A Branch-and-Price-and-Cut Method for Computing an Optimal Bramble

Sibel B. Sonuc, J. Cole Smith, Illya V. Hicks

December 16, 2011

Abstract

Given an undirected graph, a bramble is a set of connected subgraphs (called bramble elements) such that every pair of subgraphs either contains a common node, or such that an edge (i, j) exists with node i belonging to one subgraph and node j belonging to the other. In this paper, we examine the problem of finding the bramble number of a graph, along with a set of bramble elements that yields this number. The bramble number is the largest cardinality of a minimum hitting set over all bramble elements on this graph. We provide a branch-and-price-and-cut method that generates columns corresponding to bramble elements, and rows corresponding to hitting sets. We then examine the computational efficacy of our algorithm on a randomly generated data set.

1 Introduction

The concept of brambles is first introduced by Seymour and Thomas [25], where they also show that a graph having bramble number k has a *treewidth* of $k - 1$. Following this result, determining a graph's bramble number received attention from many researchers. Bodlaender et al. [6] provide a heuristic for computing brambles on general graphs (and on planar graphs with slight modification on the algorithm), where brambles are constructed such that they potentially give the best bramble order while keeping the computation and construction times reasonable. Many other studies focus on efficient algorithms for specially structured graphs. On graphs that can be formed as Cartesian products of complete graphs, Lucena [16] finds lower bounds on treewidths by using brambles. This approach employs a methodology to generate brambles for such graphs so that computing the orders of these brambles is simplified. Birmelé et al. [5] use brambles to show that a k-prism has a treewidth of $O(k^2)$. By the graph minor theorem [23], there exists a function f such that for every graph with

treewidth at least k, there is a bramble of order $k+1$ and size $f(k)$. Grohe and Marx [12] show that there is an exponential lower bound for the function f, and opens the question of the existence of an exponential upper bound on f. Grigoriev et al. [11] compute brambles in grid minors to provide lower bounds on planar graphs. In this paper, we provide an optimal integer programming algorithm to compute a general graph's bramble number.

Finding optimal brambles may be beneficial for clustering or vertex covering type of problems where the subsets must have some type of pairwise "connectivity" or hierarchical structure. Vertex covering is similar to vertex partitioning but with the relaxation that vertices may be shared by subsets; this version of vertex covering is not to be confused with the more known vertex cover problem. For example, in wireless networks, vertex partitioning algorithms have been utilized to cluster "interfering nodes" together for better scheduling of the network to achieve higher data rates [1]. Furthermore, recent work has shown that using a vertex covering type of clustering approach offer better data rates than using just vertex partitioning [24]. Given the nature of a bramble set, this is indeed a structure that could prove beneficial for this type of application and others of a similar nature. In addition, finding optimal bramble sets may prove insightful about understanding the structure of optimal tree decompositions.

A more well known and active area of research has been addressing the problem of finding a (simple, undirected) graph's bramble number, given Seymour and Thomas' result that brambles provide a lower bound on the treewidth of graphs [25]. In graph-theory, identifying a graph's treewidth is essential because of the difficulty of problems in this area and the efficiency of algorithms utilizing tree decomposition. For fixed treewidths, tree decomposition has been shown to produce polynomial-time algorithms for a number of NP-complete problems modeled on graphs [3, 4, 8, 10]. In addition to its theoretical importance with respect to graph structure, tree decomposition is an important tool in computational and theoretical complexity. Practical algorithms utilizing tree decompositions have also been developed by Lauritzen and Spiegelhalter [14] and Koster et al. [13] (see also Röhring [28]). The notions of treewidth and tree decomposition were introduced by Robertson and Seymour [22] as part of a series of papers proving Wagner's conjecture (also known as the Graph Minors Theorem). However, the problem of finding the treewidth itself is shown to be NP-hard [2]. Therefore, several approaches have been developed to compute bounds of a graph's treewidth. In addition to the bramble number of a graph, lower bound approaches include utilizing degree [21], graph minors [7], the maximum cardinality search algorithm [15], and by adding edges [9].

In this paper, we present a branch-and-price-and-cut algorithm to optimally solve the bramble problem on general graphs. To our knowledge, this is the first exact optimization algorithm that has

been developed for this problem. The benefit of this approach is that it determines optimal bramble elements (i.e., connected subgraphs that induce the optimal bramble number), which in turn yield insights into optimal bramble solutions for general graphs. The remainder of this paper is organized as follows. Section 2 gives the problem statement and general outline for our algorithm. Sections 2.1–2.4 explain four phases of our algorithm: master problem, hitting set generation, column generation, and branching, respectively. We present the result of our computational experiments in Section 3 and give our concluding remarks in Section 4.

2 Branch-and-Price-and-Cut Approach

Let $G = (V, E)$ be an undirected graph with node set V and edge set E. Let X_i and X_j denote node sets of any two subgraphs of G. These subgraphs are said to be *touching* if $X_i \cap X_j \neq \emptyset$, or if $(u, v) \in E$ for some $u \in X_i$ and $v \in X_j$.

Let $\mathcal{B}$ be a set of connected subgraphs of G. If each pair of subgraphs in $\mathcal{B}$ is touching, then $\mathcal{B}$ is called a *bramble*. Each element of a bramble corresponds to a connected subgraph and is referred to as a *bramble element*. Let S be a minimum-cardinality hitting set over elements of bramble $\mathcal{B}$, i.e., $S \subseteq V$ such that $S \cap X_i \neq \emptyset$, $\forall X_i \in \mathcal{B}$. Then we say that the bramble $\mathcal{B}$ is of *order* $|S|$. The largest bramble order over all brambles that can be defined on G gives its *bramble number* b_G.

Let C be the index set of all connected subgraphs of G that correspond to candidate bramble elements for an optimal bramble, and let K be the index set of all vertex subsets corresponding to all possible hitting sets. Our solution method consists of four main phases. The first phase solves a *master problem*, which begins with a restricted set of possible bramble elements $\bar{C} \subset C$, and a restricted set of possible hitting sets $\bar{K} \subset K$. The model for this phase is given Section 2.1, and solves a linear program to select a set of bramble elements from $\bar{C}$ that maximizes the minimum-cardinality hitting set for the bramble over $\bar{K}$. A solution to this problem thus consists of a set of bramble elements (perhaps fractionally selected, since a linear program is used), and a minimum-cardinality hitting set from $\bar{K}$, the candidate hitting sets enumerated so far.

In the second phase (explained in Section 2.2), we determine if a smaller-cardinality hitting set exists for the bramble obtained from the restricted master problem. If so, then this new hitting set is added to $\bar{K}$ in the form of a cutting plane, and the restricted master problem is solved again. The algorithm continues to repeat the first two phases until no new hitting sets need to be added to the model. Then, the third phase is invoked.

The third phase, explained in Section 2.3, generates a new connected subgraph given a solution from the first two phases of the algorithm. This phase consists of a *column generation* problem, which

introduces a new column for the master problem that corresponds to a new candidate bramble element. The master problem is solved again after the new column is introduced, and the solution is passed to the column generation model once more to seek another column. The third phase terminates when no enterable columns (having a positive reduced cost) exist.

These three phases are repeated until no more hitting sets (rows) or candidate bramble elements (columns) are generated. If the master problem returns a binary selection of bramble elements, the algorithm terminates with an integer optimal solution. Otherwise, a fourth phase of branching is started (see Section 2.4) that branches on a fractionally-selected bramble element, and repeats first three phases for each branch.

2.1 Master Problem

Let S_k denote the k^{th} enumerated subset of V, $k \in K$, which corresponds to a candidate hitting set. The node set of the i^{th} connected subgraph (i.e., candidate bramble element) in C is denoted by $X_i \subseteq V$. We also define $T(X_i) \subseteq V$ to be the set of nodes that are touching X_i, i.e., node $u \in V$ belongs to $T(X_i)$ if and only if $u \in X_i$ or $(u, v) \in E$ for some $v \in X_i$. Therefore, if two connected subgraphs with node sets X_i and X_j are not touching, then we have $X_i \cap T(X_j) = T(X_i) \cap X_j = \emptyset$.

Define binary variables $x_i = 1$ if the subgraph represented by X_i is included in the bramble, and 0 otherwise, $\forall i \in C$. Also, let z be a variable representing the minimum cardinality of a hitting set that covers all selected bramble elements. An exponentially-sized formulation of the bramble problem is given as follows.

$$\max \quad z \tag{1a}$$

$$\text{s.t.} \quad z \leq |S_k| + \sum_{i \in C : X_i \cap S_k = \emptyset} x_i \qquad \forall k \in K \tag{1b}$$

$$x_i + x_j \leq 1 \qquad \forall i, j \in C : X_i \cap T(X_j) = \emptyset \tag{1c}$$

$$x_i \in \{0, 1\} \qquad \forall i \in C. \tag{1d}$$

The objective function (1a) maximizes the cardinality of any hitting set covering all of the bramble elements selected. The value of this objective function gives the order of the bramble computed with respect to those sets that we have enumerated so far. Constraints (1b) define the upper bounds on z, given by the cardinality of a hitting set S_k, plus the number of selected bramble elements that are not covered by this hitting set. Constraints (1c) ensure that the bramble elements selected are pairwise touching, and (1d) ensures the binariness of each x-variable.

