Decomposition Branching for Mixed Integer Programming

Baris Yildiz[1], Natashia Boland[2], and Martin Savelsbergh[2]

[1]Department of Industrial Engineering, Koc University, Istanbul, Turkey
[2]H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, USA

Abstract

We introduce a novel and powerful approach for solving certain classes of mixed integer programs (MIPs): *decomposition branching*. Two seminal and widely used techniques for solving MIPs, branch-and-bound and decomposition, form its foundation. Computational experiments with instances of a weighted set covering problem and a regionalized p-median facility location problem with assignment range constraints demonstrate its efficacy: it explores far fewer nodes and can be orders of magnitude faster than a commercial solver and an automatic Dantzig-Wolfe approach.

1 Introduction

Applications of mixed integer programming can be found in many industries, such as transportation, healthcare, energy, and finance, and their economic impact is significant. This is due, in part, to the availability of effective and robust commercial solvers, such as FICO Xpress Optimization (`www.fico.com`), IBM ILOG CPLEX Optimization Studio (`www.ibm.com`), and Gurobi Optimizer (`www.gurobi.com`). What were initially highly specialized optimization codes, sold by small companies, have migrated to full optimization suites offered by large software and solution vendors.

It is well-known that mixed integer programs (MIPs) can be very difficult to solve. Their challenge continues to stimulate research in the design and implementation of efficient and effective techniques that can better solve them. For an overview of integer programming and its history, we refer the interested reader to Nemhauser and Wolsey (1988); Schrijver (1998); Wolsey (1998); Jünger et al. (2010); Bixby (2012); Conforti et al. (2014).

Two seminal and widely used techniques for solving MIPs, branch-and-bound and decomposition, form the foundation for the research presented in this paper, where we combine these techniques in an innovative way to develop a novel approach for solving MIPs: *decomposition branching*.

Branch-and-bound (Land and Doig, 1960) has become the standard paradigm for solving MIPs. Consequently, designing effective branching schemes has attracted substantial attention over the years. Major developments in this area include the introduction of pseudo-cost branching (Bénichou et al., 1971), strong branching (Applegate et al., 1995), and reliability branching (Achterberg et al., 2005). Surveys on branching schemes include Linderoth and Savelsbergh (1999) and Morrison et al. (2016). Recent research employs machine learning techniques to discover the best branching scheme

for solving a particular instance as the search progresses (Le Bodic and Nemhauser, 2015; Khalil et al., 2016; Lodi and Zarpellon, 2017; Dilkina et al., 2017). However, all this work concentrates on branching schemes based on single-variable dichotomy, and focuses primarily on the variable selection step, i.e., the choice of the variable to branch on. In this paper, we develop a new branching scheme that is quite different and *not* based on single-variable dichotomy. The scheme's branching rules divide the search space by exploiting decomposable structure in the problem.

It is well-known that real-life instances often exhibit a decomposable or nearly decomposable structure, where an instance has a nearly decomposable structure if the coefficient matrix has a bordered block-diagonal form: the nonzeroes in the matrix comprise a set of disjoint blocks with either a small set of linking constraints or linking variables. In their survey on progress in presolving for MIPs, Gamrath et al. (2015) observe that many "real-world supply chain management instances ... contain independent subproblems", meaning that they have a decomposable form with no linking variables nor linking constraints. Surprisingly, 34 out of 41 of the supply chain management instances used in their computational study had independent subproblems, and many had a large number of them. More than 180 independent subproblems were detected on average over the 41 instances. Bergner et al. (2015) develop techniques to automatically detect decomposable structure and test them on 39 instances from MIBLIB 2003 and 2010 (Achterberg et al., 2006; Koch et al., 2011). In the best decomposable arrangement found for each of the 39 instances, the majority (26 instances) had only linking constraints (i.e., no linking variables), which is the form of decomposable structure that we focus on in this paper. Out of those 26 instances, the linking constraints constituted less than 7% of the constraints in 23 cases, and less than 3% in 15 cases. It is possible, of course, that even more instances have decomposable structure with few linking constraints and no linking variables, but that the techniques of Bergner et al. (2015) found a different decomposable structure for them.

Several techniques have been developed for exploiting decomposable or nearly decomposable structure, such as Lagrangian relaxation (Geoffrion, 1974), Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960), and Benders decomposition (Benders, 1962). We note that branch-and-price (Barnhart et al., 1998), the application of Dantzig-Wolfe decomposition for the solution of MIPs, requires carefully designed branching rules to preserve the structure of the pricing problem (Vanderbeck and Savelsbergh, 2006; Vanderbeck and Wolsey, 2010; Vanderbeck, 2011). The popularity and importance of branch-and-price in applications, in particular, has initiated research into the automatic detection of decomposable structures in MIPs, which has advanced significantly in recent years (Bergner et al., 2015; Kruber et al., 2017). Quite recently, decomposable structure has also been exploited in an approach that uses an integer programming master problem derived from a multiobjective perspective (Bodur et al., 2016), as well as for generating cutting planes (Dey et al., 2017).

In the research described in this paper, we seek to use decomposable structure to define a new branching scheme. In their overview paper, Morrison et al. (2016) highlight that a great as-yet-unexplored research direction when it comes to branching is different partitioning schemes:

> "An important question here is: 'How should the B&B algorithm branch in order to generate the smallest number of unhelpful subproblems?' In this context, an unhelpful subproblem is a subproblem that does not lead to any optimal or near-optimal solutions... [so it does] not provide any gain for the algorithm but may still require substantial work to explore and prune. However, by considering a different partitioning scheme for the branching strategy, it may be possible to avoid exploring some of these

> unhelpful subproblems... Finally, it may be beneficial to explore 'hybrid' binary-wide branching strategies, which employ wide branching high in the search tree and switch to binary branching in lower regions, or vice versa." (Morrison et al., 2016)

As will become evident shortly, employing a different, wide partitioning scheme is exactly what decomposition branching does. The branching rule by which decomposition branching divides the search space at a node of the branch-and-bound tree is determined by solving a subproblem, and the resulting branching constraints cannot be expressed without reference to the value of the solution to the subproblem.

The idea of solving a subproblem to improve the performance of state-of-the-art branch-and-bound software (specifically, the SCIP software (Achterberg, 2009)) was exploited by Gamrath et al. (2015) in their survey on progress in presolving for MIPs, mentioned above. They discuss a new technique called *connected components*, which identifies "small subproblems that are independent of the remaining part of the problem and tries to solve those to optimality during the presolving phase". Using this new preprocessing technique on the 41 supply chain management instances led to an average reduction of over 18% in the number of variables and of over 16% in the number of constraints. Solving (small) subproblems to optimality is another key ingredient of decomposition branching.

More specifically, we propose an approach that exploits decomposable structure by branching. The key idea is to better solve a large MIP by solving several smaller MIPs, exploiting embedded decomposable structure. We adopt the following notational convention in our presentation: for vectors/matrices we use superscripts and for scalars/values we use subscripts. We consider a MIP of the form

$$\max \quad \sum_{i \in \mathcal{M}} c^i x^i \tag{1}$$

$$\text{s.t.} \quad x^i \in P_i, \qquad \forall i \in \mathcal{M} \tag{2}$$

$$\sum_{i \in \mathcal{M}} A^i x^i \leq b^1, \tag{3}$$

$$\sum_{i \in \mathcal{M}} D^i x^i \geq b^2, \tag{4}$$

where the input data is defined as follows: $b^1 \in \mathbb{R}^{m_1}_+$, $b^2 \in \mathbb{R}^{m_2}_+$, $m_1, m_2 \in \mathbb{Z}_+$, and $\mathcal{M} := \{1, \ldots, M\}$ is the index set of *blocks*. For each $i \in \mathcal{M}$, $c^i \in \mathbb{R}^{n_i}$, $n_i \in \mathbb{Z}_+ \setminus \{0\}$ and $A^i \in \mathbb{R}^{m_1 \times n_i}, D^i \in \mathbb{R}^{m_2 \times n_i}$. We assume that P_i is nonempty and bounded for all $i \in \mathcal{M}$, and includes any integrality requirements on the variables. For each block $i \in \mathcal{M}$, the problem has $m_1 + m_2$ coupling constraints, linking different blocks together. The linking constraints (3) correspond to limits on a set of resources shared among the blocks; we refer to b^1 as the *resource* vector. Similarly, the linking constraints (4) correspond to a set of requirements that must be satisfied, collectively, by the blocks; we refer to b^2 as the *requirement* vector.

Note that when $m_1 = m_2 = 0$ the problem is fully decomposable, so it can be solved by solving M integer programs (of smaller sizes). For convenience, in the remainder, we assume that $m_1 + m_2$ is relatively small, so the coefficient matrix is nearly decomposable.

A multi-objective optimization perspective lies at the heart of our approach, in which we think of the linking constraints as additional objectives along with the true objective. Given a fractional solution to the MIP's linear programming (LP) relaxation, the variables in each block satisfy

the constraints for that block and contribute to the objective function and to each of the linking constraints. We define a branching subproblem for each block, which seeks an integer (feasible) solution that "respects" the contribution of the current LP solution to the linking constraints, i.e., "consumes" at most as much in a $\leq$ constraint and "produces" at least as much in a $\geq$ constraint, and that maximizes the block's contribution to the objective function (for a maximization problem). If all branching subproblems yield (integer) solutions with the same contribution to the objective function as the current fractional solution, then these solutions must be optimal for the current branch-and-bound node and one can stop exploring it. Else, at least one of the branching subproblems provides a branching cut that eliminates the incumbent fractional solution.

In the following, we present the details of our approach (for a general MIP formulation). We then focus on two problems, a weighted set covering problem (Garey and Johnson, 2002) and a regionalized p-median facility location problem with assignment range constraints (Daskin and Tucker, 2018), to illustrate the implementation of the approach, to show its potential, and to introduce extensions that improve its computational efficiency. Our computational study demonstrates the benefits of decomposition branching: (1) it explores far fewer nodes and can be orders of magnitude faster than a state-of-the-art LP-based branch-and-bound algorithm (CPLEX) and (2) it significantly outperforms the implementation of Dantzig-Wolfe decomposition available in SCIP (Gamrath et al., 2020).

The remainder of the paper is organized as follows. In Section 2, we introduce the principal ideas of decomposition branching. In Section 3, we show how decomposition branching can be used to solve instances of the weighted set covering problem. In Section 4, we discuss enhancements that can result in significant performance improvements. In Section 5, we present the results of an extensive computational study in which we investigate the use of the suggested methodology to solve p-median and set-covering problems. Finally, in Section 6, we present some final remarks.

2 Decomposition Branching

The approach exploits an observation based on parameterization of the right-hand side vectors, b^1, b^2, into partitions between the blocks. Specifically, the problem may be rewritten as

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{M} c^i x^i & & \\
\text{s.t.} \quad & x^i \in P_i, & & \forall i = 1, \ldots, M \\
& A^i x^i \leq u^i, & & \forall i = 1, \ldots, M \\
& D^i x^i \geq \ell^i, & & \forall i = 1, \ldots, M \\
& \sum_{i=1}^{M} u^i \leq b^1, & & \\
& \sum_{i=1}^{M} \ell^i \geq b^2, & &
\end{aligned}
$$

where the variables, $u^i \in \mathbb{R}^{m_1}$ and $\ell^i \in \mathbb{R}^{m_2}$ for each $i = 1, \ldots, M$, are *decomposition vectors* that describe how the right-hand side vector is partitioned between blocks. This can be written

equivalently as a *resource-directive master problem (RDMP)*

$$\max \{\sum_{i=1}^{M} f_i(u^i, \ell^i) \ : \ \sum_{i=1}^{M} u^i \leq b^1, \sum_{i=1}^{M} \ell^i \geq b^2\},$$

with subproblems $SP_i(y^1, y^2)$ defined by

$$f_i(y^1, y^2) = \max \{c^i\xi \ : \ \xi \in P_i, \ A^i\xi \leq y^1, D^i\xi \geq y^2\},$$

for each $i = 1, \ldots, M$, where f_i is the value function (with respect to block i's component of the linking constraint) of the ith subproblem.

