Cutting Plane Algorithms for Solving a Stochastic Edge-Partition Problem

Z. Caner Taşkın* J. Cole Smith† Shabbir Ahmed‡
Andrew J. Schaefer§

June 24, 2009

Abstract

We consider the edge-partition problem, which is a graph theoretic problem arising in the design of Synchronous Optical Networks. The deterministic edge-partition problem considers an undirected graph with weighted edges, and simultaneously assigns nodes and edges to subgraphs such that each edge appears in exactly one subgraph, and such that no edge is assigned to a subgraph unless both of its incident nodes are also assigned to that subgraph. Additionally, there are limitations on the number of nodes and on the sum of edge weights that can be assigned to each subgraph. In this paper, we consider a stochastic version of the edge-partition problem in which we assign nodes to subgraphs in a first stage, realize a set of edge weights from a finite set of alternatives, and then assign edges to subgraphs. We first prescribe a two-stage cutting plane approach with integer variables in both stages, and examine computational difficulties associated with the proposed cutting planes. As an alternative, we prescribe a hybrid integer programming/constraint programming algorithm capable of solving a suite of test instances within practical computational limits.

1 Introduction

We begin by describing the *edge-partition problem* of Goldschmidt et al. (2003), which is defined on an undirected graph $G(N, E)$. In the deterministic edge-partition problem, we create a set K of (possibly empty) subgraphs of G such that each edge is contained in exactly one subgraph, subject to certain restrictions on the composition of the subgraphs. These

*Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA; e-mail: taskin@ufl.edu.

†Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA; e-mail: cole@ise.ufl.edu.

‡School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30032, USA; e-mail: sahmed@isye.gatech.edu.

§Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA; e-mail: schaefer@ie.pitt.edu.

restrictions include the stipulations that an edge cannot be assigned to a subgraph unless both of its incident nodes belong to the subgraph, and that no more than r nodes can be assigned to any subgraph, for some $r \in \mathbb{Z}^+$. Additionally, each edge $(i, j) \in E$ has a positive weight of w_{ij}, and the sum of edge weights assigned to each subgraph cannot exceed some given positive number b. The objective of the problem is to minimize the sum of the number of nodes assigned to each subgraph.

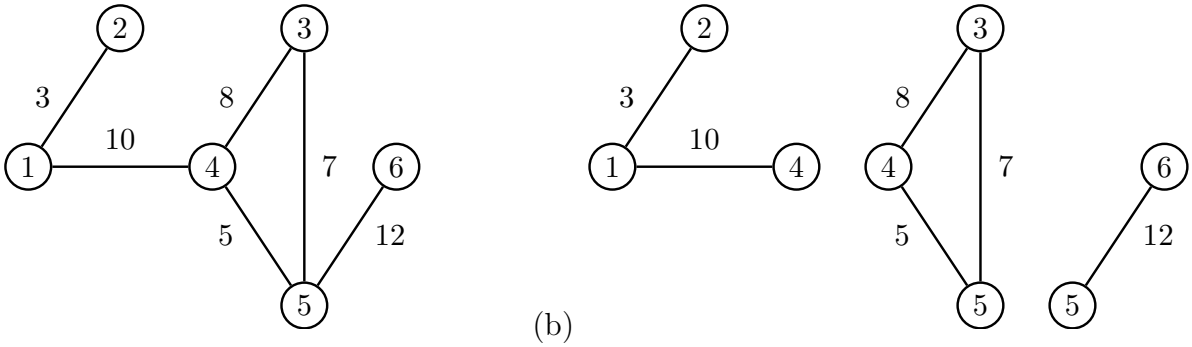

Figure 1: (a) An instance of the deterministic edge-partition problem. (b) A solution with $|K| = 3,\ r = 3,\ b = 20$.

Figure 1 illustrates the deterministic edge-partition problem. The graph G and the corresponding edge weights are shown in Figure 1a. Figure 1b shows a feasible solution that partitions G into $|K| = 3$ subgraphs, where the number of nodes in each subgraph is limited by $r = 3$, and the total weight assigned to each subgraph is limited by $b = 20$. Note that the degree of node 4 is three, which implies that it must be assigned to at least two subgraphs, or else there would be at least $4 > r$ nodes in a subgraph. Similarly, node 5 must be assigned to at least two subgraphs. Since nodes 4 and 5 are assigned to two subgraphs, and every other node is assigned to a single subgraph, the solution represented by Figure 1b is optimal.

Goldschmidt et al. (2003) discuss the edge-partition problem (with deterministic weights) in the context of designing Synchronous Optical Network (SONET) rings. In the SONET context, each edge $(i, j) \in E$ represents a demand pair between two client nodes, and the weight w_{ij} represents the number of communication channels requested between nodes i and j. All telecommunication traffic is carried over a set of SONET rings, which are represented by subgraphs in the edge-partition problem. Since each demand must be carried by exactly one ring, edges must be partitioned among the rings. (Note that the term "ring" describes only the physical SONET routing structure, and does not place any restrictions on topological properties of demand edges included on a ring. See, e.g., Goldschmidt et al. (2003) for more

details.) SONET rings are permitted to carry communication between nodes only if those nodes have been connected to the ring by equipment called Add-Drop Multiplexers (ADMs). There are technical limits on the number of ADMs that can be assigned to each ring (e.g., r), and on the total amount of channels (e.g., b) that can be assigned to a ring. Since ADMs are quite expensive, ring networks are preferred that employ as few ADMs as possible, which echoes the edge-partition problem's objective of minimizing the sum of nodes assigned to each subgraph.

The primary contribution by Goldschmidt et al. (2003) is an approximation algorithm for a specific case of the edge-partition problem in which all w_{ij} are equal to one. Sutter et al. (1998) propose a column-generation algorithm for this problem, and Lee et al. (2000) employ a branch-and-cut algorithm on a formulation that we adapt for this paper. For the case in which the weights on the edges can be split among multiple rings, Sherali et al. (2000) prescribe a mixed-integer programming approach augmented by the use of valid inequalities, anti-symmetry constraints, and variable branching rules. Smith (2005) formulates the deterministic version of the edge-partition problem as a constraint program, and examines several issues regarding symmetry and search algorithm design. Specifically, she shows how adding aggregate variables that represent the number of node copies (similar to our approach in Section 3) can improve performance.

In this paper, we consider a version of the edge-partition problem where the edge weights are uncertain, and are only realized after the node-to-subgraph assignments have been made. As we show in Section 2, this framework allows us to design more robust solutions than those in the literature, which are virtually all applied to deterministic data. We seek a minimum-cardinality set of node-to-subgraph assignments, such that there exists an assignment of edges to subgraphs satisfying the aforementioned subgraph restrictions with a pre-specified high probability. Such a probabilistic constraint is extremely hard to deal with in an optimization framework. One approach, known as *scenario approximation* (cf. Calafiore and Campi (2005); Luedtke and Ahmed (2008); Nemirovski and Shapiro (2005)) is to draw independent identically distributed (i.i.d.) realizations of the edge weights (called scenarios) and require the node-to-subgraph assignments to admit a feasible edge-to-subgraph assignment in each scenario. It can be shown that, with a sufficiently large scenario set, a solution to this scenario approximation solution is feasible to the true probabilistically constrained problem with high confidence. In this paper we develop algorithmic approaches for solving the scenario approximation corresponding to the discussed probabilistically constrained

edge-partition problem. This scenario approximation will be referred to as the *stochastic edge-partition problem*.

Relatively little work has been done in SONET network design when the edge weights are uncertain. Smith et al. (2004) consider the SONET ring design problem in which edge demands can be split among multiple rings, and propose a two-stage integer programming algorithm. The demand splitting relaxation allows the second-stage problems to be solved as linear programs, and thus standard Benders cuts can easily be derived from the second-stage recourse problems. However, in this paper we have second-stage integer programs from which dual information cannot be readily obtained.

The edge-partition problem can also be approached using stochastic integer programming theory. For problems having binary first-stage variables and mixed-integer second-stage variables, such as the problem studied in this paper, a well-known decomposition approach is the integer L-shaped method (Laporte and Louveaux, 1993). This method approximates the second-stage value function by linear "cuts" that are exact at the binary solution where the cut is generated, and are under-estimates at other binary solutions. Typical integer programming algorithms progress by solving a sequence of intermediate linear programming (LP) problems. Using disjunctive programming techniques, it is possible to derive cuts from the solutions to these intermediate LPs that are valid under-estimators of the second-stage value function at all binary first-stage solutions (Sherali and Fraticelli, 2002; Sen and Higle, 2005). This avoids solving difficult integer second-stage problems to optimality in all iterations of the algorithm, providing significant computational advantage. Scenario-wise decomposition methods have also been proposed (Carøe and Schultz, 1999) as an alternative to the above stage-wise decomposition approaches. Here copies of the first-stage variables are made corresponding to each scenario and are linked together via non-anticipativity constraints.

Our proposed methodology draws on constraint programming and stochastic integer programming theory. Hybrid algorithms of this nature have recently been successfully employed to solve notoriously difficult problems. Jain and Grossmann (2001) and Bockmayr and Pisaruk (2006) devise hybrid integer programming/constraint programming algorithms for solving machine scheduling problems. Thorsteinsson (2001) proposes a framework for integrating integer programming and constraint programming approaches. Hooker and Ottosson (2003) extend the Benders decomposition framework so that constraint logic programs can be used as subproblems to generate cuts that are added to a mixed-integer linear master problem. A recent work by Hooker (2007) uses logic-based Benders decomposition to solve several

planning and scheduling problems.

The remainder of this paper is organized as follows. In Section 2, we develop a mixed-integer programming formulation for the stochastic edge-partition problem, and provide cutting planes that can be used within a two-stage decomposition algorithm. In Section 3, we prescribe an alternative three-stage algorithm to overcome the computational difficulties associated with the weakness of the proposed cutting planes. We compare the efficacy of these algorithms in Section 4 on a set of randomly generated test instances. Finally, we conclude this paper in Section 5.