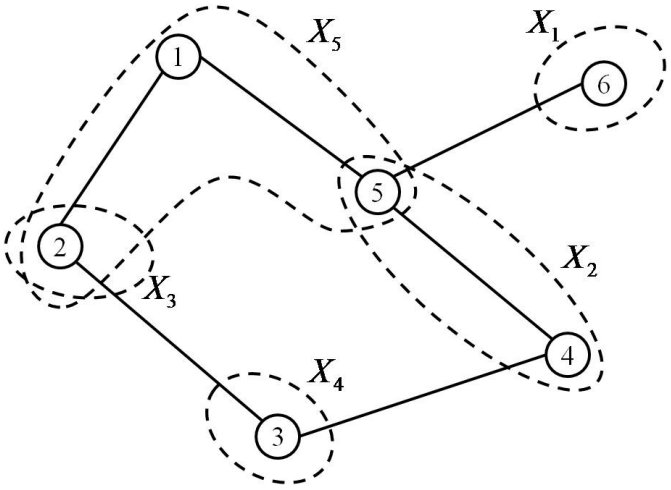

Figure 1: Graph G_1 and candidate bramble elements for Example 1

Lemma 1. Each constraint (1b) corresponding to $k \in K$ gives a valid upper bound on the bramble number of graph G.

Proof. Consider any solution $\hat{x}$, where $C' = \{i \in C : X_i \cap S_k \neq \emptyset\}$ and $C'' = \{i \in C : X_i \cap S_k = \emptyset\}$. For every $i \in C''$, let n_i be any arbitrary node belonging to X_i. Then S_k is a feasible hitting set for the elements in C', and $\cup_{i \in C''}\{n_i\}$ is a feasible hitting set for C''. Then the minimum cardinality hitting set for C is no more than $|S_k| + |\cup_{i \in C''}\{n_i\}| \leq |S_k| + \sum_{i \in C:\ X_i \cap S_k = \emptyset} x_i$.This bound corresponds to the right-hand-side of constraint (1b), thus verifying its validity. □

Rather than enumerating all sets in K and C, we consider a master problem formulation $\bar{K} \subseteq K$ and $\bar{C} \subseteq C$, and relax the binary constraints on the x-variables. We denote this (linear programming) master problem formulation as $\overline{\text{MP}}(\bar{K}, \bar{C})$. Example 1 shows how $\overline{\text{MP}}$ is modeled, given $\bar{K}$ and $\bar{C}$, for the graph G_1 in Figure 1.

Example 1. Consider the graph $G_1 = (V_1, E_1)$ depicted in Figure 1. Each candidate bramble element $X_i,\ i \in \bar{C} = \{1, \ldots, 5\}$, is encircled with dashed lines. Suppose that $|\bar{K}| = 2$, $S_1 = \{5\}$, and $S_2 = \{1, 6\}$. Then $\overline{\text{MP}}(\bar{K}, \bar{C})$ for this instance is formulated as follows:

$$
\begin{aligned}
\max\ & z \\
\text{s.t.}\ & z \leq 1 + x_1 + x_3 + x_4 \\
& z \leq 2 + x_2 + x_3 + x_4 \\
& x_1 + x_3 \leq 1 \\
& x_1 + x_4 \leq 1
\end{aligned}
$$

$$x_2 + x_3 \leq 1$$

$$0 \leq x_i \leq 1 \qquad \forall i \in \bar{C}.$$

An optimal solution for this problem is $x^* = \{0, 0, 1, 1, 1\}$ with $z^* = 3$, where the first constraint $z \leq 1 + x_1 + x_3 + x_4$ is the binding constraint for the objective function.

2.2 Adding a New Hitting Set

The solution of (1) yields a selection of bramble elements, $\{\hat{x}_j\}_{j \in \bar{C}}$, and the minimum cardinality, $\hat{z}$, with respect to the hitting sets listed in the index set $\bar{K}$. The next step is to find a new hitting set $S_{\tilde{k}}$, $\tilde{k} \in K \setminus \bar{K}$, having cardinality smaller than $\hat{z}$, if one exists. The identified hitting set results in a new constraint of the form (1b) that is violated by the current solution $\hat{z}$ and $\{\hat{x}_j\}_{j \in \bar{C}}$. Therefore, our cutting-plane generation model identifies a $\tilde{k} \in K$ that minimizes the right-hand side of (1b) over all elements of index set K with respect to a current master problem solution.

Let p_i be a binary variable that indicates whether node i belongs to hitting set $S_{\tilde{k}}$. Let q_j be a binary variable that equals 1 if $S_{\tilde{k}}$ does not hit bramble element X_j, for all $j \in \bar{C}$. Noting that $|S_{\tilde{k}}| = \sum_{i \in V} p_i$ and $\sum_{i \in \bar{C}: X_i \cap S_{\tilde{k}} = \emptyset} \hat{x}_i = \sum_{j \in \bar{C}} \hat{x}_j q_j$, the following problem seeks a hitting set $S_{\tilde{k}}$ that minimizes the right-hand side of (1b):

$$\min \quad \sum_{i \in V} p_i + \sum_{j \in \bar{C}: \hat{x}_j > 0} \hat{x}_j q_j \tag{2a}$$

$$\text{s.t.} \quad q_j \geq 1 - \sum_{i \in X_j} p_i \qquad \forall j \in \bar{C} : \hat{x}_j > 0 \tag{2b}$$

$$\sum_{i \in V} p_i \geq 1 \tag{2c}$$

$$q_j \geq 0 \qquad \forall j \in \bar{C} : \hat{x}_j > 0 \tag{2d}$$

$$p_i \in \{0, 1\} \qquad \forall i \in V. \tag{2e}$$

The objective function (2a) minimizes the right-hand-side of (1b). Constraint (2b) results in $q_j = 1$ only if $p_i = 0$ for all $i \in X_j$, and otherwise allows $q_j = 0$ (as enforced at optimality by the objective function). Note that there exists an optimal solution to problem (2) in which all q-variables are binary-valued; hence, only the nonnegativity of the q-variables needs to be enforced by (2d). Constraint (2c) ensures that a non-empty hitting set is considered, and constraints (2e) enforce binary restrictions on the p-variables.

Given $\hat{z}$ and $\{\hat{x}_j\}_{j \in \bar{C}}$, the constraint (1b) would be violated by $S_{\tilde{k}}$ if and only if $\hat{z}$ is larger than

the optimal objective function value to (2). In this case, the hitting set generated according to the p-solution of (2) is added to the master problem. Otherwise, if $\hat{z}$ equals the objective value of (2), no constraints of the form (1b) corresponding to hitting sets not in $\bar{K}$ are violated by the current solution $\hat{x}$.

Remark 1. Consider an optimal solution (p^*, q^*) to (2). Suppose that there exists an index $\hat{j} \in \bar{C}$ with $\hat{x}_{\hat{j}} > 0$ and $\sum_{i \in X_{\hat{j}}} p_i^* = 1$, i.e., $X_{\hat{j}}$ is covered by exactly one node in the hitting set. Let $\hat{i} \in X_{\hat{j}}$ be the node for which $p_{\hat{i}}^* = 1$, and further suppose that $\hat{i} \notin X_k,\ \forall k \in \bar{C} \setminus \{\hat{j}\}$. That is, node $\hat{i}$ belongs to the generated hitting set for the sole purpose of covering $X_{\hat{j}}$. Because (p^*, q^*) is optimal, we must have $\hat{x}_{\hat{j}} = 1$ and $q_{\hat{j}}^* = 0$. An alternative optimal solution would set $\bar{p}_{\hat{i}} = 0$ and $\bar{q}_{\hat{j}} = 1$, with all other $\bar{p}$- and $\bar{q}$-values matching those in (p^*, q^*). Letting $S_{\tilde{k}} = \{i \in V : p_i^* = 1\}$, this inequality becomes

$$z \leq (|S_{\tilde{k}}| - 1) + \sum_{i \in \bar{C} : X_i \cap (S_{\tilde{k}} \setminus \{\hat{i}\}) = \emptyset} x_i \quad . \tag{3}$$

The right-hand-side of (3) is less than that of (1b) generated according to $S_{\tilde{k}}$ when there is no bramble element X_j that cannot be covered by a vertex $i \in S_{\tilde{k}} \setminus \{\hat{i}\}$ (otherwise x_j is added to the summation on the right-hand-side of (3)). To generate this alternative inequality corresponding to (3), we could postprocess $S_{\tilde{k}}$ and remove any hitting set element that covers only one bramble element. On the other hand, if an equality (1b) is preferred that has the *largest* possible value of $|S_k|$, then we can verify that for each $i \in \bar{C} : \hat{x}_i = 1$, at least one node $j \in X_i$ exists in S_k. If not, one such node would be added to S_k. Our preliminary computational results show that it is slightly preferable to generate (1b) corresponding to large $|S_k|$, even though this has the effect of possibly generating slightly weaker inequalities than those corresponding to smaller $|S_k|$ sets. □

Example 2. Continuing Example 1, the next step would solve the minimum hitting set problem for the selected bramble elements X_3, X_4, and X_5 (corresponding to $x^* = \{0, 0, 1, 1, 1\}$). The minimum cardinality hitting set is $S_{\tilde{k}} = \{2, 3\}$. The solution for the hitting set model corresponding to $S_{\tilde{k}}$ is $p_2^* = p_3^* = 1$ and $p_i^* = 0$ for other values of i, and $q_j = 0$ for all $j \in \bar{C}$ with objective function value of 2. Because its cardinality is less than $z^* = 3$, we add the cut $z \leq 2 + x_1 + x_2$ to the master problem. Note that the same objective function value can be achieved by modifying the solution to (2) so that $p_3 = 0$ and $q_4 = 1$ instead, and other variable values match the p^*- and q^*-values: The sole purpose of placing node 3 in the hitting set is to cover bramble element X_4. The new inequality generated would be $z \leq 1 + x_1 + x_2 + x_4$ (as discussed in Remark 1).

