs. *Mathematical Programming Computation* 9, 2 (2017), 203–229.

[15] Dorit S Hochbaum and Anu Pathria. 1994. Node-optimal connected k-subgraphs. *manuscript, UC Berkeley* (1994).

[16] Yasuaki Kobayashi, Kensuke Kojima, Norihide Matsubara, Taiga Sone, and Akihiro Yamamoto. 2019. Algorithms and Hardness Results for the Maximum Balanced Connected Subgraph Problem. In *International Conference on Combinatorial Optimization and Applications*. Springer, 303–315.

[17] Heungsoon Felix Lee and Daniel R Dooly. 1996. Algorithms for the constrained maximum-weight connected graph problem. *Naval Research Logistics (NRL)* 43, 7 (1996), 985–1008.

[18] Heungsoon Felix Lee and Daniel R Dooly. 1998. Decomposition algorithms for the maximum-weight connected graph problem. *Naval Research Logistics (NRL)* 45, 8 (1998), 817–837.

[19] Ivana Ljubić. 2020. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks* (2020), 177–204.

[20] T Martinod, V Pollet, B Darties, R Giroudeau, and J-C König. 2021. Complexity and inapproximability results for balanced connected subgraph problem. *Theoretical Computer Science* (2021).

[21] Daniel Rehfeldt, Henriette Franz, and Thorsten Koch. 2020. *Optimal Connected Subgraphs: Formulations and Algorithms*. Technical Report 20–23. Zuse Institute Berlin.

[22] Daniel Rehfeldt and Thorsten Koch. 2019. Combining NP-hard reduction techniques and strong heuristics in an exact algorithm for the maximum-weight connected subgraph problem. *SIAM Journal on Optimization* 29, 1 (2019), 369–398.

[23] Daniel Rehfeldt, Thorsten Koch, and Stephen J Maher. 2019. Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem. *Networks* 73, 2 (2019), 206–233.

[24] Hamidreza Validi and Austin Buchanan. 2021. Political districting to minimize cut edges. (2021). http://www.optimization-online.org/DB_HTML/2021/04/8349.html