2 Formulation and Cutting Plane Approach

Let us introduce binary decision variables $x_{ik} = 1$ if node i is assigned to subgraph k and 0 otherwise, $\forall i \in N,\ k \in K$. For this formulation, we specify a value of $|K|$ that is sufficiently large to ensure that a feasible solution exists to the problem (as discussed in Section 4). We denote the vector of node-to-subgraph assignments by $\mathbf{x}$. Let $\tilde{\mathbf{w}}$ denote the random vector of edge weights with known distribution, and $\mathbf{w}$ denote a realization with components w_{ij}. We define binary decision variables $y_{ijk} = 1$ if edge (i, j) is assigned to subgraph k. Given an allowed violation probability $\epsilon \in (0, 1)$ the probabilistic edge-partition problem can be formulated as follows:

$$\text{Minimize} \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{1}$$

$$\text{subject to} \sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{2}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N,\ k \in K \tag{3}$$

$$\Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} \geq 1 - \epsilon \tag{4}$$

where

$$G(\mathbf{x}, \mathbf{w}) = \text{Minimize} \quad z \tag{5}$$

$$\text{subject to} \sum_{k \in K} y_{ijk} = 1 \quad \forall (i, j) \in E, \tag{6}$$

$$\sum_{(i,j) \in E} w_{ij} y_{ijk} \leq z \quad \forall k \in K \tag{7}$$

$$y_{ijk} \leq x_{ik},\ y_{ijk} \leq x_{jk} \quad \forall (i, j) \in E,\ k \in K \tag{8}$$

$$y_{ijk} \in \{0, 1\} \quad \forall (i, j) \in E,\ k \in K. \tag{9}$$

The objective (1) minimizes the total number of nodes assigned to subgraphs. Constraints (2) limit the number of nodes assigned to each subgraph. Constraints (6) require that the edges be partitioned among the subgraphs. Constraints (7) compute the maximum assigned weight over all subgraphs. Constraints (8) require that no edge can be assigned to a subgraph unless both of its incident nodes are assigned to that subgraph, and (3) and (9) state logical restrictions on the variables. By convention, the optimal value $G(\mathbf{x}, \mathbf{w})$ of the integer program (5)–(9) is $+\infty$ if the problem is infeasible. Given a node-to-subgraph assignment vector $\mathbf{x}$ and edge weight vector $\mathbf{w}$ there exists a feasible edge-to-subgraph assignment if and only if $G(\mathbf{x}, \mathbf{w}) \leq b$, i.e., the weight assigned to any subgraph does not exceed b. Thus the probabilistic edge partition problem (1)–(4) seeks a minimum cost node-to-subgraph assignment such that the probability that there will be a feasible edge-to-subgraph assignment when the edge weights are realized is sufficiently high.

To build a scenario approximation of the probabilistic edge partition problem (1)–(4), we generate an i.i.d. sample of $\tilde{\mathbf{w}}$ denoted by $\{\mathbf{w}^q\}_{q \in Q}$ (each realization will be called a *scenario*). The scenario approximation is then:

$$\text{Minimize} \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{10}$$

$$\text{subject to} \sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{11}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N,\ k \in K \tag{12}$$

$$G(\mathbf{x}, \mathbf{w}^q) \leq b \quad \forall q \in Q, \tag{13}$$

where the probabilistic constraint is replaced by the deterministic requirement that there must be a feasible edge-to-subgraph assignment for each scenario. As mentioned before we refer to the above problem as the stochastic edge partition problem. The following result, which follows from the general results in Luedtke and Ahmed (2008), provides justification for considering the scenario approximation.

Proposition 1. Let a desired confidence level $\delta \in (0, 1)$ be given. If the sample size $|Q|$ satisfies

$$|Q| \geq \frac{1}{\epsilon}\left[|N||K| \ln 2 - \ln \delta\right] \tag{14}$$

then any feasible solution to the stochastic edge partition problem (10)–(13) is feasible to the probabilistic edge partition problem (1)–(4) with probability at least $1 - \delta$.

Proof. Let X denote the set of solutions satisfying the deterministic constraints (2) and (3), let X^ϵ denote the set of feasible solutions to the probabilistic edge partition problem (satisfying (2)–(4)), and let X^Q denote the set of feasible solutions to the stochastic edge partition problem corresponding to a sample Q (satisfying (11)–(13)). We want to bound $|Q|$ such that $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1-\delta$.

Consider a solution $\mathbf{x} \in X \setminus X^\epsilon$, i.e., $\Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} < 1-\epsilon$. Then $\mathbf{x} \in X^Q$ if and only if $G(\mathbf{x}, \mathbf{w}^q) \leq b$ for all $q \in Q$. Since the $\mathbf{w}^q$ for $q \in Q$ are i.i.d. it follows that $\Pr\{\mathbf{x} \in X^Q\} \leq (1-\epsilon)^{|Q|}$. Now

$$\begin{aligned} \Pr\{X^Q \not\subseteq X^\epsilon\} &= \Pr\{\exists\ \mathbf{x} \in X^Q \text{ s.t. } \Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} < 1-\epsilon\} \\ &\leq \textstyle\sum_{\mathbf{x} \in X \setminus X^\epsilon} \Pr\{\mathbf{x} \in X^Q\} \leq |X \setminus X^\epsilon|(1-\epsilon)^{|Q|} \leq |X|(1-\epsilon)^{|Q|}. \end{aligned}$$

Thus $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1 - |X|(1-\epsilon)^{|Q|}$. To guarantee that $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1-\delta$ we need $|X|(1-\epsilon)^{|Q|} \leq \delta$ or equivalently

$$|Q| \geq \left[\ln|X| - \ln\delta\right] \Big/ \ln\left(\frac{1}{1-\epsilon}\right).$$

The claimed bound then follows by noting that $|X| \leq 2^{|N||K|}$ and $\ln(1/(1-\epsilon)) \geq \epsilon$. □

The above result suggests that we can obtain feasible solutions to the probabilistically constrained edge-partition problem by solving the stochastic edge partition problem with a "not too large" number of scenarios. Key to this sampling-based approach is the ability to efficiently solve stochastic edge partition instances having a modest number of scenarios, which is the motivation of this paper.

Next we describe an *extensive form* model of the stochastic edge partition problem. Let E^q be the set of edges with non-zero weights under scenario q. We define binary decision variables $y^q_{ijk} = 1$ if edge (i,j) is assigned to subgraph k in scenario q and 0 otherwise, $\forall q \in Q$, $(i,j) \in E^q$, and $k \in K$. The stochastic edge-partition problem can then be formulated as follows:

$$\text{Minimize} \quad \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{15}$$

$$\text{subject to} \quad \sum_{k \in K} y^q_{ijk} = 1 \quad \forall q \in Q,\ (i,j) \in E^q \tag{16}$$

$$\sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{17}$$

$$\sum_{(i,j) \in E^q} w^q_{ij} y^q_{ijk} \leq b \quad \forall q \in Q,\ k \in K \tag{18}$$

$$y^q_{ijk} \leq x_{ik},\ y^q_{ijk} \leq x_{jk} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K \tag{19}$$

$$x_{ik} \in \{0,1\} \quad \forall i \in N,\ k \in K \tag{20}$$

$$y^q_{ijk} \in \{0,1\} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K. \tag{21}$$

Observe that if one were to solve the above extensive form problem given by (15)–(21), integrality restrictions need only be imposed on the y-variables, which would, in turn, enforce the integrality of the x-variables at optimality. Note, also, that given a fixed set of x-values, this problem decomposes into $|Q|$ separable integer programs, where the subproblem corresponding to scenario $q \in Q$ is given by:

$$S^q(x) = \text{Maximize } 0 \tag{22}$$

$$\text{subject to (16), (18), (19), and (21).} \tag{23}$$

Under the foregoing model, it is useful to define $v_{ijk} = \min\{x_{ik}, x_{jk}\}$ as a part of the first-stage decision variables, $\forall (i,j) \in E,\ k \in K$. The presence of these variables will allow us to formulate stronger cutting planes than would be possible with just x-variables (see also Smith et al. (2004)). Assuming that $\cup_{q \in Q} E^q = E$ the extensive form problem is now equivalent to:

$$\text{Minimize} \quad \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{24}$$

$$\text{subject to} \quad \sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{25}$$

$$v_{ijk} \leq x_{ik},\ v_{ijk} \leq x_{jk} \quad \forall (i,j) \in E,\ k \in K \tag{26}$$

$$\sum_{k \in K} v_{ijk} \geq 1 \quad \forall (i,j) \in E \tag{27}$$

$$x_{ik} \in \{0,1\} \quad \forall i \in N,\ k \in K \tag{28}$$

$$F^q(\mathbf{v}) \leq b \qquad \forall q \in Q, \tag{29}$$

where

$$F^q(\mathbf{v}) = \quad \text{Minimize} \quad \max_{k \in K} \{ \sum_{(i,j) \in E^q} w^q_{ij} y^q_{ijk} \} \tag{30}$$

$$\text{subject to} \quad \sum_{k \in K} y^q_{ijk} = 1 \quad \forall (i,j) \in E^q \tag{31}$$

$$y^q_{ijk} \leq v_{ijk} \quad \forall (i,j) \in E^q,\ k \in K \tag{32}$$

$$y^q_{ijk} \in \{0,1\} \quad \forall (i,j) \in E^q,\ k \in K. \tag{33}$$

The valid inequalities (27) require that for each edge $(i,j) \in E$, both i and j must be assigned to some common subgraph, and will be useful in improving the computational efficacy of the decomposition algorithm that we propose. Note that an optimal solution exists in which $v_{ijk} = \min\{x_{ik}, x_{jk}\},\ \forall (i,j) \in E,\ k \in K$, without enforcing integrality restrictions or lower bounds on the v-variables.