Our approach is predicated on the assumption that subproblems of the form of SP_i are relatively tractable to solve (perhaps even by a recursive application of this approach), and their solution is assumed to be a "black box" for the purpose of describing our approach. More specifically, we propose to solve the original problem by branch-and-bound, in which we introduce the following new branching rule, which exploits the decomposable structure in the problem. Suppose the LP relaxation at some node of the tree has non-integer solution $\hat{x} = (\hat{x}^1, \ldots, \hat{x}^M) \in \mathbb{R}^N$, where $N = \sum_{i=1}^{M} n_i$. Also, let $\hat{P}_i \subseteq P_i$ denote the feasible set for the ith subproblem at this node of the tree. For each $i = 1, \ldots, M$ for which $\hat{x}^i$ is non-integer (in some component that the MIP requires to be integer), solve the *branching subproblem*, $BSP_i(\hat{x}^i)$, given by

$$z_i^*(\hat{x}^i) = \max \{c^i\xi \ : \ \xi \in \hat{P}_i, \ A^i\xi \leq A^i\hat{x}^i, \ D^i\xi \geq D^i\hat{x}^i\}$$

to obtain solution ξ^{i*}. If $\hat{x}^i$ is integer, set $\xi^{i*} := \hat{x}^i$ and $z_i^*(\hat{x}^i) = c^i\xi^{i*}$. If it turns out that $z_i^*(\hat{x}^i) = c^i\hat{x}^i$ for every $i = 1, \ldots, M$, then it must be that $\xi^* = (\xi^{1*}, \ldots, \xi^{M*})$ is an optimal solution for the current node of the tree. Thus, the node is pruned by optimality. Otherwise, we may stop computing $BSP_i(\hat{x}^i)$ at the first i for which $z_i^*(\hat{x}^i) \neq c^i\hat{x}^i$, in which case $z_i^*(\hat{x}^i) < c^i\hat{x}^i$ (otherwise $\hat{x}$ is not optimal for the LP relaxation at this node of the tree). Suppose i is the first such branching subproblem. Then we propose the following branching rule:

$$c^i x^i \leq z_i^*(\hat{x}^i) \quad \vee \quad \bigvee_{j=1}^{m_1} (A_j^i x^i > A_j^i \hat{x}^i) \quad \vee \quad \bigvee_{j=1}^{m_2} (D_j^i x^i < D_j^i \hat{x}^i).$$

This gives a multi-way branch, with $m_1 + m_2 + 1$ child nodes, that cuts off $\hat{x}^i$. Observe that rather than branching on the values of the variables, we branch on the contributions of the blocks to the linking constraints. Furthermore, in one branch we use the contribution of a block to the objective function, which, hopefully, leads to greater improvements in the dual bounds.

The implementation of a strict inequality in the branching rule is expected to make use of a tolerance parameter, $\epsilon > 0$, or integer rounding if the value of the left-hand side in the strict inequality is known to be integer in any feasible solution. For example, to implement $A_j^i x^i > A_j^i \hat{x}^i$, add the constraint $A_j^i x^i \geq A_j^i \hat{x}^i + \epsilon$ to the formulation of $\hat{P}_i$, or, if $A_j^i x^i$ is known to be integer for all $x^i \in \hat{P}_i$, then add the constraint $A_j^i x^i \geq \lceil A_j^i \hat{x}^i + \epsilon \rceil$ instead.

Since this branching rule does not necessarily partition the node problem feasible set, it can be strengthened by ordering the branching constraints, and adding the opposite of prior constraints to each child node created. For example, when creating the child node for the branching constraint

$A^i_j x^i > A^i_j \hat{x}^i$, the constraints

$$c^i x^i > z^*_i(\hat{x}^i) \quad \wedge \quad \bigwedge_{k=1}^{j-1} (A^i_k x^i \leq A^i_k \hat{x}^i)$$

may also be added. This leads us to propose a slightly more careful expression of the branching subproblem, $\text{BSP}_i(\hat{x}^i)$, with double-sided bounds for each linking constraint and the objective value, which may be written as

$$z^*_i(\hat{x}^i) = \max \{c^i\xi : \xi \in \hat{P}_i,\ \underline{\alpha}^i \leq c^i x^i \leq \overline{\alpha}^i, \underline{\beta}^i \leq A^i\xi \leq \min\{\overline{\beta}^i, A^i\hat{x}^i\}, \overline{\mu}^i \geq D^i\xi \geq \max\{\underline{\mu}^i, D^i\hat{x}^i\}\}.$$

If $A^i_j \hat{x}^i = \overline{\beta}^i_j$ then the branch for $A^i_j x^i > A^i_j \hat{x}^i$ can simply be eliminated. Similarly, if $D^i_j \hat{x}^i = \underline{\mu}^i_j$ then the branch for $D^i_j x^i < D^i_j \hat{x}^i$ can be eliminated.

The connection to multi-objective optimization derives from the fact that it is not difficult to show that, without loss of generality, only solutions to the ith subproblem that are nondominated points of the multi-objective optimization problem (MOSP_i)

$$\begin{array}{l} \max c^i\xi,\ \min A^i_1\xi, \ldots, \min A^i_{m_1}\xi, \max D^i_1\xi, \ldots, \max D^i_{m_2}\xi \\ \text{s.t.}\ \ \xi \in P_i, \end{array}$$

need be considered, where A^i_j denotes the jth row of A^i and D^i_j denotes the jth row of D^i. This is shown formally by Bodur et al. (2016), who use it to develop a new, resource-directive, reformulation of integer programs having decomposable structure. The reformulation has a variable for each nondominated point of a subproblem. Bodur et al. (2016) develop a solution algorithm for their reformulation, in which a modified form of column generation is used at the first iteration to initialize an integer programming master problem, and the integer programming master problem is solved at each iteration thereafter.

2.1 Special cases: linking constraints with set packing or set covering structure

In the special case that some of the linking constraints take the form of either set packing or set covering constraints, in some block, then the out-degree of the branch-and-bound tree nodes at which this block is used for branching can be reduced through standard binary variable modeling tricks. One branch can be used to model the branches for all linking constraints in the set packing form, and one branch can be used to combine all constraints in the set covering form.

To be specific, in the case of set packing, where A^i is a binary matrix for each i and $b^1 = \mathbf{1}$ is the vector of all ones, suppose that for block i, it is known [1] that $A^i_j x^i \in \{0, 1\}$ for any $x^i \in P_i$, (so it can be guaranteed that $0 \leq A^i_j \hat{x}^i \leq 1$), for all $j = 1, \ldots, q$. Then, (assuming that $A^i_j \hat{x}^i < 1$ for all $j = 1, \ldots, q$, since otherwise the branch would be eliminated), the disjunction reads

$$\bigvee_{j=1}^{q} (A^i_j x^i > A^i_j \hat{x}^i) = \bigvee_{j=1}^{q} (A^i_j x^i \geq 1),$$

which can be modeled with the single constraint

$$\sum_{j=1}^{q} A^i_j x^i \geq 1.$$

[1] This can easily be guaranteed by adding the valid constraint $A^i x^i \leq \mathbf{1}$ to the description of P_i.

This replaces q of the child nodes by a single child node.

The situation for set covering is somewhat different, since $D_j^i \hat{x}^i$ may take values that are greater than $b_j^2 = 1$, indicating that the jth item is "over-covered" by the ith block. In such a case, since all values of $D_j^i x^i$ that are in excess of 1 affect the blocks other than i in the same way (making the jth covering constraint redundant for all other blocks), it seems unhelpful to require $D^i \xi \geq D_j^i \hat{x}^i$ in the branching subproblem. Instead, we suggest using the constraint $D^i \xi \geq \min\{1, D_j^i \hat{x}^i\}$ in the branching subproblem. Then (assuming $D_j^i \hat{x}^i > 0$ for all j) the disjunction becomes

$$\bigvee_{j=1}^{q} (D_j^i x^i < \min\{1, D_j^i \hat{x}^i\}) = \bigvee_{j=1}^{q} (D_j^i x^i \leq 0) = \bigvee_{j=1}^{q} (D_j^i x^i = 0). \tag{5}$$

When for some block i, the term $D_j^i x^i$ must take on binary values for all $j = 1, \ldots, q$, this disjunction can be modeled by the single linear inequality

$$\sum_{j=1}^{q} D_j^i x^i \leq q - 1,$$

replacing q of the child nodes by a single child node. In general, when $D_j^i x^i$ may take on integer values greater than 1, modeling the disjunction linearly is less straightforward. We discuss one approach to handling this in the case of the Weighted Set Covering problem in Section 4.1.1.

Replacing the multi-term disjunction by a single inequality in the manner described above for set packing and set covering allows a multi-way branch to be replaced by a standard, two-way, branch. However this may come at a cost in the strength of the formulation: the LP relaxation bound implied by the two-way branch may be worse than that from the multi-way branch.

3 Decomposition Branching for the Weighted Set Covering Problem

Although often stated in terms of sets, e.g., Karp (1972), here we state the set covering problem using a graph, as this makes the discussions about decomposition more transparent. We consider an undirected graph $G = (V, E)$, with a weight, w_v, for each vertex $v \in V$. A vertex $v \in V$, can be covered by itself or any other vertex $u \in V$, such that $(u, v) \in E$. For $v \in V$, we let $\delta(v) = \{v\} \cup \{u \in V : \{u, v\} \in E\}$ denote v and its neighbors. For $V' \subseteq V$, we let $\mathcal{C}(V') = \{v \in V : v \in V' \text{ or } \{u, v\} \in E \text{ with } u \in V'\} = \bigcup_{u \in V'} \delta(u)$ denote the set of vertices that are covered by V'. A set of vertices $V' \subseteq V$ is called a *cover* (of G) if $\mathcal{C}(V') = V$. The weight $W(V')$ of a cover V' is $W(V') = \sum_{v \in V'} w_v$. The Weighted Set Covering problem (WSC) is to find a minimum weight cover of a graph.

Consider the following integer programming formulation of WSC:

$$\min \sum_{v \in V} w_v x_v \tag{6}$$

$$\text{s.t.} \sum_{u \in \delta(v)} x_u \geq 1, \qquad \forall v \in V \tag{7}$$

$$x_v \in \{0, 1\}, \qquad \forall v \in V. \tag{8}$$

Now consider a partition of the vertices, $V_1 \cup V_2 \cup \ldots V_M = V$. For each $i \in \mathcal{M} = \{1, 2, \ldots, M\}$, the variables x_v for $v \in V_i$ form the variables of block i; each set V_i corresponds to block i. We define $U_i = \{v \in V_i : \delta(v) \subseteq V_i\}$ for each $i \in \mathcal{M}$ to be the set of vertices in V_i that can only be covered by vertices in the same block. Note that for some i, it may be that $U_i = \emptyset$. The feasible set for block $i \in \mathcal{M}$ is defined by the coverage constraints for any vertex in U_i, i.e., by

$$P_i = \{x \in \{0,1\}^{V_i} : \sum_{u \in \delta(v)} x_u \geq 1, \ \forall v \in U_i\}.$$

The linking constraints are the coverage constraints for any vertex in $\overline{V} = V \setminus (\bigcup_{i \in \mathcal{M}} U_i)$. A vertex in $\overline{V}$ is called a *shared* vertex. Define

$$N_i = \{v \in V \setminus U_i \ : \ V_i \cap \delta(v) \neq \emptyset\} = \mathcal{C}(V_i) \setminus U_i$$

to be the set of vertices that may be covered by some vertex in V_i, but that may also be covered by a vertex in V_j for some $j \neq i$. For a given solution $\hat{x}$ to some LP relaxation, the contribution to the linking constraint of a shared vertex $v \in N_i$ by block i is

$$\sum_{u \in V_i : v \in \delta(u)} \hat{x}_u,$$

and, thus, the corresponding constraint in the decomposition branching subproblem is

$$\sum_{u \in V_i : v \in \delta(u)} x_u \geq \Big\lceil \sum_{u \in V_i : v \in \delta(u)} \hat{x}_u \Big\rceil.$$

To illustrate, consider an example in which, for some block i, $N_i = \{v_1, v_2, v_3, v_4\}$ and suppose that $\hat{x}$, the initial solution to the LP relaxation, has

$$\sum_{v \in V_i : v_1 \in \delta(v)} \hat{x}_v = 1, \quad \sum_{v \in V_i : v_2 \in \delta(v)} \hat{x}_v = 2/3, \quad \sum_{v \in V_i : v_3 \in \delta(v)} \hat{x}_v = 2/3 \qquad \text{and} \qquad \sum_{v \in V_i : v_4 \in \delta(v)} \hat{x}_v = 0.$$

Then the branching subproblem is

$$z_i^*(\hat{x}) = \min\{\sum_{v \in V_i} w_v x_v : x \in P_i, \ \sum_{v \in V_i : v_k \in \delta(v)} x_v \geq 1, \ k = 1, 2, 3\},$$

since the left-hand side of each covering expression must be integer and the right-hand sides in the constraints for v_2 and v_2 can be set to $\lceil 2/3 \rceil = 1$. If $z_i^*(\hat{x}) = \sum_{v \in V_i} w_v \hat{x}_v$, then no branching is needed from this subproblem; its integer solution could replace $\hat{x}_v$ for $v \in V_i$ to form an alternative optimal solution to the current LP relaxation, having fewer fractional variables. Otherwise, it must be that $z_i^*(\hat{x}) > \sum_{v \in V_i} w_v \hat{x}_v$, and the branching disjunction would be

$$\sum_{v \in V_i} w_v x_v \geq z_i^*(\hat{x}) \ \bigvee \ \sum_{v \in V_i : v_1 \in \delta(v)} x_v = 0 \ \bigvee \ \sum_{v \in V_i : v_2 \in \delta(v)} x_v = 0 \ \bigvee \ \sum_{v \in V_i : v_3 \in \delta(v)} x_v = 0.$$

One way to ensure that these branches partition the solution space (and so form a *polychotomy*) is to add the constraints $\sum_{v \in V_i : v_k \in \delta(v)} x_v \geq 1$ for all $k = 1, 2, 3$ to the first branch, add constraint $\sum_{v \in V_i, v_1 \in \delta(v)} x_v \geq 1$ to the latter two branches and add $\sum_{v \in V_i, v_2 \in \delta(v)} x_v \geq 1$ to the last branch.