2.3 Column Generation

The solution we seek from the column generation model is a connected subgraph of G with node set $X_n \subseteq V$, which corresponds to a new column in the master problem. From our optimal solution of (1), let α_k denote the dual variable associated with constraint (1b) corresponding to $k \in \bar{K}$. Let β_{ij} denote the dual variable associated with constraint (1c), corresponding to pair $i, j \in \bar{C}$ (where $\beta_{ij} \equiv 0$ if X_i and X_j are touching for notational convenience, noting that (1c) does not exist in this case), and let γ_i be the dual variables for the upper bounding constraints in the continuous relaxation of (1d), $\forall i \in \bar{C}$. The reduced cost of a new variable x_n is thus given by:

$$rc(x_n) = \sum_{k \in \bar{K}} u_k \alpha_k - \sum_{i \in \bar{C}} v_i \beta_{in} - \gamma_n, \tag{4}$$

where u_k is a binary constant that equals 1 if and only if $X_n \cap S_k = \emptyset$, and v_i is a binary constant that equals 1 if and only if $X_n \cap T(X_i) = \emptyset$, $\forall i \in \bar{C}$. Define $\hat{C}^1 \subseteq \bar{C}$ as the index set of bramble elements for which $\hat{x}_i = 1$ in the solution of (1). Essentially, the α_k-values are rewards for having X_n not intersect S_k, $\forall k \in \bar{K}$, and the β_{in}-values are penalties for having X_n not touching sets X_i, $\forall i \in \hat{C}^1$. Clearly, $\gamma_n = 0$ because x_n has not yet been generated, i.e., x_n currently equals zero, and γ_n cannot be positive due to complementary slackness.

2.3.1 Determining unknown duals

The only unknown values required to calculate $rc(x_n)$ are the β_{in}-values. For each X_i, $i \in \bar{C}$, that touches the new column represented by X_n, we must set $\beta_{in} = 0$ since no constraint $x_i + x_n \leq 1$ would be generated. Furthermore, note that β_{in} equals 0 necessarily for any $i \in \bar{C} : \hat{x}_i < 1$ in the current master problem solution, because the constraint $x_i + x_n \leq 1$ would not be binding (even if it exists due to X_i and X_n not touching). Therefore, we are only interested in basic variables such that $\hat{x}_i = 1$, $i \in \bar{C}$, in the solution of the master problem (1). We must thus determine appropriate β_{in}-values associated with any $i \in \hat{C}^1$ that does not touch X_n.

For $i \in \hat{C}^1$, it is possible to simply set $\beta_{in} = 0$ with X_i not touching X_n. On the other hand, using these cost coefficients may give an overly optimistic reduced cost for variable x_n, resulting in the generation of multiple columns that fail to enter the master problem basis. That is, if we set all $\beta_{in} = 0$ and identify a new column x_n with $rc(x_n) > 0$, we would generate the column and add it to the master problem. However, if we obtain $rc(x_n) \leq 0$, given a different set of duals, we would skip the generation of x_n, and would perhaps be able to terminate the column generation procedure more quickly.

The following three lemmas establish principles that allow us to derive nonzero β_{in}-values. For this discussion, the original dual solutions obtained from $\overline{\text{MP}}$ are labeled as $\bar{\beta}$ and $\bar{\gamma}$, and the revised duals as $\tilde{\beta}$ and $\tilde{\gamma}$.

Lemma 2. Consider any $i \in \hat{C}^1$ such that $\bar{\gamma}_i > 0$, and new variable $x_n,\ n \in C \setminus \bar{C}$. If X_i and X_n do not touch, then adjusting the dual optimal solution of (1) by setting $\tilde{\beta}_{in} = \bar{\gamma}_i$ and $\tilde{\gamma}_i = 0$ retains dual optimality to $\overline{\text{MP}}$.

Proof. To prove that dual optimality is retained, we need to prove that $(\tilde{\beta}, \tilde{\gamma})$ is still feasible and the dual objective remains unchanged. The dual constraint associated with primal variable x_i is

$$\gamma_i + \sum_{j \in \bar{C}: X_i \cap T(X_j) = \emptyset} \beta_{ij} \quad - \sum_{k \in \bar{K}: X_i \cap S_k = \emptyset} \alpha_k \quad \geq 0, \tag{5}$$

where $\gamma_i,\ \beta_{ij},\ \alpha_k \geq 0$ for all respective indices. After a new variable x_n is added to $\bar{C}$, variable β_{in} needs to be added to the summation on the left-hand-side of (5), where $\bar{\beta}_{in}$ (the current value of β_{in}) is taken as zero. Setting $\tilde{\gamma}_i = \bar{\gamma}_i - \Delta$ and $\tilde{\beta}_{in} = \bar{\beta}_{in} + \Delta$, the left-hand-side of (5) remains unchanged and feasibility is retained. Consequently, setting $\Delta = \bar{\gamma}_i$ (which is the maximum allowed value due to $\gamma_i \geq 0$) results in a feasible dual solution where $\tilde{\beta}_{in} = \bar{\gamma}_i$ and $\tilde{\gamma}_i = 0$. The objective function is now stated by

$$\sum_{i \in \bar{C}} \gamma_i + \sum_{i,j \in \bar{C}: X_i \cap T(X_j) = \emptyset} \beta_{ij} \quad + \sum_{k \in \bar{K}} |S_k| \alpha_k, \tag{6}$$

where both variables β_{in} and γ_i appear with a cost-coefficient value 1. Therefore, decreasing one of them by some Δ and increasing the other by the same Δ results in a zero net change in the objective function value. Hence, $\tilde{\beta}_{in} = \bar{\gamma}_i$ and $\tilde{\gamma}_i = 0$ gives an alternative optimal dual solution. □

Lemma 3. Consider a pair of variables x_i and x_j, where X_i and X_j do not touch, and where x_j has not been fixed to zero in the branching process. Suppose that $\hat{x}_i = 1$ and $\hat{x}_j = 0$ in the solution to $\overline{\text{MP}}$. Also, consider a new variable $x_n,\ n \in C \setminus \bar{C}$, representing the bramble element with node set X_n, where X_n does not touch X_i. Define $\Delta = \min\{\bar{\beta}_{ij}, -rc(x_j)\}$. Then adjusting the dual optimal solution of (1) by setting $\tilde{\beta}_{in} = \tilde{\beta}_{in} + \Delta$ and $\tilde{\beta}_{ij} = \bar{\beta}_{ij} - \Delta$ retains dual optimality.

Proof. Consider variables x_i and x_j as described, and again add β_{in} to the left-hand-side of inequality (5) due to the introduction of a new variable x_n to (1). Both dual variables β_{ij} and β_{in} appear with coefficient value of 1 in (5). Therefore, adjusting $\tilde{\beta}_{in} = \tilde{\beta}_{in} + \Delta$ and $\tilde{\beta}_{ij} = \bar{\beta}_{ij} - \Delta$ does not change its left-hand-side and retains feasibility with respect to this constraint. Further, $0 \leq \Delta \leq \beta_{ij}$ remains satisfied by our choice of Δ.