There can exist up to $|K|! - 1$ alternative optimal solutions to this problem by simply reindexing the subgraph indices. These symmetric solutions are known to impede the performance of branch-and-bound algorithms (Sherali et al., 2000; Sherali and Smith, 2001). To reduce model symmetry we can rewrite the cardinality constraints (25) (or (17) for the extensive form problem) by using the following inequalities:

$$r \geq \sum_{i \in N} x_{i1} \geq \sum_{i \in N} x_{i2} \geq \cdots \geq \sum_{i \in N} x_{i|K|}. \tag{34}$$

For a scenario q and a given vector $\hat{\mathbf{v}}$, the problem (30)–(33) is essentially an identical parallel machine scheduling problem to minimize makespan ($P/\ /C_{max}$) (with some assignment restrictions). In particular, there would be $|K|$ machines and $|E^q|$ jobs, whose processing times are given by $w^q_{ij},\ \forall (i,j) \in E^q$. Each job must be assigned to exactly one machine, and the v-variables impose some restrictions on the assignments. The integer programming scheme developed in Smith (2004) is tailored for a similar problem in which the (weighted) number of demands that cannot be placed on one of these subgraphs is minimized (i.e., minimum weighted number of tardy jobs). This is not equivalent to solving a minimum makespan problem; however, the optimal solution of $F^q(\hat{\mathbf{v}})$ is no more than b if and only if the minimum number of tardy jobs is equal to 0. If a positive lower bound to the problem of minimizing the number of tardy jobs is established, one can terminate the subproblem algorithm and conclude infeasibility.

We now present a cutting plane algorithm for solving (24)–(29). The scheme relaxes constraints (29) and adds cutting planes as necessary to enforce feasibility to the subproblems. Let us call the problem (24)–(28) the master problem (MP).

1. Solve MP. If MP is infeasible then STOP; the problem is infeasible. Otherwise let $\hat{\mathbf{v}}$ be an optimal solution of MP.
2. For $q \in Q$, compute $F^q(\hat{\mathbf{v}})$. If $F^q(\hat{\mathbf{v}}) \leq b$ for all q, then STOP; the current solution is optimal. Otherwise, continue to Step 3.

3. Update MP by adding a cutting plane of the form (37) as presented in Remark 1, and return to Step 1.

After a finite number of steps, the cutting plane algorithm will terminate with an optimal solution, or will detect infeasibility.

Remark 1. Suppose $F^{\hat{q}}(\hat{\mathbf{v}}) > b$ for some scenario $\hat{q}$ and a solution vector $\hat{\mathbf{v}}$ to MP. Let $L^{\hat{q}}$ be a global lower bound on $F^{\hat{q}}(v)$, i.e., $L^{\hat{q}} \leq F^{\hat{q}}(v)$ for all v. Also define $I(\hat{\mathbf{v}}) = \{(ijk) \ : \ \hat{v}_{ijk} = 1\}$ and $O(\hat{\mathbf{v}}) = \{(ijk) \ : \ \hat{v}_{ijk} = 0\}$. The integer optimality cut proposed by Laporte and Louveaux (1993) for this class of problems is given by

$$(F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}}) \left[\sum_{(ijk)\in I(\hat{\mathbf{v}})} v_{ijk} - \sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} \right] \leq b + (F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}})(|I(\hat{\mathbf{v}})| - 1) - L^{\hat{q}}, \qquad (35)$$

Since $L^{\hat{q}} \leq b$ for any feasible instance, and since $F^{\hat{q}}(\hat{\mathbf{v}}) > b$ by assumption, we can apply Chvátal rounding to (35) by dividing both sides by $(F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}})$ and rounding down to obtain

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} + \sum_{(ijk)\in I(\hat{\mathbf{v}})} (1 - v_{ijk}) \geq 1. \qquad (36)$$

However, the following inequality is also a valid cutting plane that dominates (36):

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} \geq 1. \qquad (37)$$

To see that (37) is valid, consider a solution v' that does not satisfy the above inequality, i.e., $v'_{ijk} = 0$ for all $(ijk) \in O(\hat{\mathbf{v}})$. Therefore, $v'_{ijk} \leq \hat{v}_{ijk}$ for all (ijk). Then $F^q(v') \geq F^q(\hat{\mathbf{v}}) > b$, and v' is not feasible. Inequality (37) dominates (36) since the left-hand-side of (37) is not more than that of (36), and the right-hand-sides are both equal to 1. Thus, (37) serves as a cutting plane that can be used in Step 3 of the above algorithm. □

Another reformulation of our subproblem might admit stronger cutting planes than the ones of the form (37). In the parlance of machine scheduling, instead of trying to minimize the maximum makespan, we may wish to minimize the total sum of tardiness. Let c_k, $\forall k \in K$ be a nonnegative variable that denotes the amount of capacity deficit in subgraph k. Then, the problem of minimizing the total capacity deficit can be formulated as problem $\mathrm{MT}^q(\mathbf{v})$ below:

$$\mathbf{MT}^q(\mathbf{v}) : \ T^q(\mathbf{v}) = \text{Minimize} \ \sum_{k\in K} c_k \qquad (38)$$

$$\text{subject to} \quad \sum_{k \in K} y^q_{ijk} = 1 \quad \forall (i,j) \in E^q \tag{39}$$

$$y^q_{ijk} \leq v_{ijk} \quad \forall (i,j) \in E^q,\ k \in K \tag{40}$$

$$c_k \geq \sum_{(i,j) \in E^q} w^q_{ij} y^q_{ijk} - b \quad \forall k \in K \tag{41}$$

$$c_k \geq 0 \ \ \forall k \in K \tag{42}$$

$$y^q_{ijk} \in \{0,1\} \ \ \forall (i,j) \in E^q,\ k \in K. \tag{43}$$

Clearly, $F^q(\mathbf{v}) \leq b$ if and only if $T^q(\mathbf{v}) = 0$, and so we can replace master problem constraints (29) with the restrictions that $T^q(\mathbf{v}) = 0$ for all scenarios $q \in Q$. If subproblems $T^q(\mathbf{v})$ are used in lieu of $F^q(\mathbf{v})$, we would obtain (36) (directly, this time) from Laporte and Louveaux's integer feasibility cut. However, we can state a stronger cutting plane for a solution vector $\hat{\mathbf{v}}$ having $T^{\hat{q}}(\hat{\mathbf{v}}) > 0$ for some scenario $\hat{q}$, by requiring that the total amount of additional capacity that must be allocated to the collection of subgraphs is at least $T^{\hat{q}}(\hat{\mathbf{v}})$. This inequality is formally stated in the following proposition.

Proposition 2. Suppose for some solution vector $\hat{\mathbf{v}}$ and for some scenario $\hat{q} \in Q$, we obtain a lower bound $LB^{\hat{q}}(\hat{\mathbf{v}}) > 0$ for $T^{\hat{q}}(\hat{\mathbf{v}})$. Then the following inequality is a valid cutting plane for problem MP, and is at least as strong as (37):

$$\sum_{(ijk) \in O(\hat{\mathbf{v}})} \min\{w^{\hat{q}}_{ij}, LB^{\hat{q}}(\hat{\mathbf{v}})\} v_{ijk} \geq LB^{\hat{q}}(\hat{\mathbf{v}}). \tag{44}$$

Proof. Suppose, by contradiction, that there exists a binary vector $\mathbf{v}^*$ such that $T^{\hat{q}}(\mathbf{v}^*) = 0$, but $\sum_{(ijk) \in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk} < LB^{\hat{q}}(\hat{\mathbf{v}})$. Then there exists a solution $(\mathbf{y}^*, \mathbf{c}^*)$ to $\text{MT}^{\hat{q}}(\mathbf{v}^*)$ having $c^*_k = 0\ \forall k \in K$. We will show that the existence of such a $\mathbf{v}^*$ contradicts the assumption that $LB^{\hat{q}}(\hat{\mathbf{v}})$ is a valid lower bound on $T^{\hat{q}}(\hat{\mathbf{v}})$. We now build a solution $(\hat{\mathbf{y}}, \hat{\mathbf{c}})$ to $\text{MT}^{\hat{q}}(\hat{\mathbf{v}})$. First, we construct $\hat{\mathbf{y}}$ as follows:

1. For $(i,j) \in E^{\hat{q}}$, if $y^*_{ijk} = 1$ and $\hat{v}_{ijk} = 1$, then set $\hat{y}_{ijk} = 1$ as well.
2. For $(i,j) \in E^{\hat{q}}$, if $y^*_{ijk} = 1$ and $\hat{v}_{ijk} = 0$, then set $\hat{y}_{ij\hat{k}} = 1$ for any $\hat{k} \in K$ for which $(ij\hat{k}) \in I(\hat{\mathbf{v}})$. (Note that $(ijk) \in O(\hat{\mathbf{v}})$ since $\hat{v}_{ijk} = 0$.)
3. Set all other $\hat{y}_{ijk} = 0$.

In other words, $\hat{\mathbf{y}}$ is constructed in two phases. In the first phase, we ensure that if edge (i,j) was assigned to subgraph k in solution y^*, then (i,j) is assigned to k in $\hat{y}$ as well,

unless $\hat{v}_{ijk} = 0$ (prohibiting this assignment). In the second phase, if $y^*_{ijk} = 1$ but $\hat{v}_{ijk} = 0$, then we assign (i, j) to any $\hat{k}$ such that $\hat{v}_{ij\hat{k}} = 1$. Note that this assignment results in a solution feasible to (39), (40), and (43). Next, let us construct $\hat{\mathbf{c}}$. Observe that in the first phase of assigning edges to subgraphs based on $(ijk) \in I(\hat{\mathbf{v}})$ for which $y^*_{ijk} = 1$, no subgraph capacities are violated since $c^*_k = 0$, $\forall k \in K$, and so we initialize $\hat{c}_k = 0$, $\forall k \in K$. In the second phase, we guarantee feasibility to (41) (and maintain feasibility to (42)) by increasing $\hat{c}_{\hat{k}}$ by $w^{\hat{q}}_{ij}$. Thus $(\hat{\mathbf{y}}, \hat{\mathbf{c}})$ is a feasible solution to $\mathrm{MT}^q(\hat{\mathbf{v}})$.