4 Enhancements

Decomposition branching repeatedly solves branching subproblems to obtain branching constraints that cut off fractional solutions. Therefore, in order to improve the computational efficiency, it is critical to generate these branching constraints as efficiently as possible and to obtain branching constraints that reduce the size of the search tree as much as possible. In this section, we discuss enhancements that seek to achieve this.

4.1 Constraint aggregation

As has already been noted, decomposition branching is likely to be polychotomous, rather than the usual dichotomous branching based on a single variable. For example, in the case of WSC, after solving the branching subproblem for block $i \in \mathcal{M}$, decomposition branching may create up to $|N_i|+1$ branches. Here we introduce an enhancement that uses constraint aggregation to reduce the number of branches created at a node.

Constraint aggregation can be accomplished by designing nonnegative $k_1 \times m_1$ matrices, S^i, for each $i \in \mathcal{M}$, with $k_1 \leq m_1$, and nonnegative $k_2 \times m_2$ matrices, T^i, for each i, with $k_2 \leq m_2$. We call S^i (and T^i) *aggregation matrices*. Then the *aggregated* branching subproblem for i is

$$\alpha_i^*(\hat{x}^i) = \max \{c^i\xi \,:\, \xi \in \hat{P}_i,\ S^iA^i\xi \leq S^iA^i\hat{x}^i, T^iD^i\xi \geq T^iD^i\hat{x}^i\}.$$

Choosing S^i to be the $m_1 \times m_1$ identity matrix would yield the original (first) set of constraints (similarly for T^i). At the other extreme, taking S^i to be the $1 \times m_1$ matrix of all ones would yield a single constraint in place of the (first) set of constraints. Some columns of S^i (or T^i) may be zero, allowing some of the linking constraints to be ignored altogether. Any surrogate relaxation suffices. Of course, it is also possible, but not necessary, to use the same aggregation matrices for all i.

Consider the situation when the above aggregated branching subproblem is solved to obtain solution ξ^{i*}. If $A^i\xi^{i*} \leq A^i\hat{x}^i$ and $D^i\xi^{i*} \geq D^i\hat{x}^i$ and $\alpha_i^*(\hat{x}^i) = c^i\hat{x}^i$ for *every* $i = 1, \ldots, M$, then it must be that $\xi^* = (\xi^{*1}, \ldots, \xi^{*M})$ is an optimal solution for the current node of the tree. Thus, the node is pruned by optimality. Otherwise, select i for which $\alpha_i^*(\hat{x}^i) < c^i\hat{x}^i$. If there is no such i, then update one or more aggregation matrices so that the current subproblem solution is no longer feasible, and re-solve the subproblem. We discuss possible update procedures in more detail below. Here, we observe that – provided the update procedure has reasonable properties, e.g., after a finite number of updates, the aggregation matrix becomes the identity matrix – this process must eventually either (i) prune the node by optimality (construct a complete integer feasible solution with the same objective value as that of the LP relaxation), or (ii) reach a situation in which $\alpha_i^*(\hat{x}^i) < c^i\hat{x}^i$ for some i. Then the branching rule we suggest is:

$$(c^ix^i \leq \alpha_i^*(\hat{x}^i)) \quad \vee \quad \bigvee_{j=1}^{k_1}((S^iA^i)_jx^i > (S^iA^i)_j\hat{x}^i) \quad \vee \quad \bigvee_{j=1}^{k_2}((T^iD^i)_jx^i < (T^iD^i)_j\hat{x}^i).$$

This gives a multi-way branch, with $k_1 + k_2 + 1$ child nodes, that cuts off $\hat{x}^i$.

The most straightforward way to update an aggregation matrix is as follows. Suppose the jth original constraint is violated by the current aggregated subproblem solution. For example, it may be that $A^i_j\xi^{*i} > A^i_j\hat{x}^i$. Then replace the jth column in the aggregation matrix by a column of

zeros, and add a new row, consisting of the jth unit vector in $\mathbb{R}^{m_1}$. The effect of this operation is to remove the linking constraint j from the aggregation, and include it as a separate, individual, constraint. This could be done for some, or all, violated constraints, adding a new row for each. A more conservative approach would be to consider adding a single new constraint that aggregates some of the violated constraints: to add an aggregation of constraints indexed by J, for each $j \in J$, replace the jth column in the aggregation matrix by a column of zeros, and then add a single row to it, consisting of the sum of the jth unit vectors over all $j \in J$.

We next show how constraint aggregation ideas can be used for WSC.

4.1.1 Constraint aggregation for weighted set covering

For WSC, one natural approach to constraint aggregation would be to select a subset $S \subseteq \overline{V}$ and replace the linking constraint $\sum_{v \in \delta(u)} x_v \geq 1$ for each $u \in S$ by the aggregation

$$\sum_{u \in S} \sum_{v \in \delta(u)} x_v \geq |S|. \tag{9}$$

Note that we intend this replacement to occur only in the branching subproblem, not in the master problem: the formulation of the LP relaxation is unchanged, except for the constraints to be introduced by branching. Constraint (9) corresponds to taking the aggregation matrix T^i to be the $1 \times |\overline{V}|$ binary matrix that is the indicator vector of S, i.e., with $T^i_v = 1$ if $v \in S$ and $T^i_v = 0$ otherwise.

Aggregation can lead to a relaxation of the original constraints; aggregation causes information loss. In the aggregation (9), the distinction between over-covering one vertex and covering more than one vertex may be lost. For example, consider the part of a WSC instance shown in Figure 1. Here block 1, defined by vertex set V_1, contains vertices $\{1, 2, 3\}$ and does not contain vertex 4. The two vertices $\{2, 4\} \subseteq N_1$ are shared vertices, with $\delta(2) \cap V_1 = \{1, 2\}$ and $\delta(4) \cap V_1 = \{3\}$. Consider the aggregation (9) in this example with $S = \{2, 4\}$.

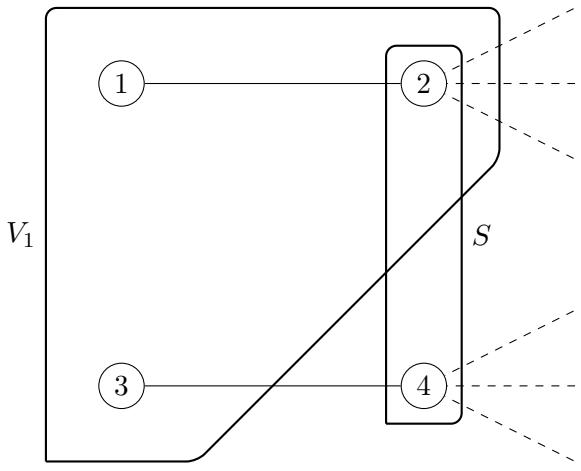


Figure 1: An example illustrating information loss from the aggregation (9).

In the unaggregated multiobjective model for block 1, the objectives $\max(x_1 + x_2)$ and $\max x_3$ appear, corresponding to covering vertices 2 and 4 respectively. If either of these objectives has value

at least 1 in some solution, then block 1 covers the corresponding vertex in that solution. In the aggregated model, these two objectives are replaced by the single objective $\max(x_1+x_2+x_3)$. This objective takes value 2 for a solution with $x_1 = 1, x_2 = 1, x_3 = 0$, in which only vertex 2, but not vertex 4, is covered by block 1. The aggregated objective cannot distinguish between this case and the case of a solution in which both vertices in S are covered by V_1, such as $x_1 = 1, x_2 = 0, x_3 = 1$. In other words, the aggregated objective does not accurately count the number of vertices in S that are covered by V_1, due to the possibility that some vertices may be over-covered.

A simple extension to the WSC model can rectify this information loss due to aggregation: introduce binary variables ℓ_v^i for each block i and shared vertex $v \in \overline{V}$, to indicate whether or not block i is "responsible" for covering vertex v. These variables now become part of the block i subproblem. The variables for block i now consist of x_v for each $v \in V_i$ and ℓ_v^i for each $v \in N_i$, with the following feasible set (in place of P_i):

$$Q_i = \{(x, \ell^i) \in \{0,1\}^{V_i} \times \{0,1\}^{N_i} : \sum_{v \in \delta(u)} x_v \geq 1,\ \forall u \in U_i \text{ and } \sum_{v \in V_i \cap \delta(u)} x_v \geq \ell_u^i,\ \forall u \in N_i\}.$$

The master problem linking constraints may be expressed in terms of the new binary variables as

$$\sum_{i \in \mathcal{M}, v \in N_i} \ell_v^i \geq 1, \qquad \forall v \in \overline{V}$$

and aggregated over a set $S \subseteq \overline{V}$, allowing (9) to be replaced by

$$\sum_{v \in S} \sum_{i \in \mathcal{M}, v \in N_i} \ell_v^i \geq |S|. \tag{10}$$

Then, in the multi-objective form of the branching subproblem for block i, the objective

$$\max \sum_{v \in N_i \cap S} \ell_v^i \tag{11}$$

appears, in addition to the objective minimizing the total weight of vertices selected. The above objective seeks to maximize the number of vertices in S that are covered by block i.

Of course, more than one subset of linking constraints may be aggregated. For example, a heuristic could be used to partition $\overline{V}$ into a desired number of subsets. However, each such subset would add one to the number of child nodes created at each branch-and-bound tree node by the resulting decomposition branching rule, increasing the tree nodes' degrees.

We illustrate the WSC decomposition branching rule that would result from constraint aggregation over only a single subset of linking constraints, S, as follows. First, recall that the branching subproblem will seek to do "at least as well" as the fractional LP solution in respect to each of the objectives derived from a linking constraint. Thus, given a fractional LP solution, $\hat{x}$, and a block i, we need to determine "how well" $(\hat{x}_v)_{v \in V_i}$ is covering vertices in S; we need to evaluate the contribution of block i to the left-hand side of constraint (10) and thus somehow need to determine values of ℓ_u^i for $u \in N_i \cap S$ that correspond to $\hat{x}$. One simple expedient is to include the ℓ_v^i variables in the master problem in the obvious way, and allow them to be set to values, $\hat{\ell}_v^i$, by the LP solver. This has an added advantage: such variables are required in the master problem in order to model the branching constraints that result from constraint aggregation, as shall become obvious below. In what follows, we assume that $\hat{\ell}_v^i$ values are available, found by this or some other means.