Algorithm 1: Computation of β_{in}-values

Initialize $\tilde{\beta}_{in} = \bar{\gamma}_i$, $\forall i \in \hat{C}^1$
while $\mathcal{L} \neq \emptyset$ **do**
  Remove (i, j) from $\mathcal{L}$; assume $i \in \hat{C}^1$
  if $rc(x_j) \leq 0$ **then**
    $\Delta = \min\{\bar{\beta}_{ij}, -rc(x_j)\}$
    $rc(x_j) = rc(x_j) + \Delta$
  else
    $\Delta = \bar{\beta}_{ij}$
  end
  $\tilde{\beta}_{in} = \tilde{\beta}_{in} + \Delta$
end

To guarantee optimality, we enforce the condition that the reduced cost of x_j remains nonpositive after the adjustment. The reduced cost of x_j is given by

$$rc(x_j) = \sum_{k \in \bar{K}: X_j \cap S_k = \emptyset} \alpha_k - \sum_{i \in \bar{C}: X_i \cap T(X_j) = \emptyset} \beta_{ij} - \gamma_j. \tag{7}$$

Subtracting $\Delta > 0$ from β_{ij} will increase $rc(x_j)$ by Δ, and so Δ can be no more than $-rc(x_j)$ to ensure that $rc(x_j)$ stays nonpositive. Because both β_{ij} and β_{in} have the same cost coefficient in the dual objective function (given in (6)), setting $\tilde{\beta}_{in} = \tilde{\beta}_{in} + \Delta$ and $\tilde{\beta}_{ij} = \bar{\beta}_{ij} - \Delta$ where $\Delta = \min\{\bar{\beta}_{ij}, -rc(x_j)\}$ does not change the dual objective function value and retains dual optimality. □

Lemma 4. Consider a variable x_j, $j \in \bar{C}$ that has been fixed to zero in the branching phase. Then for a pair (x_i, x_j) as described in Lemma 3, we can adjust $\tilde{\beta}_{in}$ by $\Delta = \bar{\beta}_{ij}$ without any limitation by $rc(x_j)$.

Proof. Follows from the arguments given for Lemma 3. Note that $rc(x_j)$ as given by (7) is now allowed to be positive, because a constraint $x_j \leq 0$ has implicitly been added, whose dual can be made arbitrarily large to ensure dual feasibility with respect to x_j. □

We are now ready to state our β-computation algorithm. Let $\mathcal{L}$ be an ordered list of given pairs (i, j) where $i \in \hat{C}^1$, $j \in \bar{C} \setminus \hat{C}^1$ and $\bar{\beta}_{ij} > 0$. Our algorithm begins by first applying Lemma 2 where possible, and then applying Lemma 3 or Lemma 4 to each $(i, j) \in \mathcal{L}$, one at a time. Algorithm 1 gives a formal statement of the procedure.

Note that Algorithm 1 yields a valid set of β-values, regardless of how $\mathcal{L}$ is ordered. However, the β_{in}-values computed by Algorithm 1 depend on the order of (i, j) pairs considered, as shown by the following example.

Example 3. Suppose that we have the touching constraints (1c) from Example 1 and optimal dual values ($\bar{\beta}$ and $\bar{\gamma}$) to the master problem, where x_1, x_2, and x_5 are basic variables that equal 1.

$$\begin{aligned} x_1 + x_3 &\leq 1 \qquad & \bar{\beta}_{13} &= 5/6 \\ x_1 + x_4 &\leq 1 \qquad & \bar{\beta}_{14} &= 2/3 \\ x_2 + x_3 &\leq 1 \qquad & \bar{\beta}_{23} &= 1/2 \\ x_5 &\leq 1 \qquad & \bar{\gamma}_5 &= 1/4 \end{aligned}$$

Suppose (only for the sake of illustration) that $rc(x_3) = -1/2$ and $rc(x_4) = -1/6$.

We consider the addition of column x_n to $\overline{\text{MP}}$, where X_n does not touch X_1, X_2, or X_5, leading to the addition of new constraints $x_1 + x_n \leq 1$, $x_2 + x_n \leq 1$, and $x_5 + x_n \leq 1$. Algorithm 1 computes dual values $\tilde{\beta}_{1n}$, $\tilde{\beta}_{2n}$, and $\tilde{\beta}_{5n}$ associated with these new constraints, first setting $\tilde{\beta}_{5n} = 1/4$ by Lemma 2 (regardless of $\mathcal{L}$), since $\bar{\gamma}_5 = 1/4$. Next, we calculate values for $\tilde{\beta}_{1n}$ and $\tilde{\beta}_{2n}$ via Lemma 3. We demonstrate the results of applying Algorithm 1 when $\mathcal{L} = [(2,3),(1,4),(1,3)]$ and $\mathcal{L} = [(1,3),(1,4),(2,3)]$.

First, consider $\mathcal{L} = [(2,3),(1,4),(1,3)]$. We extract $(i,j) = (2,3)$ from $\mathcal{L}$, and set $\Delta = \min\{\bar{\beta}_{23}, -rc(x_3)\} = \min\{1/2, 1/2\} = 1/2$. Then we set $\tilde{\beta}_{2n} = 1/2$ and $rc'(x_3) = 0$, noting that $\tilde{\beta}_{23} = 0$ now by Lemma 3. Next, we extract $(i,j) = (1,4)$, calculate $\Delta = \min\{\bar{\beta}_{14}, -rc(x_4)\} = \min\{2/3, 1/6\} = 1/6$, and update the values as $\tilde{\beta}_{1n} = 1/6$ and $\tilde{\beta}_{14} = 1/2$, while revising $rc(x_4) = 0$. Finally, we set $(i,j) = (1,3)$, and calculate $\Delta = \min\{\bar{\beta}_{13}, -rc(x_3)\} = \min\{5/6, 0\} = 0$. Since $\Delta = 0$, all values remain unchanged. We terminate since $\mathcal{L} = \emptyset$ with the revised dual values $\{\tilde{\beta}_{1n}, \tilde{\beta}_{2n}, \tilde{\beta}_{5n}\} = \{1/6, 1/2, 1/4\}$.

Now, consider $\mathcal{L} = [(1,3),(1,4),(2,3)]$. We first set $(i,j) = (1,3)$, and determine $\Delta = \min\{\bar{\beta}_{13}, -rc(x_3)\} = \min\{5/6, 1/2\} = 1/2$. Next, we set $\tilde{\beta}_{1n} = 1/2$ and $rc'(x_3) = 0$. Pair $(1,2)$ is then extracted from $\mathcal{L}$, and Δ is calculated as $\min\{\bar{\beta}_{14}, -rc(x_4)\} = \min\{2/3, 1/6\} = 1/6$. Thus, $\tilde{\beta}_{1n} = 2/3$, and $rc(x_4)$ is set to 0. Finally, setting $(i,j) = (2,3)$ gives us $\Delta = \min\{\bar{\beta}_{23}, -rc(x_3)\} = \min\{1/2, 0\} = 0$, and no values change. The algorithm terminates with $\{\tilde{\beta}_{1n}, \tilde{\beta}_{2n}, \tilde{\beta}_{5n}\} = \{2/3, 0, 1/4\}$. Hence, the two different sequences of $\mathcal{L}$ produce different $\tilde{\beta}_{in}$-values by Algorithm 1. □

2.3.2 Generating the new column

Now, we develop a mixed-integer programming model that identifies a maximum-reduced-cost column for (1) with respect to the dual values obtained from $\overline{\text{MP}}(\bar{K}, \bar{C})$ and modified by Algorithm 1. Our model works on a graph $\widetilde{G}$, which is created by appending a dummy node t to the original graph G, where node t is connected to every node $j \in V$ with a directed arc (t,j). Then $\widetilde{G} = (\widetilde{V}, \widetilde{E})$, where

$\widetilde{V} = V \cup \{t\}$ and $\widetilde{E} = E \cup \{(t, j) : j \in V\}$. A network flow enters G from the external source node t via a single arc to ensure connectedness of the node set X_n. The amount of supply at node t is equal to the number of nodes in X_n, which is determined by the model. We can verify that all nodes $j \in X_n$ are connected by ensuring that one more unit of flow enters node $j \in X_n$ than leaves it, and that no flow enters any node in $V \setminus X_n$. We denote the set of nodes in $\widetilde{V}$ adjacent to node $j \in \widetilde{V}$ by $N(j)$.

We define variables f_{ij} that represent the simulated flow from node i to node j, $\forall (i, j) \in \widetilde{E}$. Let y_j be a binary variable that equals 1 if and only if node $j \in X_n$, and let w_j be a binary variable that equals 1 if and only if there is a positive flow from node t to node $j \in V$. In the calculation of our reduced costs, we must know whether or not a bramble element intersects each enumerated hitting set in $\bar{K}$, and whether or not it touches the candidate bramble elements in $\hat{C}^1$. Hence, we introduce binary variables u_k, $\forall k \in \bar{K}$, which equal 1 if hitting set S_k does not have any common nodes with X_n, and $u_k = 0$ otherwise. Also, let v_i be a binary variable that equals 1 if and only if X_n and X_i do not touch, $\forall i \in \hat{C}^1$. The flow value f_{tj} between nodes t and j gives the number of nodes that belong to X_n.

We discuss two other features of the model before presenting the formulation. First, note that any bramble element having two or more nodes can be generated from multiple flow solutions, each in particular having $w_j = 1$ for a different $j \in X_n$. The presence of this symmetry in the model impairs its solvability (see, e.g., [17, 18, 19, 20, 26, 27]). To avoid this situation, we require that the bramble element X_n accepts flow coming from node t via its smallest-indexed node.