At the end of the second phase of assignments, we have $\sum_{k\in K} \hat{c}_k = \sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk}$, since $\sum_{k\in K} \hat{c}_k$ is increased by $w^{\hat{q}}_{ij}$ only when both $v^*_{ijk} = 1$ and $(ijk) \in O(\hat{\mathbf{v}})$. However, by assumption, we have that $\sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk} < LB^{\hat{q}}(\hat{\mathbf{v}})$. Since $\sum_{k\in K} \hat{c}_k = \sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk}$, we have that $\sum_{k\in K} \hat{c}_k < LB^{\hat{q}}(\hat{\mathbf{v}})$, which contradicts the fact that $LB^{\hat{q}}(\hat{\mathbf{v}}) \leq T^{\hat{q}}(\hat{\mathbf{v}})$. Therefore, all feasible solutions must obey the inequality

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v_{ijk} \geq LB^{\hat{q}}(\hat{\mathbf{v}}),$$

from which (44) is readily derived. Finally, by dividing both sides of (44) by $LB^{\hat{q}}(\hat{\mathbf{v}})$, we see that (44) implies (37). □

Remark 2. In cutting plane implementations based on (37), once any scenario $\hat{q}$ is found such that the current $\hat{\mathbf{v}}$ vector is proven to be infeasible with respect to scenario $\hat{q}$, a cutting plane is generated and the master problem is re-solved. No further scenarios are tested, since an identical cut would be generated for each infeasible scenario. However, a cutting plane implementation based on problem (38)–(43) above with cutting planes (44) might benefit from deriving multiple cuts for each infeasible scenario, since these cuts could be distinct. □

Remark 3. Smith et al. (2004) explore the inclusion of "warming constraints" in the master problem, which enforce simple necessary conditions for feasibility to SONET problems. Denote the degree of node $i \in N$ by $\deg(i)$, and the set of nodes adjacent to i by $A(i)$. Lee et al. (2000) show that node i must be assigned to at least $\left\lceil \frac{\deg(i)}{r-1} \right\rceil$ subgraphs, since otherwise, more than r nodes would be assigned to some subgraph. Similarly, for scenario $q \in Q$, the total weight associated with node $i \in N$ is given by $\sum_{j\in A(i)} w^q_{ij}$. Since the total weight that can be assigned to a subgraph is limited by b, $\left\lceil \frac{\sum_{j\in A(i)} w^q_{ij}}{b} \right\rceil$ is a lower bound on

the number of copies of node i. We can then compute

$$\ell_i = \max\left\{\left\lceil\frac{\deg(i)}{r-1}\right\rceil, \max_{q\in Q}\left\lceil\frac{\sum_{j\in A(i)} w_{ij}^q}{b}\right\rceil\right\}, \tag{45}$$

and impose the following valid inequalities in the master problem:

$$\sum_{k\in K} x_{ik} \geq \ell_i \quad \forall i \in N. \tag{46}$$

Let $\hat{\imath}$ denote a node having the largest lower bound, so that $\ell_{\hat{\imath}} \geq \ell_i \ \forall i \in N$. Node $\hat{\imath}$ can be assigned arbitrarily to subgraphs $1, \ldots, \ell_{\hat{\imath}}$, and we fix $x_{\hat{\imath}1} = x_{\hat{\imath}2} = \cdots = x_{\hat{\imath}\ell_{\hat{\imath}}} = 1$ accordingly. Note that the symmetry-breaking constraints (34) need to be adjusted so that they are enforced separately for subgraphs $1, \ldots, \ell_{\hat{\imath}}$, and $\ell_{\hat{\imath}} + 1, \ldots, |K|$. Sherali et al. (2000) show computationally that such a variable-fixing scheme improves solvability of problem instances.

Smith et al. (2004) note that a node i cannot be assigned to a subgraph k in an optimal solution unless an adjacent node is also assigned to the same subgraph. Therefore, we also include the following constraints in MP:

$$x_{ik} \leq \sum_{j\in A(i)} x_{jk} \quad \forall i \in N,\ k \in K. \tag{47}$$

Smith (2005) describes valid inequalities that can be derived by analyzing the topology of the graph. First, consider an edge $(i, j) \in E$ such that $\ell_i = \ell_j = 1$. Let $A(i, j) = A(i) \cup A(j) - \{i, j\}$ denote the set of distinct nodes that are adjacent to i or j. If $|A(i, j)| \geq r - 1$, then i or j must be assigned to at least two subgraphs. Similarly, we define $W^q(i, j) = \sum_{k\in A(i,j)}(w_{ik}^q + w_{jk}^q) + w_{ij}^q$, and note that if $W^q(i, j) > b$ for some $q \in Q$, then we cannot feasibly assign nodes i and j to a single subgraph. If $|A(i, j)| \geq r - 1$ or $W^q(i, j) > b$, then we state the following valid inequality:

$$\sum_{k\in K} x_{ik} + \sum_{k\in K} x_{jk} \geq 3. \tag{48}$$

Second, for each edge $(i, j) \in E$, suppose $deg(i) \geq r$, $deg(j) < r$, and $|A(i, j)| > 2r - 3$. Smith (2005) shows that nodes i and j collectively need to be assigned to at least four subgraphs, which we state as:

$$\sum_{k\in K} x_{ik} + \sum_{k\in K} x_{jk} \geq 4. \tag{49}$$

3 A Hybrid IP/CP Approach

The cutting plane algorithms presented in Section 2 are preferable to solving stochastic edge-partition instances by the extensive form problem given by (15)–(21), as we will show in Section 4. However, the two-stage cutting plane algorithms still suffer from several computational difficulties. First, the master problem, MP, contains $|N||K|$ binary variables, $|E||K|$ continuous variables, and $O(|E||K|)$ constraints, which results in large integer programs. Second, the linear programming relaxation of MP is quite weak for many problem instances. Furthermore, the lower bound improves slowly as cuts of the type (37) or (44) are added to MP in each iteration. The main reason for this slow convergence is the existence of symmetry in MP. Inequalities (34) reduce, but do not completely eliminate, symmetric solutions in MP. Therefore when a solution of MP is found to be infeasible to a subproblem, MP often simply switches to a symmetric solution having the same objective function value. On the other hand, stronger anti-symmetry constraints tend to make MP very difficult to solve.

In this section we develop a new decomposition framework in order to remedy these difficulties. We combat symmetry due to reshuffling of subgraphs by representing subgraphs as *configurations*. A configuration c is identified by a subgraph node set N_c (we allow $N_c = \emptyset$) and a positive integer α_c, which gives the number of subgraphs having node set N_c. A solution is represented by a *configuration multiset* C whose elements are pairs (N_c, α_c). We will eliminate symmetry by ensuring that no isomorphic configuration multisets (i.e., those that are identical after reindexing configuration indices) are encountered in our search.

A configuration multiset C satisfies the following necessary feasibility conditions.

F1: $\sum_{c \in C} \alpha_c = |K|$ (partitions E into $|K|$ subgraphs)

F2: $|N_c| \leq r, \ \forall c \in C$ (no subgraph contains more than r nodes)

F3: $\forall (i,j) \in E, \ \exists c \in C$ such that $i \in N_c, \ j \in N_c$ (for each edge (i,j), there is at least one subgraph to which (i,j) can be assigned)

A multiset C that satisfies F1, F2, and F3 represents a feasible solution if all edges can be partitioned on the set of subgraphs corresponding to C without violating the weight restrictions for any scenario. Note that the number of distinct configurations in C, which we denote $|C|$, will be dynamically determined in our algorithm.

We now provide an overview of our three-stage hybrid algorithm.

1. The first-stage problem determines (via optimal solution of a mixed-integer program) the number of times we assign each node to the configurations in C. For instance, in the example given in Figure 1a, we could specify that we must use two copies of nodes 4 and 5, and one copy of the other nodes.
2. In the second stage, we seek a multiset C that uses exactly the number of node assignments specified in the first phase and satisfies F1, F2, and F3. In the example mentioned above, a multiset C having configurations $\{1, 2, 4\}$, $\{3, 4, 5\}$, and $\{5, 6\}$ (each with multiplicity one) could be generated based on the first-stage solution.
3. Finally, in the third stage, we determine whether C is feasible. If C is feasible then we stop with an optimal solution. Otherwise, we return to the second stage, and generate a different multiset meeting the stated criteria. If no such multiset exists, a cut is added to the first-stage problem, which is then re-solved. For the example given above, the multiset yields a feasible solution (see Figure 1b).