Consider an example in which, for some block i and set $S \subseteq \overline{V}$, $N_i \cap S = \{v_1, v_2, v_3, v_4\}$ and suppose that $\hat{\ell}^i$, derived from the initial solution to the LP relaxation, has

$$\sum_{k=1}^{4} \hat{\ell}^i_{v_k} = 9/4.$$

Now the branching subproblem is

$$z_i^*(\hat{x}, \hat{\ell}) = \min\{\sum_{v \in V_i} w_v x_v : (x, \ell^i) \in Q_i,\ \sum_{k=1}^{4} \ell^i_{v_k} \geq \left\lceil \sum_{k=1}^{4} \hat{\ell}^i_{v_k} \right\rceil = 3\}.$$

If $z_i^*(\hat{x}, \hat{\ell}) > \sum_{v \in V_i} w_v \hat{x}_v$, then the branching rule would be

$$\sum_{v \in V_i} w_v x_v \geq z_i^*(\hat{x}, \hat{\ell}) \quad \bigvee \quad \sum_{k=1}^{4} \ell^i_{v_k} \leq 2.$$

One way to ensure that these branches partition the solution space (and so form a dichotomy) is to add the constraint $\sum_{k=1}^{4} \ell^i_{v_k} \geq 3$ to the first branch. Note that even if $\sum_{k=1}^{4} \hat{\ell}^i_{v_k}$ is already integer, for example, it takes the value 3, then, since $z_i^*(\hat{x}, \hat{\ell}) > \sum_{v \in V_i} w_v \hat{x}_v$, this branching rule cuts off the current LP solution.

Clearly, branching constraints such as $\sum_{k=1}^{4} \ell^i_{v_k} \leq 2$ or $\sum_{k=1}^{4} \ell^i_{v_k} \geq 3$ cannot be added to the master problem unless the ℓ variables are included in it.

If $z_i^*(\hat{x}, \hat{\ell}) \leq \sum_{v \in V_i} w_v \hat{x}_v$, then, unlike the case without aggregation, branching from this subproblem may be needed. There is still the possibility that the subproblem solution has not covered some particular, individual, shared vertex as well as the fractional solution did. In the next section, we describe the specific strategies we developed to address this issue, and to decide which subset(s) of linking constraints to aggregate, when.

Introducing the binary ℓ^i_v variables provides an advantage quite apart from their use for constraint aggregation. Since ℓ^i_v models $\min\{1, \sum_{u \in V_i \cap \delta(v)} x_u\}$, it allows the multi-term disjunction in the form of (5) to be modeled as a single linear inequality, even in the general case that $\sum_{u \in V_i \cap \delta(v)} x_u$ can be an integer greater than 1. If F^i denotes the set of (shared) vertices in N_i that are required to be covered in the current branching subproblem i, then the disjunction

$$\bigvee_{v \in F^i} (\sum_{u \in V_i \cap \delta(v)} x_u = 0) \qquad \text{is equivalent to} \qquad \bigvee_{v \in F^i} (\ell^i_v = 0),$$

which can be modeled by the single linear inequality

$$\sum_{v \in F^i} \ell^i_v \leq |F^i| - 1.$$

4.1.2 A specific decomposition branching algorithm for WSC

Pseudocode for a decomposition branching algorithm with constraint aggregation for solving WSC by branch-and-bound is presented in Algorithm 1. The algorithm is run at a node of the branch-and-bound tree, after the master problem LP relaxation has been solved and found to be feasible,

with one or more variables taking on non-integer values in the solution. Algorithm 1 takes as input the current fractional LP solution, consisting of the pair $\hat{x}$ and $\hat{\ell}$. It is assumed that all branching constraints and corresponding bounds on variables applied to reach the current node have been included in the appropriate branching subproblem; we signal this by writing $\hat{Q}_i$ in place of Q_i.

Our strategy is to apply constraint aggregation only to a block i for which $\hat{\ell}^i$ contains fractional values. In this case, we first aggregate *all* linking constraints, and so take $S = \overline{V}$. For an individual block, i, this is equivalent to aggregating all linking constraints over N_i, and results in the branching subproblem shown in lines 5 and 10 of Algorithm 1. If this fails to produce a branch, which occurs if $z_i^*(\hat{x}, \hat{\ell}) \leq \sum_{v \in V_i} w_v \hat{x}_v$, then we take S to be a singleton, consisting of one u for which $\hat{\ell}_u^i$ is fractional. The resulting subproblem is given at line 17, and must produce a branch.

If for some block i, $\hat{\ell}^i$ contains only the values 0 or 1, but $\hat{x}_v$ is fractional for some $v \in V_i$, we revert to the unaggregated case, in which block i is required to cover *all* shared vertices that it covers in the fractional solution. These vertices are indexed by the set F^i, calculated at line 22. The resulting branching subproblem is shown in line 23. The one additional strategy we employ is to maintain a dichotomous branching rule, by modeling the multiway branches

$$\bigvee_{v \in F^i} (\ell_v^i = 0)$$

with the single constraint $\sum_{v \in F^i} \ell_v^i \leq |F^i| - 1$, along the same lines as discussed for the special case of set covering in Section 2.1 and for the general case at the end of Section 4.1.1.

4.2 Re-using branching subproblem solutions

This enhancement is motivated by the observation that the same branching subproblem, or one with slightly different parameters, can arise at different nodes of the search tree. Thus, it is beneficial to store (relevant) information gathered during the solution of a branching subproblem and re-use it whenever possible during the search. To implement this enhancement, we define a set of *states* for a branching subproblem and maintain the best known solution for each of the states. Each time we solve a branching subproblem, we use the solution obtained to update, if appropriate, the best known solution for one or more of the states.

Recall that at a particular node of the search tree, the branching subproblem $BSP_i(\hat{x}^i)$, for $i \in \mathcal{M}$, can be written as

$$\max \{c^i \xi : \xi \in P_i,\ \underline{\alpha}^i \leq c^i \xi^i \leq \overline{\alpha}^i, \underline{\beta}^i \leq A^i \xi \leq \min\{\overline{\beta}^i, A^i \hat{x}^i\}, \overline{\mu}^i \geq D^i \xi \geq \max\{\underline{\mu}^i, D^i \hat{x}^i\}\},$$

where the branching constraints used to reach the current node of the search tree are encoded in the bounds $\underline{\alpha}^i, \overline{\alpha}^i, \underline{\beta}^i, \overline{\beta}^i, \underline{\mu}^i$, and $\overline{\mu}^i$. Therefore, a state of a branching subproblem can be defined as the set of intervals $[\underline{\alpha}^i, \overline{\alpha}^i], [\underline{\beta}^i, \min\{\overline{\beta}^i, A^i \hat{x}^i\}]$, and $[\underline{\mu}^i, \max\{\underline{\mu}^i, D^i \hat{x}^i\}]$. This state definition may or may not be practical, because the number of states may be too large to maintain and search efficiently.

However, alternative (aggregated) state definitions may be possible. For example, when solving instances of WSC, we use the following two state definitions for the branching subproblem for a given $i \in \mathcal{M}$.

- A Type 1 state associated with branching subproblem i is defined by an integer $s \in \{0, \ldots, |N_i|\}$, which represents the (minimum) number of shared vertices that have to be covered in any solution to the branching subproblem.

Algorithm 1: DB for WSC with constraints aggregation

Input: $\hat{x}, \hat{\ell}$

1 **foreach** $i \in \mathcal{M}$ with $\hat{x}_v$ fractional for some $v \in V_i$ **do**

2 **if** $\hat{\ell}^i_v$ is fractional for some $v \in N_i$ **then**

3 $\sigma_i := \sum_{v \in N_i} \hat{\ell}^i_v$

4 **if** σ_i *is fractional* **then**

5 Calculate $z^*_i(\hat{x}, \hat{\ell}) = \min\{\sum_{v \in V_i} w_v x_v : (x, \ell^i) \in \hat{Q}_i,\ \sum_{v \in N_i} \ell^i_v \geq \lceil \sigma_i \rceil\}$

6 $branch_1 : \sum_{v \in N_i} \ell^i_v \geq \lceil \sigma_i \rceil \ \bigwedge\ \sum_{v \in V_i} w_v x_v \geq z^*_i(\hat{x}, \hat{\ell})$

7 $branch_2 : \sum_{v \in N_i} \ell^i_v \leq \lfloor \sigma_i \rfloor$

8 **return**

9 **else**

10 Calculate $z^*_i(\hat{x}, \hat{\ell}) = \min\{\sum_{v \in V_i} w_v x_v : (x, \ell^i) \in \hat{Q}_i,\ \sum_{v \in N_i} \ell^i_v \geq \sigma_i\}$

11 **if** $z^*_i(\hat{x}, \hat{\ell}) > \sum_{v \in V_i} w_v \hat{x}_v$ **then**

12 $branch_1 : \sum_{v \in N_i} \ell^i_v \geq \sigma_i \ \bigwedge\ \sum_{v \in V_i} w_v x_v \geq z^*_i(\hat{x}, \hat{\ell})$

13 $branch_2 : \sum_{v \in N_i} \ell^i_v \leq \sigma_i - 1$

14 **return**

15 **else**

16 Choose some $u \in N^i$ with $\hat{\ell}^i_u$ fractional

17 Calculate $z^*_i(\hat{x}, \hat{\ell}) = \min\{\sum_{v \in V_i} w_v x_v : (x, \ell^i) \in \hat{Q}_i,\ \ell^i_u = 1\}$

18 $branch_1 : \ell^i_u = 1 \ \bigwedge\ \sum_{v \in V_i} w_v x_v \geq z^*_i(\hat{x}, \hat{\ell})$

19 $branch_2 : \ell^i_u = 0$

20 **return**

21 **else**

22 $F^i := \{v \in N_i : \hat{\ell}^i_v = 1\}$

23 Calculate $z^*_i(\hat{x}, \hat{\ell}) = \min\{\sum_{v \in V_i} w_v x_v : (x, \ell^i) \in \hat{Q}_i,\ \ell^i_v = 1,\ \forall v \in F^i\}$

24 **if** $z^*_i(\hat{x}, \hat{\ell}) > \sum_{v \in V_i} w_v \hat{x}_v$ **then**

25 $branch_1 : \ell^i_v = 1,\ \forall v \in F^i \ \bigwedge\ \sum_{v \in V_i} w_v x_v \geq z^*_i(\hat{x}, \hat{\ell})$

26 $branch_2 : \sum_{v \in F^i} \ell^i_v \leq |F^i| - 1$

27 **return**

28 **else**

29 Replace $(\hat{x}_v)_{v \in V_i}$ with the corresponding part of the solution of the branching subproblem solved at line 23

- A Type 2 state associated with branching subproblem i is defined by a set $S \subset N_i$, where S represents a subset of shared vertices, every one of which must be covered in any solution to the branching subproblem.

In Algorithm 1, the value stored for a Type 1 state may be used to avoid solving the branching subproblem in lines 5 and 10, and the value stored for a Type 2 state may be used to avoid solving the branching subproblem in lines 17 and 23.

4.3 Early termination of branching subproblem solving

To generate a branching constraint, it is not always necessary to solve a subproblem to optimality. Instead of using the optimal solution value, an upper bound on the optimal solution value (for a problem in maximization form) can be used to generate a branching constraint, if it cuts off the fractional solution to the master problem. Thus, one can consider terminating solution of a branching subproblem as soon as the value of the upper bound falls below the value of the fractional solution to the master problem. Clearly, when solving a branching subproblem to optimality is difficult, i.e., time consuming, early termination (especially early in the search process) may reduce overall solution time. Early termination can be combined with re-use of branching subproblem solutions (as discussed above) by storing the best known upper bound value for a state.

4.4 Branching subproblem ordering

For a given fractional solution to the master problem, decomposition branching solves the subproblems to find a constraint that eliminates the current fractional solution to the master problem (or to construct a feasible solution with the same objective function value). Thus, the order in which the branching subproblems are solved affects the search (and thus the solution time).

Certain subproblem orderings may work better for different problem classes and instances. For example, if solution times can differ significantly between branching subproblems, then it may be beneficial to solve the branching subproblems in nondecreasing order of some "estimate" of their solution time. Another consideration may be the likelihood that a branching constraint resulting from the solution to a subproblem will improve the dual bound.

The latter, for example, has motivated the subproblem ordering used when solving WSC instances. Recall that at a node in the search tree, because of earlier branching decisions, an interval bounding the objective function value of subproblem i is known. For WSC, we process the branching subproblems in nondecreasing order of $\underline{\alpha}^i$, a lower bound on the value of branching subproblem i, in the hope that solutions to subproblems that have a larger contribution to the objective function will have a larger impact on the dual bound. Our computational results show that even such a simple (and possibly naive) ordering rule can significantly improve the performance of the suggested approach.