In addition, we must ensure that the column generation model does not regenerate any set X_i corresponding to a basic variable x_i in the current branching step, or one that has been forbidden by a prior branching step. In a generic column generation model, a column that corresponds to a basic variable x_i would have a zero reduced cost, and therefore would not be regenerated. However, our objective function for (8) is just an estimation for the reduced cost of x_n because we do not know the optimal solution for $\overline{\text{MP}}$ with the new variable x_n *a priori*. Hence, it is very likely that the column generation model estimates a positive reduced cost for a new variable replicating a basic variable x_i since it represents a favorable column. Thus, we have an additional constraint set ensuring that $X_n \neq X_i$ for all previously generated X_i. Our formulation of the column generation (CG) model is then given as follows.

$$\max \sum_{k \in \bar{K}} \alpha_k u_k - \sum_{i \in \hat{C}^1} \beta_{in} v_i \tag{8a}$$

$$\text{s.t.} \sum_{i \in V} w_i = 1 \tag{8b}$$

$$w_i \leq y_i \qquad \forall i \in V \tag{8c}$$

$$y_i \leq \sum_{j=1}^{i} w_j \qquad \forall i \in V \tag{8d}$$

$$f_{t1} \leq (|V|-1)w_1 \tag{8e}$$

$$f_{tj} \leq (|V|-j+1)w_j \qquad \forall j \in V \setminus \{1\} \tag{8f}$$

$$\sum_{i \in V} f_{ij} \leq (|V|-2)y_j \qquad \forall j \in V \tag{8g}$$

$$\sum_{i \in V} f_{ij} \leq (|V|-2)\sum_{k=1}^{j-1} w_k \qquad \forall j \in V \setminus \{1\} \tag{8h}$$

$$\sum_{j \in V} f_{ij} \leq (|V|-2)y_i \qquad \forall i \in V \tag{8i}$$

$$f_{tj} + \sum_{i \in N(j)} f_{ij} - \sum_{i \in N(j)} f_{ji} = y_j \qquad \forall j \in V \tag{8j}$$

$$\sum_{j \in X_i} (1-y_j) + \sum_{j \in V \setminus X_i} y_j \geq 1 \qquad \forall i \in \bar{C} \tag{8k}$$

$$u_k \leq 1 - y_j \qquad \forall k \in \bar{K},\ \forall j \in S_k \tag{8l}$$

$$v_i \geq 1 - \sum_{j \in T(X_i)} y_j \qquad \forall i \in \hat{C}^1 \tag{8m}$$

$$u_k \geq 0 \qquad \forall k \in \bar{K} \tag{8n}$$

$$v_i \geq 0 \qquad \forall i \in \bar{C} \tag{8o}$$

$$f_{i1} = 0,\ f_{ij} \geq 0 \qquad \forall i,j \in V \tag{8p}$$

$$w_j, y_j \in \{0,1\} \qquad \forall j \in V \tag{8q}$$

The objective function (8a) corresponds to the reduced cost of the generated column. Constraint (8b) ensures that node t initializes flow to create X_n through exactly one node i in V, and constraint (8c) forces that node i to be visited (i.e., $y_i = 1$) when $w_i = 1$. Constraints (8d) enforce anti-symmetry conditions by stating that if $w_j = 1$, no nodes $1, \ldots, j-1$ may belong to X_n. Constraints (8e) and (8f) prohibit flow from t to $j \in N$ if $w_j = 0$, and otherwise place an upper bound on f_{tj}. The upper bound on f_{t1} is $|V|-1$ in (8e), which disallows the generation of bramble element representing G itself. This exclusion is valid because the inclusion of a bramble element covering all nodes in V does not change the bramble order, due to the fact that every hitting set element belongs to this bramble element. The upper bound on f_{tj} is $|V|-j+1$ if $w_j = 1$ for some $j > 1$, because nodes $k = 1, \ldots, j-1$ are not included in the bramble element.

Constraints (8g) prohibit flow into node $j \in N$ if $y_j = 0$, and constraints (8h) prohibit internal

flow into node j if $w_1 = \cdots = w_{j-1} = 0$. In the latter case, even if $y_j = 1$, there exists a solution in which all flow entering node j arrives from external node t. Similarly, constraints (8i) prohibit flow out of node $i \in N$ if $y_i = 0$. Otherwise, constraints (8g)–(8i) give an upper bound on $\sum_{i \in V} f_{ij}$ of $|V| - 2$, since one unit of flow is consumed at some prior node i, and $|X_n| \leq |V| - 1$ by (8e) and (8f). Constraints (8j) are the flow balance constraints, which ensure that node j will consume one unit of flow if it belongs to X_n.

Constraints (8k) prevent regeneration of any bramble element in set P. Since the u_k-values are pushed toward 1 at optimality, we need only to force $u_k = 0$ by constraints (8l) when an element of S_k is chosen to be in X_n. Similarly, optimization pushes the v_i-values toward zero, and constraints (8m) force v_i to equal one if X_n does not touch X_i. Finally, constraints (8n)–(8q) state non-negativity and logical constraints on the variables, where fixing $f_{i1} = 0,\ \forall i \in V$, is justified by the same logic used to justify (8h).

2.4 Branching and Heuristic Scheme

The branching phase is invoked if the x-solution after the first three phases is not all integer. In this case, we need to branch on some $x_i,\ i \in \bar{C}$, for which $\hat{x}_i \in (0, 1)$ in $\overline{\text{MP}}(\bar{K}, \bar{C})$. Each branch defines a subproblem corresponding to fixing either $x_i = 0$ or $x_i = 1$. Every subproblem is then evaluated by applying the first three phases, and the branching steps are recursively applied to the subproblems as in a typical branch-and-bound procedure.

In order to improve the computation time of the proposed algorithm, we periodically apply a heuristic routine at nodes of the branch-and-bound tree. Given a fractional solution $\{\hat{x}_i\}_{i \in \bar{C}}$, the heuristic starts with index sets $\mathcal{F} = \{i \in \bar{C} : \hat{x}_i = 1\}$ and $\mathcal{A} = \{i \in \bar{C} : 0 < \hat{x}_i < 1\}$. Our heuristic attempts to create a feasible bramble solution containing as many elements from $\mathcal{F} \cup \mathcal{A}$ as possible, noting that given solutions consisting of brambles $\mathcal{B}'$ and $\mathcal{B}''$ with $\mathcal{B}' \subset \mathcal{B}''$, the order of $\mathcal{B}'$ is not more than $\mathcal{B}''$. We initialize our solution by fixing all elements in $\mathcal{F}$ equal to one, noting that these elements are all pairwise touching (and perhaps more likely to belong to an optimal bramble). We then sort the elements of $\mathcal{A}$ in non-increasing order of their $\hat{x}_i$-values. One-by-one, we determine if the next candidate bramble element in $\mathcal{A}$ touches all elements in $\mathcal{F}$. If so, we add the element to $\mathcal{F}$; otherwise, we discard it as required by feasibility. After the termination of the algorithm, we solve the minimum hitting set problem over $\mathcal{F}$ to determine the order of our solution. The summary of this process is given in Algorithm 2.

Algorithm 2: Lower bound heuristic

Set $\mathcal{F} = \{i \in \bar{C} : \hat{x}_i = 1\}$ and $\mathcal{A} = \{i \in \bar{C} : 0 < \hat{x}_i < 1\}$
while $\mathcal{A} \neq \emptyset$ **do**
  Choose $k \in \arg\max_{j \in \mathcal{A}} \{\hat{x}_j\}$
  if $X_k \cap T(X_i) \neq \emptyset \ \ \forall i \in F$ **then**
    $\mathcal{F} = \mathcal{F} \cup \{k\}$
  end
  $\mathcal{A} = \mathcal{A} \setminus \{k\}$
end
Let $\tilde{x}_i = 1 : \ \forall i \in \mathcal{F}$, and $\tilde{x}_i = 0 : \ \forall i \in \bar{C} \setminus \mathcal{F}$
Compute $\mathcal{LB}$ = cardinality of minimum hitting set over $\hat{x}$
Return $\mathcal{LB}$

3 Computational Results

In this section, we present computational results that demonstrate the efficacy of our proposed approached, and prescribe implementation details based on our experiments. We randomly generated 30 instances, where the i^{th} graph is denoted as $G_i(V_i, E_i)$. Table 1 gives the bramble number (z_i), number of nodes ($|V_i|$), number of edges ($|E_i|$) and corresponding density (d_i) for instance i. The density of graph G_i is measured by $100 \times |E_i|/[|V_i|(|V_i| - 1)/2]$, e.g., a complete graph has 100% density. Three instances are generated for each $(|V|, d)$ pair where the sub-indices are marked with # in Table 1. An instance is referred to by the number of nodes, density, and sub-index associated by its graph (e.g., G12-25-1 is the first twelve-node instance with 25% edge density). To generate a random graph instance with the given number of nodes $|V|$ and edges $|E|$ (or density d), we start by randomly generating a $|V|$-node tree as follows. We add each node $u \in \{1, \ldots, |V|\}$ to the tree one-by-one and connect u to a randomly selected node v that is already in the tree. The second part of the procedure consists of randomly selecting the remaining $|E| - (|V| - 1)$ edges to complete the graph.