3.1 First-Stage Problem

For all $i \in N$, let z_i be an integer variable that represents the number of copies of node i to be used in forming configurations. We say that an $|N|$-dimensional vector $\mathbf{z}$ *induces* a multiset C if C contains exactly z_i copies of node i, $\forall i \in N$. The first-stage problem can succinctly be written as:

$$\text{Minimize} \quad \sum_{i \in N} z_i \tag{50}$$

$$\text{subject to} \quad \mathbf{z} \text{ induces a feasible multiset} \tag{51}$$

$$\ell_i \leq z_i \leq |K| \quad \forall i \in N \tag{52}$$

$$z_i \text{ integer}, \tag{53}$$

where ℓ_i is a lower bound on the number of copies required for node i, as given in (45). In order to formulate the first-stage problem as an integer program, we rewrite (51) as an exponential set of linear inequalities by considering the z-vectors that violate it. We first need to introduce auxiliary binary variables t_{ik}, $\forall i \in N$, $k = \ell_i, \ldots, |K|$, so that $t_{ik} = 1$ if $z_i = k$. Then, given a vector $\hat{\mathbf{z}}$ that does not induce a feasible multiset, we note that no $\bar{\mathbf{z}}$ such that $\bar{z}_i \leq \hat{z}_i$, $\forall i \in N$, induces a feasible multiset. Hence, at least one component of $\hat{\mathbf{z}}$

must be increased, and so

$$\sum_{i \in N} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1 \tag{54}$$

is a valid inequality. Our first-stage problem can now be expressed as the following integer program:

$$\text{Minimize} \quad \sum_{i \in N} z_i \tag{55}$$

$$\text{subject to} \quad z_i = \sum_{k=\ell_i}^{|K|} k t_{ik} \quad \forall i \in N \tag{56}$$

$$\sum_{k=\ell_i}^{|K|} t_{ik} = 1 \quad \forall i \in N \tag{57}$$

$$\sum_{i \in N} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1 \quad \forall \hat{\mathbf{z}} \in \mathcal{Z} \tag{58}$$

$$t_{ik} \text{ binary} \quad \forall i \in N,\ k = \ell_i, \ldots, |K|, \tag{59}$$

where $\mathcal{Z}$ is the set of all z-vectors that do not induce a feasible multiset. (The z-variables are in fact unnecessary in this formulation, but we keep them for ease of exposition.) In our algorithm we relax constraints (58) in the first-stage problem, and add them in a cutting-plane fashion. In every iteration we solve the first-stage problem to find $\hat{\mathbf{z}}$, and solve the second- and third-stage problems to seek a feasible multiset induced by $\hat{\mathbf{z}}$. If a feasible multiset is found, then $\hat{\mathbf{z}}$ induces an optimal solution and we stop. Otherwise, we add a cut of type (58), and re-solve the first-stage problem.

3.2 Second-Stage Problem

Our second-stage problem seeks a multiset induced by $\hat{\mathbf{z}}$ that satisfies F1, F2, and F3, using a constraint programming search. Given a set of constraints, a set of variables, and the domain of each variable (i.e., the set of values that each variable can take), constraint programming seeks a value assignment to each variable that satisfies all constraints. Constraints are propagated to reduce variable domains, which in turn trigger new constraint propagations. When no more domain reductions are possible, the algorithm searches for a solution by fixing a variable to a value in its domain, then recursively propagating constraints and reducing variable domains. If the domain of a variable becomes empty during constraint propagation,

then the algorithm backtracks. We refer the reader to Smith (1995), Lustig and Puget (2001), and Rossi et al. (2006) for a thorough discussion of constraint programming techniques.

3.2.1 Foundations

In our second-stage algorithm, a *solution* corresponds to a multiset C induced by $\hat{\mathbf{z}}$ that meets conditions F1, F2, and F3. In a solution each node i has a corresponding $|C|$-dimensional *distribution vector* β^i, which represents the number of copies of node i to be allocated to each existing configuration in C. Note that β^i_c cannot exceed α_c, and that $\sum_{c \in C} \beta^i_c = \hat{z}_i$. The *domain* of a node $i \in N$ is the set of possible β^i-vectors that i can take. We say that a node i is *processed* if we have selected its distribution vector β^i. A *partial multiset* is constructed by processing a subset of the nodes in N.

For instance, consider a five-node graph, and let the z-vector obtained by the first-stage problem be $\hat{\mathbf{z}} = (2, 3, 1, 4, 3)$. Suppose that nodes 1, 2, and 3 have been processed, and the following partial multiset with $|C| = 3$ has been obtained:

- $N_1 = \emptyset,\ \alpha_1 = 5$,
- $N_2 = \{1, 2\},\ \alpha_2 = 2$, and
- $N_3 = \{2, 3\},\ \alpha_3 = 1$.

Suppose that we process node 4 by choosing its distribution vector as $\hat{\beta}^4 = (2, 1, 1)$. Adding node 4 to two of the five copies of N_1 creates a new configuration N'_1 whose node set consists only of node 4 (with multiplicity two), and reduces the multiplicity of N_1 by two. After similarly adding one copy of node 4 to N_2 and one copy of node 4 to N_3, we obtain the following partial multiset with $|C'| = 5$:

- $N_1 = \emptyset,\ \alpha_1 = 3$ (reduced α_1),
- $N'_1 = \{4\},\ \alpha'_1 = 2$ (generated from configuration 1 by adding node 4 to N_1),
- $N_2 = \{1, 2\},\ \alpha_2 = 1$ (reduced α_2),
- $N'_2 = \{1, 2, 4\},\ \alpha'_2 = 1$ (generated from configuration 2 by adding node 4 to N_2), and
- $N_3 = \{2, 3, 4\},\ \alpha_3 = 1$ (added node 4 to N_3).

In general, when we process node i by choosing a distribution vector β^i, we update the partial multiset C as follows. For each configuration $c \in C$ if $\beta^i_c = 0$, then no changes are made to c (since no copies of node i are added to c). If $\beta^i_c = \alpha_c$, then we update configuration c by setting $N_c = N_c \cup \{i\}$. Finally, if $0 < \beta^i_c < \alpha_c$, then we create a new configuration c' having $N_{c'} = N_c \cup \{i\},\ \alpha_{c'} = \beta^i_c$, and update configuration c by setting $\alpha_c = \alpha_c - \beta^i_c$.

Remark 4. Recall that the configurations in a partial multiset C can be ordered in $|C|!$ symmetric ways. Our algorithm avoids this symmetry by generating only one such ordering after processing a node. Furthermore, the configuration multisets that we compute by processing node i according to the β^i-vectors in its domain must be pairwise nonisomorphic, since the β^i-values in the domain of node i are distinct. Hence, we never encounter isomorphic configuration multisets in the second-stage search. □

3.2.2 Domain Expansion

Processing a node modifies the current partial multiset, and therefore distribution vectors of the remaining unprocessed nodes need to be updated. Domains of nodes are reduced by constraint propagation as we will describe in the next section, but must also be expanded as new configurations are generated. We describe the initialization and expansion of node domains below.

In the beginning of the second stage we initialize our multiset C with a single configuration having $N_1 = \emptyset$ and $\alpha_1 = |K|$. Each node can only be added to the lone configuration, and so the domain for node i is initially the single one-dimensional vector $\beta^i = (\hat{z}_i)$. Our algorithm next processes some node $i \in N$, and updates the existing set of configurations: $N_1 = \emptyset,\ \alpha_1 = |K| - \hat{z}_i$ and $N_2 = \{i\},\ \alpha_2 = \hat{z}_i$. Next, the domains of all unprocessed nodes are updated to reflect the changes in C. For each unprocessed node j, we enumerate all possible ways of partitioning $\hat{z}_j$ copies into node sets N_1 and N_2. This logic is repeated at all future steps as well. For instance, in the example given above, suppose that $\hat{\beta}^5 = (2, 0, 1)$ was the only vector in the domain of node 5 before processing node 4. Since processing node 4 modifies the first configuration by reducing α_1 and generates a new configuration (N_1', α_1'), we expand the domain of node 5 by enumerating all possible ways of assigning $\hat{\beta}_1^5 = 2$ copies of node 5 to configurations (N_1, α_1) and (N_1', α_1'). On the other hand, since $\hat{\beta}^5$ does not assign node 5 to the second configuration, the distribution vectors in the expanded domain do not add node 5 to (N_2, α_2) or (N_2', α_2'). Finally, since processing node 4 does not generate any new configurations from the third configuration, all distribution vectors in the expanded domain of node 5 assign a single copy of node 5 to (N_3, α_3). After processing node 4 and updating the configurations as described above, the domain of node 5 is expanded to:

$$\{(2, 0, 0, 0, 1),\ (1, 1, 0, 0, 1),\ (0, 2, 0, 0, 1)\}.$$

3.2.3 Constraint Propagation

The constraints we impose in the second-stage problem limit the number of nodes in each configuration (F2) and require that each edge has both its end points in at least one configuration (F3). Condition F1 (requiring $|K|$ total configurations) will be implicitly satisfied. We apply constraint propagation algorithms to remove distribution vectors inconsistent with F2 or F3 from the expanded node domains. Let $i \in N$ be the last processed node, and let $C_i \subseteq C$ represent the subset of configurations to which node i has been added. We only need to execute constraint propagation for configurations $c \in C_i$, since these are the only newly modified configurations.

To enforce F2, the propagation algorithm identifies all configurations to which r nodes have been assigned. For each such configuration c, we remove all distribution vectors β^j having $\beta^j_c > 0$ from the domains of all unprocessed nodes $j \in N$. To enforce F3, the propagation algorithm iterates over the domains of the unprocessed nodes j adjacent to i, and removes all distribution vectors that do not add at least one copy of j to any configuration in C_i. Otherwise, the configurations containing node i would be disjoint from those containing node j, which violates F3.

3.2.4 Forward Checking

After all constraints are propagated, we first check whether the domain of any unprocessed node is empty; if so, then we backtrack. Else, we further analyze the current partial multiset before resuming the search with the next unprocessed node. This step identifies whether the current partial multiset can eventually yield a feasible multiset as early as possible to avoid performing unnecessary backtracking steps (van Beek, 2006).

We call one such test *implied node assignment analysis*. Suppose that we identify a processed node i such that $\hat{z}_i = 1$, and the configuration c to which i has been assigned. By condition F3 it follows that all unprocessed nodes j adjacent to i must also be assigned to configuration c. We use this analysis to augment partial configurations with implied node assignments, and then check whether any augmented configuration contains more than r nodes, and hence violates F2.

We also perform an *implied edge assignment analysis* by finding all edges that can only be assigned to a single configuration. For each $(i, j) \in E$, if both nodes i and j have been processed, then we check whether both i and j are in a single configuration c for which

$\alpha_c = 1$. In this case edge (i, j) can only be assigned to configuration c. On the other hand if (without loss of generality) node i has been processed but node j has not yet been processed, and $\hat{z}_i = 1$, then edge (i, j) can only be assigned to the configuration to which i has been assigned. After finding all implied edge assignments, we check whether F3 is violated for any scenario.