We emphasize that our aim here is only to provide an example that shows that even simple ordering rules can enhance the computational performance. Clearly, more sophisticated ordering rules can be developed, for example using integer infeasibilities, that may perform better. The impact of subproblem ordering on solution time is somewhat similar to the impact of branching variable selection in traditional branch-and-bound, for which many different options have been explored, e.g., pseudo-cost branching, strong branching, and realibility branching.

5 Computational experiments

In order to evaluate the effectiveness of decomposition branching (and the enhancements presented to improve its computational efficiency), we have conducted a comprehensive numerical study with instances from two problems: WSC, which we have used above to illustrate many aspects of decomposition branching, and FLAR, a regionalized form of the p-median facility location problem with assignment range constraints (Daskin and Tucker, 2018). The choice to use the latter is, in part, motivated by the fact that real-life instances were available, but also because p-median facility location is another fundamental problem in discrete optimization and this particular variant is especially hard to solve.

We implemented our algorithms in Java using IBM ILOG CPLEX Studio 12.10. We use callbacks to implement the proposed branching scheme. When CPLEX has solved the linear programming relaxation at a node and is ready to branch, we take over control and (1) solve the branching subproblems (using CPLEX in default settings) and (2) either produce a feasible solution with the same objective function value, which is then passed to CPLEX and results in the node being pruned, or define the child nodes by adding branching constraints, after which control is given back to CPLEX. All experiments were run on a 64-bit Linux operated workstation with an Intel Xeon Gold 6134 processor at 3.20 GHz and 96 GB of RAM.

In addition to comparing the performance of our decomposition branching algorithms to the performance of CPLEX (with default settings), we also compare it to the performance of GCG (Gamrath, 2010), the Dantzig-Wolfe decomposition implementation available in the SCIP Optimization Suite 7.0 (Gamrath et al., 2020). We compile the SCIP Optimization Suite with CPLEX as the LP solver and use GCG with its default settings, letting the algorithm automatically choose how many threads to use. For both WSC and FLAR instances, we inform the solver about the decomposition structure by indicating which are the linking constraints (i.e., (14) for the WSC and (18) for the FLAR instances). We found that letting the solver automatically detect/decide the decomposition structure resulted in significantly worse performance (even for FLAR instances, where there is only a single linking constraint).

5.1 Weighted set covering

5.1.1 Instances

Each instance has $M \times n$ nodes, where M is the number of blocks and n the number of nodes in a block, so $V = V_1 \cup V_2, \ldots \cup V_M$ with $|V_i| = n$ for all $i = 1, \ldots, M$. The node weights are drawn from the set $\{1, \ldots, W\}$ with equal probability. Each block has $\lfloor \rho n(n-1)/2 \rfloor$ edges between its nodes, where $\rho \in (0, 1]$ represents the block density. Furthermore, $s \times M$ edges connect different blocks, where $s \in \{1, \ldots, n\}$ is a parameter representing the block connectivity. More specifically, the edge set E is generated as follows.

- For each block, we generate a random Hamiltonian cycle and add its n edges to E.
- For each block, we randomly pick distinct pairs of nodes and add the edge connecting them to E until we have added $\lfloor \rho n(n-1)/2 \rfloor - n$ distinct edges (duplicates are skipped).
- For each block $i = 1, \ldots, M$, we randomly pick a set $C_i \subseteq V_i$ of s nodes. Then for each $i = 1, \ldots, M$ and for each node in C_i, we randomly pick a node in $V_1 \setminus C_1 \cup \ldots \cup V_{i-1} \setminus C_{i-1} \cup$

$V_{i+1} \setminus C_{i+1} \cup \ldots \cup V_M \setminus C_M$ and add the edge connecting the two selected nodes to E, again ensuring no duplicate edges are created.

5.1.2 Implementation

The master problem at a node is

$$\min \quad \sum_{v \in V} w_v x_v \tag{12}$$

$$\text{s.t.} \quad ((x_v)_{v \in V_i}, \ell^i) \in Q_i, \qquad \forall i \in \mathcal{M} \tag{13}$$

$$\sum_{i \in \mathcal{M}, v \in N_i} \ell^i_v = 1, \qquad \forall v \in \overline{V} \tag{14}$$

together with all branching constraints added to create the node, where

$$Q_i = \{(x, \ell^i) \in \{0,1\}^{V_i} \times \{0,1\}^{N_i} : \sum_{v \in \delta(u)} x_v \geq 1,\ \forall u \in U_i \text{ and } \sum_{v \in V_i \cap \delta(u)} x_v \geq \ell^i_u,\ \forall u \in N_i\}.$$

What we will refer to as the *basic* implementation follows Algorithm 1 and uses the two forms of branching subproblem solution re-use discussed in Section 4.2. Preliminary experiments have shown that for these instances, the branching subproblems solve quite fast and an optimal solution is found quite early in the search. However, the (global) lower bound improves only slowly. As a result, the order in which the branching subproblems are solved has a significant impact on the overall performance, and always processing the subproblems in nonincreasing order of the best known bound on the objective function value, as suggested in Section 4.4, provided the best results. What we will refer to as the *enhanced* version uses this subproblem ordering. Because the branching subproblems solve quickly, there is no need to incorporate early termination ideas.

5.1.3 Experiments

Among the instance parameters, M and s are the most relevant when it comes to determining the potential of decomposition branching. As M increases and as s decreases, the benefit of decomposition branching should increase. In our experiments, we set $n = 150$ and $\rho = 0.3$, consider all combinations of $W \in \{3, 5, 7\}$, $M \in \{4, 6, 8\}$ and $s \in \{2, 3, 4\}$, generate five random instances for each combination, and impose a run time limit of 7200 seconds.

5.1.4 Analysis

The results of our experiments can be found in Table 1, where we compare the performance of decomposition branching (DB) and its enhancement with branching subproblem ordering (DBE) to the performance of commercial solver CPLEX and the Danzig-Wolfe (DW) decomposition implementation in SCIP. In the table, for each algorithm, we report the solution time and optimality gap averages along with the number of times an optimal solution is found (and proved to be optimal) for the five random instances. For the solution times and optimality gaps, we report both the arithmetic and geometric averages under the columns headed "mean" and "g.mean", respectively. When computing optimality gap averages, we consider only solutions with positive gaps, which means that they might be taken over different instances.

Table 1: Results for WSC instances.

			CPLEX					DB					DBE					DW				
			time		gap (%)			time		gap (%)			time		gap (%)			time		gap (%)		
W	M	s	mean	g.mean	mean	g.mean	#opt	mean	g.mean	mean	g.mean	#opt	mean	g.mean	mean	g.mean	#opt	mean	g.mean	mean	g.mean	#opt
3	4	2	7230.6	7230.6	23.0	22.9	0	350.0	273.3	0.0	0.0	5	16	15.0	0.0	0.0	5	579.9	528.9	0.0	0.0	5
		3	7250.8	7250.8	23.8	23.5	0	5813.4	4859.3	11.2	11.1	2	37.6	24.5	0.0	0.0	5	856.3	717.9	0.0	0.0	5
		4	7241.2	7241.2	24.6	24.4	0	1602.0	781.9	0.0	0.0	5	66.4	31.4	0.0	0.0	5	2799.3	1928.3	10.5	10.5	4
	6	2	7237.0	7237.0	27.6	27.5	0	578.4	503.3	0.0	0.0	5	63.6	60.7	0.0	0.0	5	841.9	760.5	0.0	0.0	5
		3	7244.6	7244.6	27.8	27.7	0	6021.8	5018.8	10.0	8.1	1	123.8	54.7	0.0	0.0	5	2521.0	1957.3	0.0	0.0	5
		4	7251.4	7251.4	28.7	28.6	0	7200.0	7200.0	14.6	14.5	0	229	88.6	0.0	0.0	5	4601.7	4236.0	0.0	0.0	5
	8	2	7223.0	7223.0	30.2	30.2	0	7224.2	7224.1	18.9	15.4	0	106	72.3	0.0	0.0	5	2240.8	1957.5	0.0	0.0	5
		3	7227.0	7227.0	30.6	30.6	0	7282.2	7281.3	33.5	33.2	0	425.8	228.8	0.0	0.0	5	4545.7	4084.4	0.0	0.0	5
		4	6905.2	6872.6	31.2	31.2	0	7232.4	7232.4	34.9	34.7	0	4857.8	2135.5	1.8	1.8	2	5841.6	5499.2	2.7	2.6	3
5	4	2	5579.8	4907.3	10.2	10.1	3	168.0	82.5	0.0	0.0	5	8.6	7.4	0.0	0.0	5	221.4	187.9	0.0	0.0	5
		3	6125.8	5420.2	8.5	8.3	1	136.2	116.5	0.0	0.0	5	10.8	6.9	0.0	0.0	5	300.3	285.9	0.0	0.0	5
		4	6254.6	5865.2	11.3	11.1	2	60.2	52.4	0.0	0.0	5	10.8	8.2	0.0	0.0	5	625.5	602.7	0.0	0.0	5
	6	2	7238.0	7238.0	16.8	16.6	0	70.4	66.1	0.0	0.0	5	15.6	12.6	0.0	0.0	5	333.0	314.6	0.0	0.0	5
		3	7239.2	7239.2	17.2	17.1	0	469.8	327.7	0.0	0.0	5	34.8	25.9	0.0	0.0	5	847.6	747.0	0.0	0.0	5
		4	7239.4	7239.4	17.0	16.9	0	1166.4	867.4	0.0	0.0	5	111.6	39.3	0.0	0.0	5	1544.5	1377.4	0.0	0.0	5
	8	2	7236.2	7236.2	19.4	19.3	0	589.6	519.7	0.0	0.0	5	39.4	34.6	0.0	0.0	5	501.1	485.9	0.0	0.0	5
		3	7233.0	7233.0	20.0	19.9	0	7224.4	7224.4	17.9	17.5	0	163	99.0	0.0	0.0	5	880.7	836.1	0.0	0.0	5
		4	7231.4	7231.4	20.4	20.3	0	7246.2	7245.8	21.4	21.2	0	2189.2	531.1	0.0	0.0	5	1382.1	1348.7	0.0	0.0	5
7	4	2	210.0	194.3	0.0	0.0	5	43.0	41.2	0.0	0.0	5	4	3.9	0.0	0.0	5	91.8	88.0	0.0	0.0	5
		3	214.2	197.3	0.0	0.0	5	147.0	49.1	0.0	0.0	5	6.8	5.7	0.0	0.0	5	195.7	185.7	0.0	0.0	5
		4	206.2	203.1	0.0	0.0	5	40.2	38.0	0.0	0.0	5	12.8	9.0	0.0	0.0	5	531.1	412.7	0.0	0.0	5
	6	2	7225.8	7225.8	9.3	8.8	0	53.2	48.5	0.0	0.0	5	19.8	14.7	0.0	0.0	5	189.1	182.0	0.0	0.0	5
		3	7224.4	7224.4	9.3	8.6	0	1389.0	356.0	0.0	0.0	5	53.6	35.3	0.0	0.0	5	387.7	341.8	0.0	0.0	5
		4	7225.8	7225.8	8.4	8.3	0	981.0	478.7	0.0	0.0	5	85.4	51.9	0.0	0.0	5	865.1	778.5	0.0	0.0	5
	8	2	6399.2	6091.8	12.9	12.8	0	5269.0	4603.9	7.0	7.0	2	31.4	26.8	0.0	0.0	5	422.3	398.9	0.0	0.0	5
		3	7233.4	7233.4	14.3	14.1	0	7233.0	7232.9	10.7	10.2	0	69	45.6	0.0	0.0	5	773.9	745.5	0.0	0.0	5
		4	5194.4	4915.1	13.1	13.0	0	3644.2	2982.4	20.9	20.9	4	471.2	133.6	0.0	0.0	5	1000.7	973.4	0.0	0.0	5

The results clearly demonstrate the benefits of decomposition branching (for these instances). Enhanced decomposition branching (DBE) finds an optimal solution in all but three of the 135 instances within the time limit, whereas CPLEX only solves 21 of the instances within the time limit. Note too that when CPLEX fails to find an optimal solution, the gaps are not small (around 30% for the harder instances). The difference in solution times is also striking. For the instances solved by both algorithms, DBE is sometimes more than 1000 times faster than CPLEX. The reason is obvious: decomposition branching results in drastically smaller search trees. As expected, the solution times for decomposition branching start to increase when instances get larger, especially when the block connectivity (s) increases and decomposition branching needs to explore more nodes to converge. Importantly, we see that DBE also outperforms Dantzig-Wolfe decomposition. There are many instances where DBE is more than 40 times faster than DW. Also, looking at the solution statistics for the most challenging instances, i.e., with $W = 3$, and $s = 4$, for which both algorithms sometimes fail to prove optimality within the given time limit, the optimality gaps for DBE are much smaller than those for DW.