The branch-and-price-and-cut algorithm is implemented using CPLEX 11.0 and ILOG Concert Technology 2.5. The computations are performed on a Dell PowerEdge 2600 UNIX machine with two Pentium4 3.2 GHz (1M Cache) processors and 6.0 GB memory. We count the number of master problem iterations, the number of branches created, and the CPU time (in seconds) required by our algorithm for each instance. Each instance G_i is limited to a maximum of $600|V_i|$ MP iterations, after which the algorithm is terminated. The tables in this section show the performance of our algorithm with different parameter and algorithmic settings. Our algorithm's default settings, which are used in the following tables unless otherwise stated, are given as follows. We initialize $\mathcal{L}$ as described in Section 2.3.1 as an empty list, and add the indices of a bramble element pair to $\mathcal{L}$ each time a constraint (1c) is generated corresponding to the pair. The element pairs in the list are considered in a last-in-first-out

Table 1: Instance data

$\lvert V_i\rvert$	d_i	$\lvert E_i\rvert$	#	z_i	$\lvert V_i\rvert$	d_i	$\lvert E_i\rvert$	#	z_i
12	25	17	1	4	15	25	27	1	5
			2	3				2	4
			3	4				3	4
	40	27	1	6		40	42	1	8
			2	6				2	7
			3	5				3	7
	55	37	1	7		55	58	1	9
			2	7				2	9
			3	7				3	9
	70	47	1	8		70	74	1	11
			2	9				2	10
			3	8				3	11
	85	57	1	10		85	90	1	12
			2	10				2	12
			3	10				3	12

fashion when Algorithm 1 is applied. When branching is required by the algorithm, we branch on a variable x_i corresponding to an element X_i having the smallest cardinality, where ties are broken in favor of a variable having the smallest index. We do not incorporate the heuristic algorithm to improve the lower bound in branching calculations in the default setting of the algorithm.

The hitting set model (2) might have alternative optimal hitting sets with different cardinalities. Because of the objective of the optimal bramble number problem, the effect of favoring smaller hitting sets in the hitting set generation phase needs to be investigated. Table 2 presents results of our computational study on different perturbations on the objective function of HS model, where the first group of columns labeled as "no ϵ" corresponds to the original form of the HS objective with no perturbation. Following the arguments in Remark 1, for each $j \in \bar{C}$ with $\hat{x}_j = 1$, the coefficient of q_j is changed to $\hat{x}_j - \epsilon$ so that any bramble element of size one is ignored by the optimal hitting set. Such a perturbation of the objective function with a small $\epsilon > 0$ tends to generate smaller sized hitting sets, while retaining the same objective function value (within the subtracted ϵ's). The column group labeled as "with $-\epsilon$" of Table 2 shows that, perhaps surprisingly, it is not computationally effective to force the HS model to generate smaller sized hitting sets. One possible reason for this behavior appears to be that large hitting sets tend to enable more bramble elements to be selected with ease in the subsequent iterations of the algorithm without updating the set of generated hitting sets ($\bar{K}$) often. On the other hand, favoring larger hitting sets (given in column group "with $+\epsilon$" of Table 2)

Table 2: ϵ perturbation in HS objective

G			no ϵ			with $-\epsilon$			with $+\epsilon$		
$\|V_i\|$	d_i	index	CPU	# MP	# bb	CPU	# MP	# bb	CPU	# MP	# bb
12	25	1	9	335	33	14	430	45	9	337	35
		2	16	296	12	10	273	21	17	328	20
		3	23	418	23	18	353	23	22	415	22
	40	1	19	470	24	19	455	22	34	630	40
		2	7	230	10	8	287	12	7	244	12
		3	12	308	19	13	378	28	13	306	21
	55	1	12	384	18	16	422	20	13	376	18
		2	16	468	32	23	588	48	14	433	25
		3	7	285	18	37	592	21	10	323	22
	70	1	6	212	13	8	241	9	8	234	13
		2	9	261	6	7	228	6	8	249	7
		3	14	449	37	10	301	29	8	258	18
	85	1	17	347	2	6	156	2	7	174	2
		2	6	154	2	6	165	3	5	150	2
		3	5	161	4	8	244	7	5	152	6
15	25	1	27	543	40	31	564	37	37	636	54
		2	12	292	23	18	377	29	12	292	23
		3	38	749	83	38	735	104	40	747	88
	40	1	413	2305	45	2122	6941	876	350	2364	68
		2	60	957	44	50	753	45	66	972	52
		3	26	519	46	31	665	62	26	600	48
	55	1	55	1026	52	104	1726	66	70	1051	53
		2	69	1182	45	60	1041	32	57	990	31
		3	30	624	27	28	580	23	22	526	16
	70	1	15	315	8	23	396	13	17	349	7
		2	22	430	13	22	434	16	15	356	8
		3	30	535	19	40	676	22	32	540	19
	85	1	15	311	10	15	300	5	55	712	6
		2	14	358	24	16	319	11	16	358	19
		3	10	208	2	13	258	4	11	252	4

turned out to be slightly better, yet still not ideal either. Although adding the ϵ perturbation improved the efficiency of the algorithm significantly for a few instances, overall it also failed to outperform the other two strategies in most instances. Hence, we keep the original form of the HS objective function (given in (2a)) in the default setting of our algorithm.

In Section 2.3.1, we show that different $\mathcal{L}$ orderings lead to different cost coefficients for objective function (8a). We have tested different ordering strategies for $\mathcal{L}$ including FIFO, LIFO, cardinality based orderings and reduced-cost based orderings. Although the ordering of $\mathcal{L}$ determines coefficients of the objective function for column generation phase, computationally the overall performance of the algorithm stays almost the same under all ordering strategies. Under different objective functions, different columns might be given priority to enter the basis, yet the final set of columns to be added to the model does not change despite this prioritization. This result is due to the structure of the problem, where there is a number of bramble elements that determine the optimal bramble number, and the alternative optimal solutions can be generated via adding feasible bramble elements. On the other hand, LIFO ordering (where last generated variable is considered first) requires slightly fewer iterations for instances G15-70-1, G15-70-2 and G15-85-1 while its effect on CPU time is almost negligible. This result is not surprising, because the columns are generated with respect to their potential reduced cost, and it should be expected that the last column generated will be more likely to be effective in the algorithm. For this reason, the LIFO ordering scheme for $\mathcal{L}$ is used in the rest of the computational study in this paper.

Tables 3−4 summarize the results for different branching strategies for the fourth phase of the algorithm (presented in Section 2.4). Each branching strategy is characterized by two of the attributes of bramble elements (or their associated variables), one as primary and the other as tie-breaking. The branching rules we test in this paper include branching on the variable with the lowest index i (Rule i), with the smallest cardinality $|X_i|$ for the corresponding subgraph (Rule $|X_i|$), and with the most fractional x_i-value (i.e., the smallest value of $|\hat{x}_i - 0.5|$, Rule x_i). Each branching strategy is denoted as a pair of these rules: (primary rule, tie-breaking rule). No tie-breaking rule is necessary when primary rule is Rule i.

In Table 3, we compare branching on the lowest index i, the smallest cardinality $|X_i|$, and the most fractional x_i. In the latter two, any tie is broken in the favor of the variable with the lower index. The results given in this table show that branching on a variable corresponding to the smallest cardinality element significantly outperforms the other two branching rules. Especially with the larger graphs (G15-40, G15-55), the smallest-cardinality branching strategy is the only approach that allows the algorithm to terminate with an optimal solution within our computational limit. To test further

Table 3: Comparison of branching strategies — 1

G			i			$(\lvert X_i\rvert, i)$			(x_i, i)		
$\lvert V_i\rvert$	d_i	#	CPU	MP	bb	CPU	MP	bb	CPU	MP	bb
12	25	1	97	2836	608	9	335	33	117	2504	494
		2	16	305	12	16	296	12	16	313	13
		3	24	503	40	23	418	23	30	582	61
	40	1	79	1517	185	19	470	24	74	1378	128
		2	30	722	62	7	230	10	29	725	57
		3	18	449	45	12	308	19	17	395	34
	55	1	25	711	56	12	384	18	31	779	70
		2	16	424	29	16	468	32	15	416	28
		3	9	321	25	7	285	18	9	354	27
	70	1	7	249	17	6	212	13	7	252	16
		2	10	321	11	9	261	6	8	233	4
		3	27	811	112	14	449	37	39	1002	138
	85	1	16	347	2	17	347	2	16	347	2
		2	5	154	2	6	154	2	6	162	2
		3	5	161	4	5	161	4	5	165	4
15	25	1	422	3937	563	27	543	40	536	4126	540
		2	28	581	80	12	292	23	29	608	87
		3	30	678	75	38	749	83	35	757	94
	40	1	24612	>9000	489	413	2305	45	20098	>9000	732
		2	4990	8755	1098	60	957	44	1765	>9000	1340
		3	376	4259	777	26	519	46	862	>9000	1879
	55	1	4112	>9000	730	55	1026	52	2296	>9000	1140
		2	5308	>9000	810	69	1182	45	6515	>9000	747
		3	2307	>9000	1088	30	624	27	1967	>9000	1017
	70	1	15	352	11	15	315	8	17	352	11
		2	25	469	17	22	430	13	81	1247	121
		3	26	534	18	30	535	19	98	1194	88
	85	1	16	311	10	15	311	10	19	353	15
		2	16	358	24	14	358	24	17	362	23
		3	10	208	2	10	208	2	11	208	2

branching strategies, we keep Rule $|X_i|$ either as the primary or tie-breaking rule due to these results.