Finally, we consider a *singleton node analysis*, in which we ensure that each node is adjacent to at least one other node in each configuration. For each processed node i, and for all configurations $c \in C_i$, we seek a node j adjacent to i so that either $j \in N_c$ (if j also has been processed), or $\beta_c^j > 0$ for some distribution vector in the domain of j (if j has not been processed). If no such j can be found for a configuration $c \in C_i$, then the current partial solution cannot lead to an optimal solution; node i can ultimately be removed from configuration c without affecting feasibility conditions, leading to a reduction in the objective function value.

3.2.5 Node Selection Rule

The order in which variables are processed can significantly affect the performance of constraint programming algorithms (Smith, 1995; Lustig and Puget, 2001). Especially for infeasible second-stage problem instances, processing the "problematic" nodes first can quickly lead to the detection of infeasibility, and can result in significant savings in computational time. We employ a dynamic node selection rule in which the order of nodes considered can vary in different sections of the search tree. In accordance with the "fail-first" principle widely used in constraint programming algorithms (Haralick and Elliott, 1980; van Beek, 2006), our node selection rule first picks an unprocessed node that

1. has the fewest number of distribution vectors in its domain,
2. has the fewest number of copies to be partitioned, and
3. has the largest number of unprocessed adjacent nodes,

breaking ties in the given order. In this manner, we can quickly enumerate all possible distribution vectors of a few key nodes, allowing constraint propagation to quickly reduce the size of the remaining search space.

3.2.6 Distribution Vector Ordering Rule

Once the next node to be processed has been identified, all distribution vectors in its domain need to be tried, one by one, to see if any of them leads to a feasible multiset. For an infea-

sible second-stage problem instance, the order in which these vectors are instantiated does not matter, because all vectors must be enumerated before infeasibility can be concluded. However, for feasible problem instances it is important to find a vector that leads to a feasible multiset as soon as possible in order to curtail our search. Our ordering rule attempts to sort the distribution vectors in nonincreasing order of the likelihood that the vector leads to a feasible multiset. We calculate the feasibility likelihood score of a distribution vector β^i in the domain of an unprocessed node i with respect to a partial multiset C as:

$$FL(i, C, \beta^i) = \sum_{c \in C} \beta^i_c |\{j \in N_c : (i, j) \in E\}|. \tag{60}$$

$FL(i, C, \beta^i)$ measures the total number of adjacent node pairs (i, j) that would be added across all configurations if β^i is selected to be the distribution vector for node i. Our vector ordering rule sorts vectors in the domain of the chosen node in nondecreasing order of their FL-scores. By allowing for a higher degree of flexibility in assigning edges, we increase the likelihood that a feasible partition of edges to subgraphs can be found.

3.3 Third-Stage Problem

Given a solution of the second-stage problem that consists of a configuration multiset C satisfying F1, F2, and F3, the third-stage problem must verify whether C is feasible. We first generate the set of subgraphs from the multiset $C = \{c_1, c_2, \ldots, c_{|C|}\}$ by assigning the nodes in N_{c_1} to the first α_{c_1} subgraphs, then assigning the nodes in N_{c_2} to the next α_{c_2} subgraphs, and so on. Since we have enforced $\sum_{c \in C} \alpha_c = |K|$, this transformation creates exactly $|K|$ subgraphs, some of which can be empty. Then we iterate over all subgraphs and set $v_{ijk} = 1$ if nodes i and j are in subgraph k, and $v_{ijk} = 0$ otherwise. We then use formulation (38)–(43) to solve the third-stage problem.

Note that this transformation re-introduces symmetry into the third-stage problem. However, the solution of the third-stage problems does not constitute a bottleneck in the algorithm, and symmetry-breaking constraints appended to the transformed subproblem will not impact the computational efficacy of the overall algorithm.

3.4 Infeasibility Analysis

If a z-vector is found not to induce a feasible multiset, we add a constraint to the first-stage problem so that the same z-vector is not generated in subsequent iterations. Constraints

(58) state that the number of copies of some node must be increased, but they do not contain any information about *which* nodes need to be added. We observe that the progress of our second-stage algorithm can be analyzed to identify a "problematic" subset of nodes whose corresponding z-values cause infeasibility regardless of other variable values. Given a vector $\hat{\mathbf{z}}$ for which no feasible multiset exists, if a node $i \in N$ has not been processed, or has not been identified as the reason of infeasibility in any step of the backtracking algorithm, then $\hat{\mathbf{z}}$ will not induce a feasible multiset for any value of $\hat{z}_i$. Let $P \subseteq N$ denote the set of nodes that have been processed, or whose domains have become empty due to constraint propagation in the second-stage algorithm, possibly during different backtracking steps. The following is a valid inequality:

$$\sum_{i \in P} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1. \tag{61}$$

Constraints (61) clearly dominate (58) for any $P \subset N$, and get stronger as $|P|$ decreases. Based on this observation, we update our node selection rule by giving preference to selecting nodes that have already been added to P. Our revised node selection rule first picks a node that

0. has been added to P in a previous backtracking step,
1. has the fewest number of distribution vectors in its domain,
2. has the fewest number of copies to be partitioned, and
3. has the largest number of unprocessed adjacent nodes,

again breaking ties in the stated order.

3.5 Enhancements for the First-Stage Problem

Our computational studies revealed that the first-stage integer programming model solution represents the bottleneck operation of our algorithm. In order to decrease the computational time spent by the first-stage problem, we investigate several strategies.

3.5.1 Valid Inequalities

The valid inequalities that we discuss in Remark 3 can be adapted to the first-stage problem in order to eliminate the z-vectors that violate the corresponding necessary feasibility conditions. In particular, constraints (46) translate to simple lower bounds (52) on the z-variables.

Constraints (48), which are written for node pairs that satisfy the conditions discussed in Remark 3, can be written as:

$$z_i + z_j \geq 3. \tag{62}$$

Similarly, each constraint of type (49) can be equivalently represented as following:

$$z_i + z_j \geq 4. \tag{63}$$

Smith (2005) discusses an additional valid inequality, which cannot be represented using the x-variables in our two-stage algorithm, but can be written in terms of the z- and t-variables in the first-stage problem of our hybrid algorithm. For nodes $i \in N$ and $j \in N$, if $(i,j) \notin E, deg(i) \leq r-1, deg(j) \leq r-1,\ |A(i,j)| \geq r-1$, and there exists a common neighbor $k \in N$ so that $k \in A(i),\ k \in A(j),\ deg(k) \geq r$, and if i, j, k have more than $2r-4$ distinct neighbors in total, then $z_i = 1,\ z_j = 1$ implies $z_k \geq 3$. This condition can be written as:

$$z_k \geq -1 + 2(t_{i1} + t_{j1}), \tag{64}$$

which reduces to $z_k \geq 3$ for $z_i = z_j = 1$, and is redundant otherwise.

3.5.2 Heuristic for Obtaining an Initial Feasible Solution

The existence of a good initial feasible solution can help improve the performance of the first-stage problem because it provides a good upper bound, and allows the solver to apply strategies such as reduced cost fixing. We first solve the first-stage model enhanced with valid inequalities (62)–(64) to obtain an initial solution $\hat{\mathbf{z}}$, and execute the second- and third-stage algorithms to seek a feasible multiset. If one is found, we terminate with an optimal solution. Otherwise we investigate the set of processed nodes $\hat{P} \subseteq N$, and pick a node $\hat{\imath} \in \hat{P}$ having the fewest number of copies (breaking ties by picking a node having the largest degree). We then set $\hat{z}_{\hat{\imath}} = \hat{z}_{\hat{\imath}} + 1$, and re-invoke the second- and third-stage algorithms. This algorithm eventually finds a feasible multiset, or concludes that the entire problem is infeasible after generating the solution $\hat{z}_i = |K|,\ \forall i \in N$. We also generate a cut of type (61) for each $\hat{\mathbf{z}}$ generated before a feasible multiset is found, which we add to the first-stage problem in order to improve the lower bound.

3.5.3 Processing Integer Solutions

We can interrupt the branch-and-bound solution process of the first-stage problem each time the solver finds an integer solution $\hat{\mathbf{z}}$, and check whether $\hat{\mathbf{z}}$ induces a feasible multiset by

solving the second- and third-stage problems. If a feasible multiset exists, we accept $\hat{\mathbf{z}}$ as the new incumbent and resume solving the first-stage problem. Otherwise, we reject $\hat{\mathbf{z}}$, generate a constraint of type (61), and again resume the solution process. The same idea is also applicable to the master problem (MP) of the two-stage algorithm discussed in Section 2.

In our tests, this approach turned out to be more effective than solving the first-stage problem to optimality in each iteration, adding a cut, and re-solving it. The reason is that the problem is solved using a single branch-and-bound tree, which we tighten by adding cuts as necessary on integral nodes, instead of repeatedly generating a branch-and-bound tree in each iteration. It also allows us to obtain good feasible solutions for problem instances that are too difficult to solve to optimality.

We note that this approach requires a minor modification to the second-stage algorithm. All constraint propagation (Section 3.2.3) and forward checking rules (Section 3.2.4) except for singleton node analysis are based on necessary conditions for feasibility of configurations, and therefore they are valid for any integral $\hat{\mathbf{z}}$. However, singleton node analysis is based on an optimality condition, and hence can only be used if $\hat{\mathbf{z}}$ is a candidate optimal solution to the first-stage problem.