The results also show the impact of the order in which the branching subproblems are processed. When processing branching subproblems in nondecreasing order of the current lower bound on their objective function value, only three instances cannot be solved to optimality within the time limit (as opposed to 31 when branching subproblems are processed in the order in which they appear in the formulation). Furthermore, the enhanced version is around 10 times faster, where the difference is more pronounced for instances with a more uniform weight distribution ($W \leq 5$), a larger number of blocks (M) and a larger block connectivity (s). Such instances have a larger root gap and hence improving their (global) lower bound faster is critical. In Table 2, we present further details related to the performance of DB and DBE. For each instance, we report (i) the number of branch-and-bound nodes explored (BB) solving the master problem, (ii) the total number of nodes processed when solving branching subproblems (the number of nodes processed when solving a branching subproblem is taken to be one if it solves at the root node), (iii) the number of times a branching subproblem is solved, (iv) the number of times a branching subproblem solve is avoided using information stored from a previous branching subproblem solve (as a fraction/percentage of the total), and (v) the average solve time of a branching subproblem (in seconds). Table 2 shows the geometric mean for (i) and (ii) and the arithmetic mean for (iii) and (iv) over the five instances in each class of instances in the columns headed BB, sub-BB, sub-solve, and sub-avoid, respectively. The columns headed sub-time report (v) for DB and DBE. By reporting the geometric mean for (i) and (ii), we reduce the effect of "outliers" on these statistics. We report the same statistics, but with arithmetic mean for (i) and (ii) in Table 5 in Appendix A.

Most importantly, we see that the time required to solve a branching subproblem, on average, is very small. Even though the branching subproblem is, essentially, a weighted set cover problem, it has a much smaller set of nodes compared to the set of nodes defining the full instance. While CPLEX cannot solve a full instance within the time limit of 7200 seconds (in most cases), it can solve the branching subproblems in less than one second. This highlights the fundamental premise and advantage of decomposition branching: solving a large MIP by solving a sequence of small MIPs can be much faster than solving the large MIP directly. For the WSC instances, both the number of branching subproblems solved and the time required to solve them are quite small, and, hence, decomposition branching performs much better than traditional branch-and-bound. Furthermore, we see that storing and re-using subproblem solutions (which is done in both variants) is critical, especially when the instances get harder to solve. The number of times branching subproblem

Table 2: DB and DBE solution details for WSC instances.

				DB					DBE				
W	N	s	CPLEX BB	BB	sub-BB	sub-solve	sub-avoid	sub-time	BB	sub-BB	sub-solve	sub-avoid	sub-time
3	4	2	9676718.5	992.0	3937.2	207.0	89.91	0.32	57.1	18.1	27.0	66.67	0.51
		3	6435546.6	4806.0	10181.6	863.2	95.49	0.33	123.4	148.0	66.2	86.00	0.51
		4	5988718.0	25123.3	28473.1	4739.2	95.62	0.29	165.1	4937.3	104.6	94.19	0.48
	6	2	8546101.4	51613.7	18306.3	1464.6	99.14	0.30	799.4	7358.1	118.0	91.08	0.45
		3	6492743.6	251879.9	50582.8	13267.4	98.47	0.25	452.9	1219.4	114.4	97.92	0.49
		4	8322687.9	326785.2	106841.9	22677.6	97.77	0.22	894.4	1482.7	162.6	97.82	0.49
	8	2	6674443.4	1038472.9	46401.6	3652.2	99.92	0.34	1386.5	5513.2	165.4	96.31	0.47
		3	5187285.4	427804.7	87064.1	12175.6	99.37	0.26	7688.7	13871.6	455.4	99.25	0.43
		4	6940123.1	265958.2	120468.5	15764.8	98.74	0.26	103594.3	38605.8	2511.8	99.64	0.39
5	4	2	12398253.3	201.4	1.0	138.2	87.55	0.29	32.7	1.0	19.0	48.65	0.38
		3	12907732.2	454.1	2.2	107.2	89.22	0.34	44.2	2.2	22.6	71.10	0.39
		4	11182910.9	966.6	1.0	191.6	91.31	0.32	43.0	1.0	22.6	63.90	0.38
	6	2	6936629.7	1797.2	3.1	225.2	96.06	0.30	131.1	2.6	31.8	78.97	0.39
		3	6188938.9	18721.5	15.0	1512.6	97.87	0.23	240.3	1.0	60.8	89.50	0.38
		4	5513324.7	48734.4	43.4	3100.6	98.07	0.29	415.6	3.6	106.6	96.30	0.39
	8	2	9933888.7	58979.2	7.4	1010.0	99.68	0.28	805.6	9.6	78.8	94.11	0.35
		3	7867518.1	434202.6	22.1	5155.4	99.75	0.26	2502.6	13.0	126.6	98.80	0.43
		4	7211619.6	564883.3	91.2	12089.6	99.45	0.23	13680.6	17.5	398.8	99.74	0.38
7	4	2	1104132.3	96.0	1.0	38.4	83.48	0.29	22.5	1.0	8.0	37.50	0.32
		3	1073646.4	302.8	1.0	94.0	87.06	0.30	47.3	1.0	16.6	73.57	0.31
		4	1009810.1	813.6	1.0	134.0	92.1	0.26	71.5	1.0	27.6	89.46	0.32
	6	2	7843993.9	1231.7	2.4	174.4	96.88	0.28	246.7	2.4	45.8	91.58	0.32
		3	9030398.3	14383.5	1.0	997.0	98.37	0.21	790.5	1.0	89.2	97.02	0.31
		4	10338690.9	38791.2	2.0	3444.6	99.02	0.17	910.1	1.0	123.6	95.97	0.29
	8	2	7294920.5	7892.1	2.2	361.6	98.69	0.28	733.9	2.7	64.8	96.87	0.34
		3	6516897.8	49405.2	2.5	1327.0	99.33	0.24	1797.6	2.6	110.2	98.59	0.30
		4	5885628.7	328959.3	1.0	3954.4	99.72	0.23	6163.6	1.0	257.8	99.65	0.31

state information can be used to avoid solving a branching subproblem increases rapidly when the instances get larger and more difficult to solve. Without this enhancement, decomposition branching might not have been able to solve as many instances as it has. The results also reveal the benefits of the subproblem ordering employed in DBE, namely, of processing the branching subproblems in nondecreasing order of their current objective lower bound value. On average, the branching subproblem solution times are higher, but the drastic reduction in the number of nodes in the search tree easily makes up for this.

5.2 p-median facility location with assignment range constraints

In this section, we analyze the computational performance of decomposition branching (and its suggested enhancements) when solving instances of a regionalized variant of the p-median facility location problem with assignment range constraints investigated by Daskin and Tucker (2018). The formulation used in our computational experiments is given below. The main difference with the formulation of Daskin and Tucker (2018) is that we have introduced regions so as to have a natural nearly-decomposable structure.

We are given a set of regions R, a set of demand points I, with I_r the set of demand points in region $r \in R$, a set of candidate facility locations J, with J_r the set of candidate facility locations in region $r \in R$, a quantity h_i at demand point $i \in I$, distances d_{ij} between demand point $i \in I$ and facility location $j \in J$, the maximum number p of facilities to locate, and the maximum allowable range of assigned demands to a facility in a region, ρ^{max}. We introduce the following decision variables: x_j, a binary variable indicating whether a facility is opened at location $j \in J$ or not, y_{ij}, a binary variable indicating whether a demand point $i \in I$ is assigned to a facility at location $j \in J$ or not, U_r, the maximum demand assigned to any of the open facilities in region $r \in R$, and

L_r the minimum demand assigned to any of the open facilities in region $r \in R$.

With these inputs and decision variables, our variant of the p-median facility location problem with assignment range constraints (FLAR) can be formulated as follows (where M_r is a suitably chosen constant):

$$\min \sum_{i \in I} \sum_{j \in J} h_i d_{ij} y_{ij} \tag{15}$$

$$\text{s.t.} \sum_{j \in J_r} y_{ij} \geq 1 \qquad \forall r \in R, i \in I_r \tag{16}$$

$$y_{ij} \leq x_j \qquad \forall r \in R, i \in I_r, j \in J_r \tag{17}$$

$$\sum_{j \in J} x_j \leq p \tag{18}$$

$$U_r \geq \sum_{i \in I_r} h_i y_{ij} \qquad \forall r \in R, \forall j \in J_r \tag{19}$$

$$L_r \leq \sum_{i \in I_r} h_i y_{ij} + M_r(1 - x_j) \qquad \forall r \in R, \forall j \in J_r \tag{20}$$

$$U_r - L_r \leq \rho^{max} \qquad \forall r \in R \tag{21}$$

$$x_j \in \{0, 1\} \qquad \forall j \in J \tag{22}$$

$$y_{ij} \in \{0, 1\} \qquad \forall r \in R, i \in I_r, j \in J_r. \tag{23}$$

Constraints (19)-(21) capture the assignment range constraints and limit the difference between the maximum demand assigned to a facility in a region and the minimum demand assigned to a facility in a region.

There is just a single constraint linking the regions, namely the one that limits the number of facilities that can be opened across the regions.

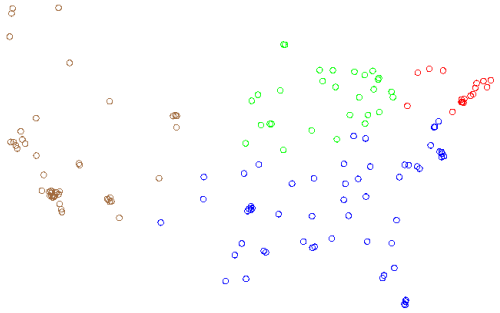
5.2.1 Instances

We generate instances using the 150-city and 880-city data sets from Daskin and Tucker (2018). To partition the cities into regions, we use the region and division definitions of the United States Census Bureau (2018). In Figure 2, we illustrate the partitions of the cities for the two data sets. More information on the region and division definitions can be found in Table 6 in Appendix B.

For the 150-city data set, we consider two instances: one with four and one with nine blocks (corresponding to US census regions and divisions, respectively). For the 880-city data set, we consider a single instance with nine blocks (corresponding to US census divisions).

5.2.2 Implementation

Because FLAR has only a single linking constraint, there is no need for constraint aggregation. On the other hand, as Figure 2 shows (and Table 6 does so even more clearly), the number of cities in a region can differ widely. Branching subproblems with fewer cities are likely to be easier to solve. Therefore, in what we refer to as the *basic* version of decomposition branching (DB), we process branching subproblems in nondecreasing order of size (i.e., the number of cities in the associated region). Preliminary experiments also revealed that some of the branching subproblems take quite



(a) Partition of the 150 node US data set into four census regions.

(b) Partition of the 880 node US data set into nine census regions.

Figure 2: Region definitions.

a long time to solve. Therefore, in what we refer to as the *enhanced* version of decomposition branching (DBE), we also incorporate an early termination strategy. To describe this strategy in detail, we introduce the following notation.

A branching subproblem is completely characterized by the region, r, and the maximum number of facilities allowed to be opened, u. We let $\underline{S}(r,u)$ and $\overline{S}(r,u)$ denote the best known lower and upper bound values, respectively, on the objective function value of a branching subproblem, for $r \in R$ and $u \in \{1,\ldots,p-(|R|-1)\}$ (because at least one facility has to be opened in each region). Furthermore, we let $\overline{z}_r(u,t,g)$ and $\underline{z}_r(u,t,g)$ denote the best upper and lower bounds found when solving the branching subproblem for region r for $t > 0$ seconds, or, if at time t the optimality gap is still greater than target g, until the time that the optimality gap drops below g. Finally, let $Solve(r,u,t,g)$ denote the process of solving branching subproblem r with control parameters u,t, and g. Note that when $Solve(r,u,t,g)$ terminates, the gap between $\underline{z}_r(u,t,g)$ and the optimal value for the branching subproblem for region r, restricted to at most u facilities, is guaranteed to be no more than g.