Table 4 shows the results for branching with Rule $|X_i|$ as the primary rule, with either Rule i (in the first group of columns) or Rule x_i (in the second group of columns) as tie-breakers. In Table 4, while Rule x_i reduces the number of iterations for some instances, using Rule i results in a slightly lower CPU time than other implementations in most cases due to avoidance of any additional re-ordering. For example, although the instances G12-25-1 and G15-55-1 have more iterations under Rule i, the resulting CPU time for both are still the lowest. While instance G15-40-1 is the only one with significantly better results under Rule x_i, the difference between the results observed for instance G15-40-2 is much stronger. Hence, we keep Rule $(|X_i|, i)$ as the default branching rule.

The last group of columns in Table 4 tests branching strategy $(x_i, |X_i|)$. These results indicate that not only it is important to pick a good pair of branching rules, it is also important to assign them into their correct roles as primary and tie-breaker. This strategy performs worse than its counterpart strategy $(|X_i|, x_i)$, and even fails to optimize two of the instances (G15-40-2 and G15-55-2) within the given time limit. In addition, these results support our observation from Table 3. When branching on the element with the smallest cardinality is adopted, the results are significantly better than the other branching strategies analyzed here. The performance of the algorithm is improved even when it is used as the tie-breaking rule in column $(x_i, |X_i|)$ in Table 4 (versus column (x_i, i) in Table 3) where a weak primary branching rule is adopted. This result is due to the structure of the problem, and in particular due to touching requirement of the selected bramble elements.

To provide intuition that explains the behavior of these branching rules, note that each time we branch on a variable x_i, we make a decision on whether or not to add the corresponding candidate bramble element X_i in the solution bramble. If we consider a large subgraph, it is more likely that this large bramble element will touch other bramble elements and intersect the binding hitting set. Therefore the choice on its selection may be less likely to affect the optimal master problem solution that is obtained. Hence, branching on a bramble element having a high cardinality value may just create two almost identical problems with the same result. On the contrary, if we consider a relatively small sized subgraph, which may touch relatively few bramble elements, it will affect the selection of the rest of the bramble elements as well as the hitting set. We have tested this observation by branching on a largest-cardinality element, and the results are given on a set of 6-node and 9-node graphs with varying densities. (These graphs are small due to limitations of this approach. Note also that it is not possible to generate a 25%-dense 6-node connected graph.) In Table 5, the first group of columns labeled as G give the graph characteristics, the column group labeled $(|X_i|, i)$ is as in Table 3, and the column group labeled as $(\sim |X_i|, i)$ gives the results for branching on a largest-cardinality

Table 4: Comparison of branching strategies — 2

G			$(\|X_i\|, i)$			$(\|X_i\|, x_i)$			$(x_i, \|X_i\|)$		
$\|V_i\|$	d_i	#	CPU	MP	bb	CPU	MP	bb	CPU	MP	bb
12	25	1	9	335	33	10	333	27	10	342	29
		2	16	296	12	16	296	12	16	296	12
		3	23	418	23	23	430	22	23	430	22
	40	1	19	470	24	20	472	23	17	415	17
		2	7	230	10	7	230	10	7	230	10
		3	12	308	19	14	321	21	13	305	21
	55	1	12	384	18	14	360	15	14	360	15
		2	16	468	32	18	473	33	20	512	33
		3	7	285	18	8	285	18	8	291	19
	70	1	6	212	13	5	187	8	6	187	8
		2	9	261	6	8	238	5	8	238	5
		3	14	449	37	16	455	36	16	455	36
	85	1	17	347	2	16	347	2	17	347	2
		2	6	154	2	5	162	2	7	162	2
		3	5	161	4	6	165	4	5	165	4
15	25	1	27	543	40	27	548	39	23	489	32
		2	12	292	23	13	292	23	13	292	23
		3	38	749	83	39	704	80	54	980	141
	40	1	413	2305	45	282	1900	31	402	2099	29
		2	60	957	44	348	2698	138	3349	>9000	1175
		3	26	519	46	31	633	56	107	1366	143
	55	1	55	1026	52	69	1019	49	105	1361	72
		2	69	1182	45	93	1245	44	3089	>9000	1137
		3	30	624	27	32	639	30	23	549	18
	70	1	15	315	8	15	315	8	17	315	8
		2	22	430	13	25	448	17	67	1007	81
		3	30	535	19	31	533	18	29	533	19
	85	1	15	311	10	19	353	15	21	353	15
		2	14	358	24	16	362	23	17	362	23
		3	10	208	2	10	208	2	11	208	2

element first. Under this rule, the total number of iterations performed increases drastically as the instances get larger. Even for 9-node instances, the limit of maximum number of iterations (5400) is reached, as opposed to the corresponding runs where branching is done on the smallest cardinality element, where it takes less than 4 seconds to terminate with an optimal solution. These results show that branching on a smallest-cardinality element appears to be the best strategy for this particular problem, paired with a tie-breaking rule customized to the structure of that instance.

The results for the application of Algorithm 2 are summarized in Table 6, where different frequencies for executing the heuristic within the branch-and-bound tree are compared. The first column group labeled as "no heuristic" reports the results for a branching scheme without any heuristic calculation to improve the lower bound for objective function value. In this setting, only the objective function values of the integer solutions found are used to determine a lower bound value. As we move toward the right-hand-side on the tables, the frequency for the use of the algorithm increases. The second column group gives the result where Algorithm 2 is employed at every fifth fractional solution, the third column corresponds to at every second fractional solution, and the last column group corresponds to use of the heuristic at every fractional solution found. The heuristic improves the results only in two instances (G12-55-3 and G12-70-2). Although applying the heuristic at every fifth fractional solution has the most number of improved results, there are some instances where this frequency gives the worst results (e.g., G12-55-1, G15-40-3). We observed that the optimal bramble number is often found in very early iterations of the algorithm for most instances, where the rest of the algorithm runs to fathom the remaining branches that have been opened. This observation indicates the efficiency of our column generation model (8) and cutting plane generation model (2), and so in the default setting of our branch-and-price-and-cut algorithm, we do not incorporate Algorithm 2. However, while this heuristic does not appear to be effective in reducing the computational time needed to solve the optimal bramble problem by our technique, we still recommend its periodic use within the branch-and-bound tree for situations in which a good feasible solution must quickly be obtained for the problem.

A common result from all of the tables given in this section regards the relationship between the density of the graph and performance of the algorithm to find the optimal solution. Both sparse graphs ($d < 40$) and dense graphs ($d > 55$) are observed to be significantly easier to solve compared to those with medium-density ($d \in \{40, 55\}$). In Tables 2–6, the number of iterations and CPU time increase significantly for medium-density instances. The reason that these medium-density instances are difficult to solve is due to the size of the problem they generate. When solving a sparse graph, the number of generated columns cannot grow too large, because there are not many connected candidate bramble elements to generate. In addition, among those bramble elements generated, only a few of

Table 5: Comparison of branching strategies — 3

G				$(\lvert X_i\rvert, i)$			$(\sim \lvert X_i\rvert, i)$		
$\lvert V_i\rvert$	$d_i(\lvert E_i\rvert)$	#	z_i	CPU	MP	bb	CPU	MP	bb
6	40 (6)	1	3	0	60	3	2	107	12
		2	3	0	46	3	1	80	8
		3	3	0	34	1	1	48	3
	55 (9)	1	3	1	45	2	2	73	9
		2	4	0	34	1	1	37	1
		3	4	1	54	2	1	53	2
	70 (11)	1	4	1	48	3	1	53	4
		2	4	0	48	2	1	52	3
		3	4	1	57	2	1	48	2
	85 (13)	1	5	0	29	1	1	29	1
		2	5	1	23	0	1	23	0
		3	5	0	27	0	1	27	0
9	25 (9)	1	3	1	79	6	39	1678	381
		2	3	2	80	8	2	100	13
		3	3	1	46	4	1	43	4
	40 (15)	1	4	1	69	4	154	>5400	1270
		2	4	2	95	10	156	>5400	1270
		3	4	3	145	11	164	>5400	1282
	55 (20)	1	6	3	170	7	30	1027	183
		2	5	4	168	7	190	>5400	1253
		3	4	4	161	14	96	3557	844
	70 (26)	1	6	2	109	5	7	247	32
		2	6	3	144	12	16	527	90
		3	6	2	136	5	175	>5400	1291
	85 (31)	1	8	1	59	1	2	56	1
		2	7	1	81	2	2	80	2
		3	7	1	58	1	2	77	3