4 Computational Results

We implemented the algorithms discussed in the previous sections using CPLEX 11.1 running on a Windows XP PC with a 3.4 GHz CPU and 2 GB RAM. Our base set of test problem instances consists of 225 randomly generated problem instances for which the expected edge density of the graph (measured as $\frac{|E|}{|N|\times(|N|-1)}$) takes values 0.2, 0.3, and 0.4, the number of nodes ranges from 5 to 15, and the number of scenarios is between 1 (corresponding to the deterministic edge-partition problem) and 100. There is no practical limit on the number of subgraphs ($|K|$), but a limit needs to be specified to model the problem (see Sherali et al. (2000); Goldschmidt et al. (2003); Smith (2005)). Choosing $|K|$ too small may make the problem infeasible, and large values of $|K|$ increase difficulty of the problem. In our tests, we chose $|K|$ sufficiently large to yield a feasible edge partition in each problem instance. In generating instances we first picked a random subset of edges to have a positive weight, and then we assigned a weight uniformly distributed between 1 and 10 to each edge in each scenario. We generated five problem instances for each problem size, which is determined by the expected edge density, the number of nodes and the number of scenarios. The data

set names and details used in our experiments are given in Table 1.

Name	$\|N\|$	$\|K\|$	$\|Q\|$	r	b		Name	$\|N\|$	$\|K\|$	$\|Q\|$	r	b
5-1	5	5	1	4	20		12-1	12	10	1	5	50
5-30	5	5	30	4	20		12-30	12	10	30	5	50
5-100	5	5	100	4	20		12-100	12	10	100	5	50
8-1	8	7	1	4	35		15-1	15	10	1	8	70
8-30	8	7	30	4	35		15-30	15	10	30	8	70
8-100	8	7	100	4	35		15-100	15	10	100	8	70
10-1	10	8	1	5	40							
10-30	10	8	30	5	40							
10-100	10	8	100	5	40							

Table 1: Descriptions of the problem instances used for comparing algorithms

We used the default options of CPLEX for solving the extensive form problems. Preliminary computational experience on our two-stage algorithm indicated that the best implementation includes the valid inequalities (27), (46)–(49), and the symmetry-breaking constraints (34), and uses the model given by (38)–(43) for the subproblem, which is the formulation that minimizes the total tardiness. In our base setting for the three-stage algorithm, we used our heuristic to find an initial feasible solution, generated valid inequalities (62)–(64), and (similar to the two-stage algorithm) we used formulation (38)–(43) for the third-stage problem. We used callback functions of CPLEX to generate a single branch-and-bound tree for both two-stage and three-stage algorithms as discussed in Section 3.5. We imposed a half-hour (1800 seconds) time limit past which we halted the execution of an algorithm in all our experiments.

Our first experiment compares the performance of the extensive form, two-stage, and three-stage algorithms. Table 2 summarizes the results of these three algorithms on low density graphs having expected edge density 0.2. For each problem size, we report the following statistics calculated over five random instances: (i) the number of problems solved to optimality ("Solved"), (ii) the average optimality gap obtained at the root node ("Root Gap"), (iii) the average final optimality gap for instances that could not be solved within the allowed time limit ("Final Gap"), (iv) the average amount of time spent by each algorithm on the instances that were solved to optimality ("Time"). Out of the 75 instances in this data set, CPLEX could solve the extensive form to optimality for 61 instances, while both two-stage and three-stage algorithms solved all 75 instances to optimality within a few seconds. The results reveal that the performance of the extensive form formulation deteriorates rapidly as the number of scenarios increases, but the effect of the number of scenarios is mitigated for the two-stage and three-stage algorithms. We observe that the average optimality gap

obtained by the three-stage algorithm at the root node is 1.46%, which is significantly less than the initial gaps obtained using other approaches.

	Extensive Form				Two-Stage				Three-Stage			
Name	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time
5-1	5	0.00%	-	0.1	5	5.00%	-	0.1	5	0.00%	-	0.1
5-30	5	18.33%	-	6.6	5	4.00%	-	0.2	5	0.00%	-	0.1
5-100	5	12.38%	-	5.4	5	11.00%	-	0.6	5	0.00%	-	0.3
8-1	5	25.90%	-	0.4	5	6.67%	-	0.1	5	0.00%	-	0.1
8-30	5	12.89%	-	4.0	5	3.64%	-	0.2	5	0.00%	-	0.1
8-100	5	37.61%	-	223.1	5	14.84%	-	1.2	5	0.00%	-	0.3
10-1	5	19.58%	-	0.5	5	17.80%	-	0.4	5	0.00%	-	0.1
10-30	5	57.01%	-	147.3	5	10.71%	-	0.8	5	0.00%	-	0.2
10-100	4	30.35%	7.14%	684.1	5	13.94%	-	2.0	5	0.00%	-	0.4
12-1	5	47.25%	-	8.1	5	24.66%	-	2.2	5	0.00%	-	0.1
12-30	4	55.09%	25.00%	507.3	5	17.99%	-	4.3	5	3.08%	-	0.3
12-100	2	62.21%	24.88%	713.1	5	36.28%	-	4.7	5	2.11%	-	0.8
15-1	5	31.85%	-	33.0	5	64.38%	-	16.4	5	4.56%	-	0.2
15-30	1	65.29%	21.65%	864.6	5	39.13%	-	27.1	5	7.29%	-	0.6
15-100	0	57.33%	28.47%	-	5	24.49%	-	20.4	5	4.86%	-	1.2

Table 2: Comparison of the algorithms on graphs having edge density = 0.2

Tables 3 and 4 compare the three approaches on denser graphs having edge density 0.3 (medium density) and 0.4 (high density), respectively. We observe that performances of all three algorithms deteriorate as the edge density increases, which is not surprising due to the nature of the edge-partition problem. The number of instances that can be solved

	Extensive Form				Two-Stage				Three-Stage			
Name	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time
5-1	5	0.00%	-	0.1	5	2.86%	-	0.1	5	0.00%	-	0.1
5-30	5	25.76%	-	10.4	5	6.15%	-	0.5	5	0.00%	-	0.1
5-100	5	10.00%	-	3.1	5	10.77%	-	0.4	5	0.00%	-	0.2
8-1	5	31.30%	-	0.5	5	11.20%	-	0.1	5	0.00%	-	0.1
8-30	5	42.57%	-	18.0	5	7.48%	-	0.4	5	0.00%	-	0.2
8-100	4	39.37%	7.14%	110.0	5	16.19%	-	1.3	5	1.43%	-	0.3
10-1	5	32.42%	-	3.7	5	16.27%	-	0.6	5	1.18%	-	0.1
10-30	4	51.33%	21.05%	953.0	5	40.82%	-	8.2	5	5.83%	-	0.3
10-100	2	61.24%	29.05%	382.6	5	35.43%	-	302.7	5	8.89%	-	0.5
12-1	5	53.85%	-	312.0	5	39.49%	-	16.8	5	4.65%	-	0.2
12-30	0	63.41%	27.06%	-	5	46.98%	-	120.5	5	9.31%	-	0.8
12-100	0	84.24%	65.50%	-	4	42.78%	4.35%	89.0	5	11.99%	-	1.4
15-1	4	46.88%	11.54%	460.4	5	72.86%	-	250.5	5	12.93%	-	0.9
15-30	0	66.01%	42.41%	-	2	72.20%	16.02%	30.4	5	13.51%	-	3.5
15-100	0	80.76%	74.48%	-	0	53.05%	13.21%	-	5	16.31%	-	4.1

Table 3: Comparison of the algorithms on graphs having edge density = 0.3

by the extensive form decreases from 61 for low density graphs to 49 for medium density graphs, and finally to 36 for high-density graphs. The two-stage algorithm also exhibits a similar behavior; it can solve 75, 66, and 61 instances for low, medium, and high-density graphs, respectively. On the other hand, the three-stage algorithm is able to solve almost all instances, failing to solve two instances in the high-density 15-30 and 15-100 data sets to

optimality within the allowed time limit. Table 4 clearly shows that the three-stage algorithm dominates the other approaches, and the two-stage algorithm provides better results than directly solving the extensive formulation. Our analysis of optimal solutions obtained for the

	Extensive Form				Two-Stage				Three-Stage			
Name	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Solved**	**Root Gap**	**Final Gap**	**Time**
5-1	5	5.00%	-	0.1	5	0.00%	-	0.1	5	0.00%	-	0.1
5-30	5	24.67%	-	2.6	5	19.79%	-	0.3	5	0.00%	-	0.1
5-100	5	12.38%	-	5.6	5	8.31%	-	0.6	5	0.00%	-	0.2
8-1	5	41.32%	-	2.0	5	3.33%	-	0.1	5	0.00%	-	0.1
8-30	5	48.89%	-	140.9	5	17.68%	-	1.1	5	1.43%	-	0.1
8-100	3	47.23%	22.50%	113.0	5	21.08%	-	8.7	5	2.50%	-	0.4
10-1	5	45.08%	-	48.3	5	32.36%	-	3.5	5	2.16%	-	0.1
10-30	0	61.52%	20.64%	-	5	56.55%	-	39.5	5	8.45%	-	0.4
10-100	0	64.47%	50.91%	-	3	54.82%	7.50%	151.7	5	12.73%	-	1.5
12-1	1	67.13%	14.30%	33.2	5	40.60%	-	327.3	5	7.86%	-	0.5
12-30	0	88.61%	46.74%	-	5	42.93%	-	160.9	5	3.16%	-	0.8
12-100	0	84.37%	68.24%	-	5	51.54%	-	583.7	5	13.91%	-	1.7
15-1	2	60.11%	11.21%	369.6	3	53.01%	5.56%	410.0	5	11.57%	-	0.9
15-30	0	85.29%	65.29%	-	0	66.72%	22.66%	-	3	18.03%	4.74%	120.2
15-100	0	96.00%	86.92%	-	0	62.62%	24.58%	-	3	19.98%	6.45%	173.8

Table 4: Comparison of the algorithms on graphs having edge density = 0.4

problem instances shown in Tables 2–4 showed that the average objective function value for the deterministic (single-scenario) problem instances is 14.8. This value is smaller than the average objective function value for 30-scenario and 100-scenario instances (15.52 and 15.6, respectively). We also observe that several subgraphs can be empty in an optimal solution.