As mentioned above, the regions are processed in nondecreasing order of the number of demand points they contain. Given a fractional solution $(\bar{x},\bar{y})$ at a node in the search tree, the value of the best known feasible solution to the master problem, z^{UB}, and its associated optimality gap, $\bar{g} = 1 - \sum_{i\in I_r}\sum_{j\in J_r} h_i d_{ij}\bar{y}_{ij}/z^{UB}$, the steps taken by the enhanced version of decomposition branching are presented in Algorithm 2.

The algorithm first checks, for each of the branching subproblems associated with a region $r \in R$, if the sum of the values of the decision variables indicating whether or not a facility is opened in the current solution to the master problem, denoted by $\bar{X}_r = \sum_{j\in J_r}\bar{x}_j$, is fractional. If so, then the branching constraints $\sum_{j\in J_r} x_j \geq \lceil \bar{X}_r \rceil$ for one branch and $\sum_{j\in J_r} x_j \leq \lfloor \bar{X}_r \rfloor$ for the other, form a dichotomy that cuts off the fractional solution. In this case, since the branching subproblem value is not essential to determining branching constraints, the latter branch is strengthened through the use of a lower bound known to be within 10% of optimality, rather than with the true optimal value of the branching subproblem.

Preliminary computations revealed that even though it takes a significant effort to solve branching subproblems to optimality, high-quality solutions are often obtained early in the subproblem solve. To take advantage of this, we embed a simple heuristic procedure (H-MIP) to obtain high-quality feasible solutions during the tree search, which reduces the size of the search tree. More

Algorithm 2: DBE for FLAR.

Input: $\bar{x}, \bar{y}, z^{UB}, \bar{g}, t$

foreach $r \in R$ **do**
  Set $\bar{X}_{rr} := \sum_{j \in J_r} \bar{x}_j$

foreach $r \in R$ **do**
  if $\bar{X}_r$ *is fractional* **then**
    if $\left(1 - \underline{S}(r, \lfloor \bar{X}_r \rfloor)/\overline{S}(r, \lfloor \bar{X}_r \rfloor)\right) > 0.1$ **then**
      $Solve(r, \lfloor \bar{X}_r \rfloor, t, 0.1)$
      $\underline{S}(r, \lfloor \bar{X}_r \rfloor) := \max\{\underline{z}_r(\lfloor \bar{X}_r \rfloor, t, 0.1), \underline{S}(r, \lfloor \bar{X}_r \rfloor)\}$
      $\overline{S}(r, \lfloor \bar{X}_r \rfloor) := \min\{\overline{z}_r(\lfloor \bar{X}_r \rfloor, t, 0.1), \overline{S}(r, \lfloor \bar{X}_r \rfloor)\}$
    $branch_1 : \sum_{j \in J_r} x_j \geq \lceil \bar{X}_r \rceil$
    $branch_2 : \sum_{j \in J_r} x_j \leq \lfloor \bar{X}_r \rfloor \ \bigwedge \ \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} y_{ij} \geq \underline{S}(r, \lfloor \bar{X}_r \rfloor)$
    return

z^{UB}_{new} =H-MIP($\bar{X}$)
if $z^{UB}_{new} < z^{UB}$ **then**
  $z^{UB} := z^{UB}_{new}$

foreach $r \in R$ **do**
  if $\underline{S}(r, \bar{X}_r) > \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} \bar{y}_{ij}$ **then**
    $branch_1 : \sum_{j \in J_r} x_j \geq \bar{X}_r + 1$
    $branch_2 : \sum_{j \in J_r} x_j \leq \bar{X}_r \ \bigwedge \ \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} y_{ij} \geq \underline{S}(r, \bar{X}_r)$
    return

foreach $r \in R$ **do**
  if $\left(1 - \underline{S}(r, \bar{X}_r)/\overline{S}(r, \bar{X}_r)\right) > \bar{g}$ **then**
    $Solve(r, \bar{X}_r, t, \bar{g})$
    $\underline{S}(r, \lfloor \bar{X}_r \rfloor) := \max\{\underline{z}_r(\lfloor \bar{X}_r \rfloor, t, \bar{g}), \underline{S}(r, \lfloor \bar{X}_r \rfloor)\}$
    $\overline{S}(r, \lfloor \bar{X}_r \rfloor) := \min\{\overline{z}_r(\lfloor \bar{X}_r \rfloor, t, \bar{g}), \overline{S}(r, \lfloor \bar{X}_r \rfloor)\}$
    if $\underline{S}(r, \bar{X}_r) > \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} \bar{y}_{ij}$ **then**
      $branch_1 : \sum_{j \in J_r} x_j \geq \bar{X}_r + 1$
      $branch_2 : \sum_{j \in J_r} x_j \leq \bar{X}_r \ \bigwedge \ \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} y_{ij} \geq \underline{S}(r, \bar{X}_r)$
      return

$z^{UB}_{new} := 0$
foreach $r \in R$ **do**
  $Solve(r, \bar{X}_r, \infty, 0)$
  $\underline{S}(r, \bar{X}_r) := \underline{z}_r(\bar{X}_r, \infty, 0)$
  $\overline{S}(r, barX_r) := \overline{z}_r(\bar{X}_r, \infty, 0)$
  if $\underline{S}(r, \bar{X}_r) > \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} \bar{y}_{ij}$ **then**
    $branch_1 : \sum_{j \in J_r} x_j \geq \bar{X}_r + 1$
    $branch_2 : \sum_{j \in J_r} x_j \leq \bar{X}_r \ \bigwedge \ \sum_{i \in I_r} \sum_{j \in J_r} h_i d_{ij} y_{ij} \geq \underline{S}(r, \bar{X}_r)$
    return
  else
    $z^{UB}_{new} := z^{UB}_{new} + \overline{S}(r, \bar{X}_r)$

if $z^{UB}_{new} < z^{UB}$ **then**
  $z^{UB} := z^{UB}_{new}$

specifically, when we have $\bar{X}_r$ integer, for all $r \in R$, we set a short time limit (10 seconds) for solving the branching subproblems with the allocations $\bar{X}_r, r \in R$ and combine the best solutions found within this time to form a feasible solution for the whole problem.

After running H-MIP to look for a new upper bound, the algorithm checks, for each of the subproblems, if there is a stored subproblem lower bound that provides a branching constraint on the objective function value that eliminates the current solution to the master problem. If the algorithm encounters such a stored subproblem lower bound, it uses it to branch.

If none of the branching subproblems has a stored lower bound that provides a branching constraint, then the algorithm seeks a branching subproblem that, when solved to a smaller optimality gap than the current master problem gap, $\bar{g}$, yields a lower bound whose associated branching constraint cuts off the current master problem solution.

Finally, the algorithm solves the subproblems to optimality to find a branching constraint on the objective function value that eliminates the current solution to the master problem. If no such branching constraint can be found for any of the subproblems, then a feasible solution to the master problem has been identified. The algorithm updates the best known feasible solution, if necessary, and then fathoms the node.

5.2.3 Experiments

For the maximum number of facilities to open, p, we consider $p \in \{6, 9, 12, \ldots, 30\}$ for the instance with four regions and $p \in \{12, 15, \ldots, 30\}$ for the instances with nine regions. For the 150-city instances, we use assignment range limit $\rho_{max} \in \{100,000, 125,000\}$ and for the 880-city instances, we use assignment range limit $\rho_{max} \in \{600,000, 750,000\}$.

The overall time limit for the 150-city and 880-city instances are set to 10,800 and 36,000 seconds, respectively. For the branching subproblem solution time limit, we use $t = 25$ seconds, for the instances derived from the 150-city data set, and $t = 150$ seconds for the instance derived from the 880-city data set; this time limit does not apply to the final subproblem solution (lines 31-33 in Algorithm 2), where we need to solve the subproblem to optimality to ensure the correctness of the algorithm.

5.2.4 Analysis

The results of our experiments can be found in Table 3, where we compare the performance of decomposition branching (DB) and its enhancement with early termination and primal heuristic (DBE) to the performance of commercial solver CPLEX and the Danzig-Wolfe (DW) decomposition implementation in SCIP. The information presented is the same as in Table 1. Here we use "TL" to indicate that the time limit allowed for this instance was reached.

The results demonstrate the superior performance of decomposition branching, especially the enhanced variant. It is clear that incorporating early termination and a primal heuristic provides significant benefits, drastically reducing the solution time and finding optimal solutions for many instances for which none of the other algorithms can find even near optimal solutions.

Interestingly, we see that the performance of DW for the FLAR instances is less consistent than its performance for WSC instances, even though FLAR instances have only a single linking constraint.

Table 4 presents further details about the computational performance of DB and DBE. The column headings have the same meaning as in Table 2, but with $\#subSolve$ now counting how

Table 3: Results for FLAR instances.

				CPLEX		DB		DBE		DW	
ρ_{max}	prob	M	p	time	gap	time	gap	time	gap	time	gap
100000	150 city	4	6	111	0.01	204	0.00	7	0.00	11016	∞
			9	1041	0.01	565	0.00	29	0.00	11016	∞
			12	10806	1.48	1046	0.00	72	0.00	11015	∞
			15	10807	3.15	2629	0.00	146	0.00	11015	∞
			18	10804	2.25	2642	0.00	225	0.00	11015	∞
			21	10803	0.86	2243	0.00	305	0.00	11015	∞
			24	10804	1.25	2744	0.00	406	0.00	11016	∞
			27	10804	0.98	3233	0.00	484	0.00	11015	∞
			30	10803	0.41	4636	0.00	587	0.00	11015	∞
		9	12	107	0.01	239	0.00	6	0.00	45	0.00
			15	217	0.01	448	0.00	10	0.00	86	0.00
			18	6860	0.01	712	0.00	39	0.00	451	0.00
			21	10823	1.30	1048	0.00	137	0.00	124	0.00
			24	10821	1.54	1542	0.00	457	0.00	535	0.00
			27	10825	1.58	3022	0.00	1229	0.00	105	0.00
			30	10815	1.33	6520	0.00	2559	0.00	128	0.00
125000	150 city	4	6	53	0.01	106	0.00	6	0.00	10806	0.57
			9	137	0.01	568	0.00	22	0.00	11015	110.02
			12	9323	0.01	614	0.00	59	0.00	36719	173.80
			15	10806	2.29	1502	0.00	118	0.00	11015	188.23
			18	10805	1.33	1608	0.00	169	0.00	11015	238.14
			21	10802	0.52	2920	0.00	244	0.00	11015	292.13
			24	10803	0.90	3140	0.00	336	0.00	11015	352.64
			27	8138	0.01	2627	0.00	383	0.00	11015	425.66
			30	10805	1.01	3998	0.00	456	0.00	11015	499.31
		9	12	85	0.01	244	0.00	6	0.00	92	0.00
			15	186	0.01	497	0.00	11	0.00	157	0.00
			18	10800	0.04	1239	0.00	35	0.00	496	0.00
			21	10819	1.44	2641	0.00	120	0.00	240	0.00
			24	10833	1.95	3925	0.00	438	0.00	91	0.00
			27	10831	2.40	4702	0.00	1229	0.00	164	0.00
			30	10821	1.63	6242	0.00	2746	0.00	394	0.00
600000	880 City	9	12	1558	0.01	1332	0.00	151	0.00	5010	0
			15	8789	0.01	28400	0.00	240	0.00	36694	7.29
			18	36007	0.20	36000	8.66	163	0.00	28776	0
			21	36006	4.50	36000	0.83	338	0.00	28142	0
			24	20602	7.61	63650	6.46	624	0.00	36692	29.86
			27	18400	8.88	66756	10.20	1770	0.00	36689	40.82
			30	26392	8.97	54446	12.03	1841	0.00	31093	0
750000	880 City	9	12	1336	0.00	1686	0.00	159	0.00	6055	0
			15	3067	0.00	27714	0.00	360	0.00	36698	9.24
			18	36006	0.14	36000	6.05	275	0.00	35917	0
			21	30601	3.20	36000	3.87	289	0.00	7309	0
			24	25671	6.65	45893	8.01	572	0.00	36692	21.06
			27	33687	7.62	44223	10.64	1101	0.00	36689	∞
			30	16154	7.41	39821	17.78	1875	0.00	36692	22.28

many times the function $Solve()$ is executed.