Table 6: Effect of the lower bound heuristic

G			no heuristic			every 5			every 2			at all fractional		
$\|V_i\|$	d_i	#	CPU	MP	bb	CPU	MP	bb	CPU	MP	bb	CPU	MP	bb
12	25	1	9	335	33	9	322	31	9	322	31	10	322	31
		2	16	296	12	15	296	12	16	296	12	16	296	12
		3	23	418	23	21	418	23	22	404	21	22	404	21
	40	1	19	470	24	20	474	23	20	482	22	20	482	22
		2	7	230	10	6	230	10	7	230	10	7	230	10
		3	12	308	19	14	308	19	14	308	20	14	308	20
	55	1	12	384	18	15	397	19	15	392	18	15	392	18
		2	16	468	32	16	442	28	14	381	25	15	381	25
		3	7	285	18	8	280	17	26	474	20	7	246	17
	70	1	6	212	13	6	211	13	6	205	11	7	223	11
		2	9	261	6	10	261	6	8	203	2	8	203	2
		3	14	449	37	15	449	37	15	459	42	15	459	42
	85	1	17	347	2	17	347	2	16	337	1	16	337	1
		2	6	154	2	6	154	2	6	146	1	5	146	1
		3	5	161	4	5	150	2	5	141	1	5	141	1
15	25	1	27	543	40	20	455	28	20	455	28	20	455	28
		2	12	292	23	13	292	23	13	292	23	13	292	23
		3	38	749	83	39	728	88	38	742	87	39	742	87
	40	1	413	2305	45	392	2340	44	351	2125	41	389	2365	47
		2	60	957	44	48	836	36	72	1128	54	48	753	46
		3	26	519	46	29	616	47	27	557	48	27	550	42
	55	1	55	1026	52	50	911	48	57	1003	50	67	1176	58
		2	69	1182	45	64	1027	40	99	1338	42	100	1338	42
		3	30	624	27	26	574	28	31	680	24	28	622	26
	70	1	15	315	8	16	315	8	18	373	8	18	373	8
		2	22	430	13	24	430	13	29	556	30	30	556	30
		3	30	535	19	22	430	9	25	470	14	25	470	14
	85	1	15	311	10	16	311	10	17	311	10	17	311	10
		2	14	358	24	16	369	24	16	371	25	17	371	25
		3	10	208	2	10	208	2	10	210	2	10	210	2

Table 7: Algorithm performance on larger instances

G				CPU	MP	bb
$\|V_i\|$	d_i ($\|E_i\|$)	#	z_i			
16	40 (48)	1	8	167	1645	161
		2	8	1809	3974	131
17	40 (55)	1	9	455	3045	70
		2	9	161	2079	58
18	40 (62)	1	9	47416	7303	105
		2	8	17010	9107	281

them will be chosen in the optimal bramble because of the touching restriction given by constraints (1c).

Moreover, the tables given in this section show that highly dense graphs are even easier to solve compared to their sparse counterparts. In a dense graph, it is easier to determine a connected subgraph due to the excess number of edges present, which leads to an easier column generation model to solve. For the same reason, it is also easier to satisfy the touching requirement on the bramble elements, which consequently reduces the number of constraints (1c) in the master problem. Although a dense graph might lead to a lot of variables in the worst case because of the number of candidate bramble elements, an efficient column generation model prevents most of them from being generated and appear in the master problem. Having fewer constraints with a good set of candidate variables permits the master problem to identify an optimal solution at a very early iteration. On the other hand, the graphs with medium-density level carry the advantages of neither sparse nor dense graphs.

Table 7 shows the performance of our branch-and-price-and-cut algorithm on some larger instances. All of the large instances given correspond to graphs with 40% edge density, which is among the hard density levels for our test data. For instances having 16 or 17 nodes, the CPU time is closer to those with fewer nodes. However, instances with 18 nodes show a significant increase both in CPU and the number of master problem iterations. In addition, we have encountered a number of 18-node instances that were terminated early due to the limit on the maximum number of iterations. These results indicate that further algorithmic enhancements may be investigated for larger instances.

4 Conclusions

Seymour and Thomas [25] opened a new area of research when they introduced the concept of brambles (they called them *screens*) and proved that the treewidth of a graph is exactly one less than its bramble number. Today, the literature includes a vast variety of work searching for the most efficient techniques to create good brambles. In this paper, we proposed an exact algorithm based on a branch-and-price-and-cut strategy for computing the optimal bramble of a given graph. We modeled the optimal bramble problem as a three-stage mixed-integer problem, with a fourth stage of branching that ensures an integer solution. We showed that micro-level parameters of our algorithm such as the tendency for smaller or larger hitting sets, estimation of the dual values in the objective function of phase 3 and adding a lower bound heuristic in phase 4 do not affect the average performance of the algorithm. On the other hand, a good branching rule determines the performance of the algorithm because it defines the elements included in the optimal bramble. Finally, we showed that the edge-density of graph is significant in terms of difficulty of an instance. Our computational results indicate that sparse and dense graphs compose easier instances (dense graphs are slightly easier) where graphs with medium edge density (around 50%) yield the most difficult instances for computing the optimal bramble.

To the best of our knowledge, this study is the first exact algorithm for computing the optimal bramble number on general graphs. According to our computational results, the computational time to compute the optimal bramble number can be reasonably small when correct algorithmic settings are chosen. Our branch-and-price-and-cut algorithm with the suggested parameters and settings provide very good results on general graphs. On the other hand, with a few instances, different settings yield slightly better results than our default setting. This points to a possibility of further improvement on some aspects of the algorithm, especially on the branching scheme. One strategy that is worth investigating is a dynamic branching scheme where switching in-between branching rules is allowed. Another algorithmic enhancement to be investigated is related to customizing the phases of the algorithm depending on the edge density of the graph. Finally, it is possible that there exists a rich class of specially structured bramble instances that may be polynomially solvable. Using relaxations of the graph instances we consider (obtained by augmenting the graph with additional edges), we may be able to compute additional bounds on the bramble number that could be employed in our optimization scheme.

References

[1] V. Aggarwal, A. Avestimehr, and A. Sabharwal. On achieving local view capacity via maximal independent graph scheduling. *IEEE Transactions on Information Theory*, 57(5):2711–2729, 2011.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[4] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987.

[5] E. Birmelé, J. Bondy, and B. Reed. Brambles, prisms and grids. In Bondy et al., editor, *Graph Theory in Paris, Proceedings of a Conference in Memory of Claude Berge*, pages 37–44, Birkhäuser, 2007. Springer.

[6] H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth lower bounds with brambles. *Algorithmica*, 51(1):81–98, 2008.

[7] H. L. Bodlaender, A. Koster, and T. Wolle. Contraction and treewidth lower bounds. *Journal of Graph Algorithms and Applications*, 10(1):5–49, 2006.

[8] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1):555–581, 1992.

[9] F. Clautiaux, J. Carlier, A. Moukrim, and S. Negre. New lower and upper bounds for graph treewidth. *Lecture Notes in Computer Science*, 2647:70–80, 2003.

[10] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[11] A. Grigoriev, B. Marchal, and N. Usotskaya. On planar graphs with large tree-width and small grid minors. *Electronic Notes in Discrete Mathematics*, 32:35–42, 2009.

[12] M. Grohe and D. Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory, Series B*, 99(1):218–228, 2009.

[13] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.

[14] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[15] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM Journal on Discrete Mathematics*, 16(3):345–353, 2003.

[16] B. Lucena. Achievable sets, brambles, and sparse treewidth obstructions. *Discrete Applied Mathematics*, 155:1055–1065, 2007.

[17] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002.

[18] F. Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming*, 98(1):3–21, 2003.

[19] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. *Integer Programming and Combinatorial Optimization*, 4513:104–118, 2007.

[20] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Constraint orbital branching. *Integer Programming and Combinatorial Optimization*, 5035:225–239, 2008.

[21] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM Journal on Discrete Mathematics*, 10:146, 1997.

[22] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[23] N. Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[24] P. E. Santacruz, V. Aggarwal, and A. Sabharwal. Going beyond link scheduling: Distributed k-clique scheduling. preprint, 2011.

[25] P. D. Seymour and R. Thomas. Graph searching, and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.

[26] H. D. Sherali and A. Ghoniem. Defeating symmetry via objective perturbations and hierarchical constraints. *IIE Transactions*, 43(8):575–588, 2011.

[27] H. D. Sherali and J. C. Smith. Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407, 2001.

[28] A. T. von hein Röhrig, T. Hagerup, and H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, 1998.