The results clearly show that, unlike for the WSC instances, solving the branching subproblems is challenging and time-consuming. For the 150-city instances, the branching subproblems can still be solved in a reasonable time, but for the 880-city instances solving the branching subproblems becomes computationally prohibitive without the early-termination enhancement. This is evidenced by the fact that even though a time limit of 36,000 was imposed, for the instance with $p = 27$ and

Table 4: DB and DBE solution details for FLAR instances.

					DB					DBE				
rmax	prob	N	p	CPLEX BB	BB	sub-BB	sub-solve	sub-avoid	sub-time	BB	sub-BB	sub-solve	sub-avoid	sub-time
100000	150 city	4	6	78085	11	4428	7	63.158	23.5	17	6262	6	73.913	0.91
			9	550051	31	32882	13	81.944	34.9	23	31104	11	68.571	2.24
			12	2933661	71	206432	16	92.233	53.5	83	105911	18	86.567	3.79
			15	2513949	185	874138	23	95.886	108.4	155	288737	24	90.805	5.83
			18	1713849	479	930104	32	97.868	77.9	323	363419	33	94.349	6.49
			21	865736	757	856861	43	98.252	48.5	667	403633	43	96.399	6.78
			24	1257866	1769	878216	54	99.147	44.9	1199	442985	55	97.491	7.05
			27	1639576	2071	1049243	61	99.209	47.7	1771	455401	64	98.109	7.24
			30	1670604	1073	1345564	65	98.278	67.8	3071	494292	76	98.693	7.31
		9	12	56580	75	416	16	94.872	14.3	85	416	18	91.781	0.25
			15	241333	233	416	23	98.113	18.5	471	416	25	97.908	0.28
			18	7355516	2211	26316	37	99.700	17.7	5276	23450	43	99.748	0.49
			21	10358962	10241	204973	54	99.917	13.7	32310	202471	63	99.942	0.80
			24	9684401	30973	226088	65	99.971	12.4	139647	162602	79	99.984	0.72
			27	8658653	83596	234552	68	99.989	19.8	395035	57480	95	99.993	0.52
			30	7265836	197475	239608	74	99.995	38.7	985429	225188	105	99.997	0.74
125000	150 city	4	6	38761	11	3976	7	72.000	11.4	17	4138	6	73.913	0.83
			9	149701	27	10979	12	78.571	41.0	21	9245	11	64.516	1.62
			12	3130062	83	190405	17	92.797	25.4	81	78160	18	86.154	3.03
			15	2489900	195	450800	23	96.082	62.9	159	127451	24	91.078	4.63
			18	1740275	459	591057	32	97.756	41.3	309	221646	32	94.275	5.10
			21	1004997	775	730704	43	98.294	61.2	657	275665	43	96.337	5.40
			24	1635509	1807	641797	54	99.163	49.0	1187	323251	53	97.605	5.93
			27	1489238	2063	697022	62	99.184	38.3	1773	298370	66	98.043	5.52
			30	1413044	2747	786366	67	99.348	54.6	3175	386111	76	98.739	5.69
		9	12	176590	75	0	16	94.872	14.9	97	0	19	92.369	0.20
			15	441551	235	0	23	98.115	20.4	423	0	25	97.835	0.28
			18	8751352	2259	5310	37	99.704	28.3	5272	8606	43	99.748	0.40
			21	10590957	10268	32586	55	99.910	38.6	29327	32378	62	99.936	0.44
			24	9239053	30271	53222	63	99.968	42.7	135396	54068	79	99.984	0.58
			27	10561325	93448	52481	68	99.989	39.7	395035	57480	95	99.993	0.52
			30	7798522	211686	54349	73	99.995	31.8	984460	59161	104	99.997	0.58
600000	880 City	9	12	104983	27	498193	12	83.562	49.9	23	11369	12	66.667	5.72
			15	447447	249	561386	24	98.021	28.1	48	0	15	84.375	12.72
			18	1775127	91	1472762	26	96.491	442.2	29	30782	16	77.465	6.39
			21	2157267	865	1680029	35	99.321	343.4	119	51717	25	90.942	8.62
			24	1303148	159	5121914	34	97.611	1749.1	569	128138	38	97.769	10.40
			27	889314	114	5373567	33	96.729	1924.1	2313	382884	55	99.855	14.74
			30	1298186	49	6150496	33	91.057	1547.5	3832	387163	52	99.535	14.77
750000	880 City	9	12	107381	27	325030	12	83.562	42.6	23	12188	12	66.667	6.01
			15	125037	243	415231	24	97.822	18.8	38	0	15	81.928	17.11
			18	2148935	123	596607	25	96.966	114.0	16	156536	18	82.524	12.41
			21	1820443	150	733444	29	97.013	113.6	99	50356	23	89.450	8.35
			24	1150639	82	4669157	30	97.263	1350.1	480	109617	36	97.475	9.88
			27	1274833	44	2368370	30	91.643	1374.1	1382	287589	48	98.815	12.62
			30	814396	10	4328473	18	73.529	2205.4	3635	241118	58	99.441	12.54

$r = 600000$ the DB algorithm finished after more than 66,000 seconds, indicating that the solution of the last branching subproblem took more than 30,000 seconds. (In our implementation, the time limit is checked only at the highest level and not during the execution of the subroutines.)

6 Final remarks

Our computational experiments with WSC and FLAR provide orthogonal perspectives on the performance of decomposition branching. In the former, the branching subproblems are easy to solve, and the linear programming bound is weak, whereas in the latter, the branching subproblems are hard to solve, but the linear programming bound is strong. The results demonstrate that in both cases decomposition branching can provide significant computational advantages over simply handing an instance over to a commercial solver.

Even though the initial computational results have far exceeded our expectations regarding the potential of decomposition branching, more research and experimentation is needed to fully understand and assess its benefits and how to best implement it. One of the interesting avenues for further research is to investigate recursive application of decomposition branching.

References

T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *ORL*, 34(4):361–372, 2006.

D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP, 1995.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. E. Lübbecke, E. Malaguti, and E. Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1-2):391–424, 2015.

R. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Math. Extra Volume: Optimization Stories*, pages 107–121, 2012.

M. Bodur, S. Ahmed, N. Boland, and G. L. Nemhauser. Decomposition of loosely coupled integer programs: A multiobjective perspective. *Optimization Online*, 2016.

M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer, 2014.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8: 101–111, 1960.

M. S. Daskin and E. L. Tucker. The trade-off between the median and range of assigned demand in facility location models. *International Journal of Production Research*, 56(1-2):97–119, 2018.

S. S. Dey, M. Molinaro, and Q. Wang. Analysis of sparse cutting planes for sparse milps with applications to stochastic milps. *Mathematics of Operations Research*, 43(1):304–332, 2017.

B. Dilkina, E. B. Khalil, and G. L. Nemhauser. Comments on: On learning and branching: a survey. *TOP*, 25(2):242–246, 2017.

G. Gamrath. *Generic branch-cut-and-price*. PhD thesis, 2010.

G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398, 2015.

G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL `http://www.optimization-online.org/DB_HTML/2020/03/7705.html`.

M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

A. M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematica; Programming Study*, 2:82–114, 1974.

M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey. *50 Years of Integer Programming 1958-2008*. Springer, 2010.

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

E. B. Khalil, P. Le Bodic, L. Song, G. L. Nemhauser, and B. N. Dilkina. Learning to branch in mixed integer programming. In *AAAI*, pages 724–731, 2016.

T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, et al. MIPLIB 2010. *MPC*, 3(2):103, 2011.

M. Kruber, M. E. Lübbecke, and A. Parmentier. Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer, 2017.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

P. Le Bodic and G. L. Nemhauser. How important are branching decisions: fooling mip solvers. *Operations Research Letters*, 43(3):273–278, 2015.

J. T. Linderoth and M. W. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP*, 25(2):207–236, 2017.

D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19: 79–102, 2016.

G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

United States Census Bureau. Census regions and divisions of the united states. `https://www2.census.gov/geo/pdfs/maps-data/maps/reference/us_regdiv.pdf`. Accessed: 2018-07-16.

F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130 (2):249–294, 2011.

F. Vanderbeck and M. W. Savelsbergh. A generic view of Dantzig–Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.

F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

L. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

Appendix A WSC solution statistics for DB and DBE

Table 5: DB and DBE solution details for WSC instances (arithmetic averages).

				DB					DBE				
W	N	s	CPLEX BB	BB	sub-BB	sub-solve	sub-avoid	sub-time	BB	sub-BB	sub-solve	sub-avoid	sub-time
3	4	2	10762313.2	1157.2	6752.4	207.0	89.91	0.32	63.8	882.6	27.0	66.67	0.51
		3	6483242.6	9875.6	15113.2	863.2	95.49	0.33	369.8	7396.8	66.2	86.00	0.51
		4	5991116.8	64432.2	40743.6	4739.2	95.62	0.29	1307.4	13796.0	104.6	94.19	0.48
	6	2	9148801.8	61137.8	19952.8	1464.6	99.14	0.30	928.0	8185.6	118.0	91.08	0.45
		3	6735174.8	313442.6	58392.2	13267.4	98.47	0.25	3564.0	11850.8	114.4	97.92	0.49
		4	9435653.8	334347.8	126717.4	22677.6	97.77	0.22	5913.4	26899.0	162.6	97.82	0.49
	8	2	7331639	1052678.0	48356.8	3652.2	99.92	0.34	3544.0	16023.8	165.4	96.31	0.47
		3	5206759.2	430498.4	89492.6	12175.6	99.37	0.26	27678.0	22921.2	455.4	99.25	0.43
		4	7437892.2	276824.6	140610.8	15764.8	98.74	0.26	339394.6	79919.8	2511.8	99.64	0.39
5	4	2	13592245.8	715.2	1.0	138.2	87.55	0.29	37.6	1.0	19.0	48.65	0.38
		3	16522725.8	552.4	12.4	107.2	89.22	0.34	72.2	12.4	22.6	71.10	0.39
		4	12047156.8	1312.6	1.0	191.6	91.31	0.32	60.4	1.0	22.6	63.90	0.38
	6	2	7320028.8	2344.2	59.2	225.2	96.06	0.30	147.8	23.0	31.8	78.97	0.39
		3	6232590.8	34165.2	88.4	1512.6	97.87	0.23	458.6	1.0	60.8	89.50	0.38
		4	5521535.6	74052.8	109.0	3100.6	98.07	0.29	2546.2	124.6	106.6	96.30	0.39
	8	2	10862502.4	80042.6	288.8	1010.0	99.68	0.28	1082.6	116.6	78.8	94.11	0.35
		3	8775009.6	594490.8	143.2	5155.4	99.75	0.26	8142.4	248.8	126.6	98.80	0.43
		4	7885695.2	577463.2	282.6	12089.6	99.45	0.23	123857.2	142.4	398.8	99.74	0.38
7	4	2	1238160	133.4	1.0	38.4	83.48	0.29	23.0	1.0	8.0	37.50	0.32
		3	1167754.8	381.4	1.0	94.0	87.06	0.30	73.4	1.0	16.6	73.57	0.31
		4	1069927	941.2	1.0	134.0	92.1	0.26	276.0	1.0	27.6	89.46	0.32
	6	2	8049311.8	2181.2	18.6	174.4	96.88	0.28	474.8	16.2	45.8	91.58	0.32
		3	9497699.2	28266.2	1.0	997.0	98.37	0.21	2118.4	1.0	89.2	97.02	0.31
		4	10740509	140734.6	7.2	3444.6	99.02	0.17	2841.4	1.0	123.6	95.97	0.29
	8	2	8040470.2	8601.8	11.0	361.6	98.69	0.28	1079.4	27.2	64.8	96.87	0.34
		3	6664075.4	68058.2	22.4	1327.0	99.33	0.24	4321.8	26.8	110.2	98.59	0.30
		4	5929355.4	409938.4	1.0	3954.4	99.72	0.23	40444.0	1.0	257.8	99.65	0.31

Appendix B Partition of cities into United States census regions and divisions

Table 6 details the partition of the cities in the two data sets into United States census regions and divisions (United States Census Bureau).

Table 6: Partition of cities into United States census regions and divisions.

		Number of Cities	
Region	Division	150 node	880 node
Northeast	New England	7	58
	Middle Atlantic	10	117
South	South Atlantic	25	144
	East South Central	11	73
	West South Central	21	91
MidWest	East North Central	18	117
	West North Central	11	114
West	Mountain	16	86
	Pacific	31	80