Our next experiment analyzes the performance of our three-stage algorithm for larger instances. For this experiment, we generated additional random problem instances using the parameter settings given in Table 5. Similar to our previous experiments, we generated

Name	$\|N\|$	$\|K\|$	r	b
5	5	5	4	20
8	8	7	4	35
10	10	8	5	40
12	12	10	5	50
15	15	10	8	70
17	17	10	8	100
20	20	10	10	120
22	22	10	10	140

Table 5: Descriptions of the problem instances used for analyzing the three-stage algorithm

problem instances for which the expected edge density of the graph takes values 0.2, 0.3, and 0.4. For each data set, we calculated the number of scenarios corresponding to $\epsilon, \delta = 0.05$ and $\epsilon, \delta = 0.01$ using Proposition 1. Hence, inequality (14) ensures that we can be 95% (99%, respectively) certain that all demands can be satisfied 95% (99%, respectively) of the time. We generated five random instances for each data set, resulting in 240 instances in

total. In addition to the columns given in Table 2, Tables 6, 7, and 8 show the relative gap between the quality of the solution found by our initial heuristic (Section 3.5.2) and the best lower bound obtained ("Heuristic Gap").

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**
5	407	5	0.00%	-	0.8	2.86%	2194	5	0.00%	-	3.5	0.00%
8	837	5	0.00%	-	2.2	1.54%	4343	5	0.00%	-	12.7	3.33%
10	1169	5	10.88%	-	5.6	5.09%	6006	5	1.33%	-	19.6	1.33%
12	1724	5	1.11%	-	13.2	4.19%	8779	5	3.00%	-	58.5	3.33%
15	2140	5	7.24%	-	22.1	2.74%	10858	5	12.41%	-	170.9	4.37%
17	2417	5	10.19%	-	41.0	4.78%	12245	5	9.82%	-	211.1	8.01%
20	2833	5	16.55%	-	79.8	7.01%	14324	5	12.40%	-	403.7	4.51%
22	3110	5	14.77%	-	128.2	6.49%	15710	5	15.78%	-	699.9	6.46%

Table 6: Three-Stage algorithm on graphs having edge density = 0.2

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**
5	407	5	0.00%	-	0.8	2.86%	2194	5	0.00%	-	3.5	0.00%
8	837	5	0.00%	-	2.6	2.86%	4343	5	3.33%	-	13.0	2.86%
10	1169	5	6.58%	-	7.5	8.99%	6006	5	11.86%	-	27.4	4.80%
12	1724	5	8.61%	-	16.1	4.51%	8779	5	8.22%	-	71.7	4.31%
15	2140	5	15.45%	-	45.1	3.05%	10858	4	18.53%	3.45%	176.4	4.25%
17	2417	5	13.63%	-	42.4	3.43%	12245	5	9.93%	-	189.3	2.68%
20	2833	5	17.47%	-	362.5	3.24%	14324	5	18.13%	-	639.1	3.32%
22	3110	4	16.18%	4.76%	159.3	5.82%	15710	5	15.86%	-	738.5	3.45%

Table 7: Three-Stage algorithm on graphs having edge density = 0.3

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**	$\lvert Q \rvert$	**Solved**	**Root Gap**	**Final Gap**	**Time**	**Heuristic Gap**
5	407	5	0.00%	-	0.8	2.22%	2194	5	0.00%	-	3.2	0.00%
8	837	5	7.71%	-	2.7	2.43%	4343	5	7.25%	-	17.3	5.33%
10	1169	5	16.38%	-	9.4	5.71%	6006	5	15.84%	-	52.1	9.73%
12	1724	5	16.71%	-	63.2	5.41%	8779	5	14.13%	-	118.4	5.45%
15	2417	4	16.24%	2.86%	338.8	4.07%	12245	2	16.55%	4.71%	549.8	6.86%
17	2140	1	23.46%	8.46%	993.7	9.23%	10858	1	24.77%	11.44%	1515.5	13.07%
20	2833	0	18.83%	9.46%	-	9.46%	14324	0	20.01%	11.30%	-	11.81%
22	3110	0	18.36%	11.05%	-	11.47%	15710	0	17.83%	9.90%	-	10.29%

Table 8: Three-Stage algorithm on graphs having edge density = 0.4

Our algorithm can solve 206 instances out of 240 to optimality, and provides an average optimality gap of 9.21% for the 34 instances that it cannot solve to optimality. The maximum optimality gap obtained for the entire data set is 21.22%. The results also suggest that our heuristic for finding an initial feasible solution is quite effective: the average optimality gap for our heuristic is 4.97%, and the maximum optimality gap is 22.72%. Since these calculations are based on the lower bounds obtained for the problem instances that could not be solved to optimality, our reported gaps possibly overestimate the true gap between heuristic and optimal objective values.

5 Conclusions

In this paper we considered the stochastic edge-partition problem, which arises in a telecommunication network design problem in Synchronous Optical Networks. We first developed an integer programming formulation of the problem, and prescribed a cutting plane algorithm with integer variables in both stages. Our computational tests showed that both the direct solution of the integer programming formulation and the execution of our cutting plane algorithm are capable of solving only small problem instances to optimality, especially for graphs having a high edge density. We then designed a hybrid integer programming/constraint programming algorithm to overcome the computational difficulties encountered by the first two approaches. Our hybrid approach first allocates node copies that are to be distributed across configurations using an integer programming formulation, and then assigns nodes to subgraphs using a constraint programming algorithm. After assigning nodes to subgraphs, it partitions edges to subgraphs for each scenario in a third stage, using another integer programming formulation. This decomposition of the problem also leads to the development of an effective heuristic, which we use to obtain initial upper bounds. Our computational experiments show that the hybrid approach significantly outperforms the other approaches we designed, and can handle thousands of scenarios unlike the other approaches. It can solve problem instances of relatively large dimensions to optimality, and can provide tight bounds on instances that cannot be solved to optimality within given time limits. Our algorithm can also be used to solve the deterministic (single-scenario) edge-partition problem more efficiently than the direct solution of the traditional integer programming formulation of the problem.

Acknowledgments. The authors are grateful to two anonymous referees, Osman Özaltın and Semra Ağralı, whose remarks helped improve the presentation of the paper. The authors gratefully acknowledge the support of the Air Force Office of Scientific Research under Grants F49620-03-1-0377 and FA9550-07-1-0404, and the support of the National Science Foundation under Grants CMII-0620780 and CMMI-0758234.

References

Bockmayr, A., Pisaruk, N., 2006. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. Computers & Operations Research 33/10,

2777–2786.

Calafiore, G. C., Campi, M. C., 2005. Uncertain convex programs: Randomized solutions and confidence levels. Mathematical Programming 102, 25–46.

Carøe, C. C., Schultz, R., 1999. Dual decomposition in stochastic integer programming. Operations Research Letters 24, 37–45.

Goldschmidt, O., Hochbaum, D. S., Levin, A., Olinick, E. V., 2003. The SONET edge-partition problem. Networks 41, 13–23.

Haralick, R. M., Elliott, G. L., 1980. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14/3, 263–313.

Hooker, J. N., 2007. Planning and scheduling by logic-based Benders decomposition. Operations Research 55, 588–602.

Hooker, J. N., Ottosson, G., 2003. Logic-based Benders decomposition. Mathematical Programming 96, 33–60.

Jain, V., Grossmann, I. E., 2001. Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS Journal on Computing 13, 258–276.

Laporte, G., Louveaux, F. V., 1993. The integer L-shaped method for stochastic integer programs with complete recourse. Operations Research Letters 13, 133–142.

Lee, Y., Sherali, H. D., Han, J., Kim, S., 2000. A branch-and-cut algorithm for solving an intra-ring synchronous optical network design problem. Networks 35 (3), 223–232.

Luedtke, J., Ahmed, S., 2008. A sample approximation approach for optimization with probabilistic constraints. SIAM Journal on Optimization 19, 674–699.

Lustig, I. J., Puget, J.-F., 2001. Program does not equal program: Constraint programming and its relationship to mathematical programming. Interfaces 31 (6), 29–53.

Nemirovski, A., Shapiro, A., 2005. Scenario approximation of chance constraints. In: Calafiore, G., Dabbene, F. (Eds.), Probabilistic and Randomized Methods for Design Under Uncertainty. Springer, London, pp. 3–48.

Rossi, F., van Beek, P., Walsh, T. (Eds.), 2006. Handbook of Constraint Programming. Elsevier.

Sen, S., Higle, J. L., 2005. The C^3 theorem and a D^2 algorithm for large scale stochastic integer programming: Set convexification. Mathematical Programming 104, 1–20.

Sherali, H. D., Fraticelli, B. M. P., 2002. A modification of Benders' decomposition algorithm for discrete subproblems: an approach for stochastic programs with integer recourse. Journal of Global Optimization 22, 319–342.

Sherali, H. D., Smith, J. C., 2001. Improving zero-one model representations via symmetry considerations. Management Science 47 (10), 1396–1407.

Sherali, H. D., Smith, J. C., Lee, Y., 2000. Enhanced model representations for an intra-ring synchronous optical network design problem allowing demand splitting. INFORMS Journal on Computing 12 (4), 284–298.

Smith, B. M., April 1995. A tutorial on constraint programming. Tech. Rep. 95.14, School of Computing, University of Leeds.

Smith, B. M., 2005. Symmetry and search in a network design problem. In: Barták, R., Milano, M. (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, pp. 336–350.

Smith, J. C., 2004. Algorithms for distributing telecommunication traffic on a multiple-ring SONET-based network. European Journal of Operational Research 154 (3), 659–672.

Smith, J. C., Schaefer, A. J., Yen, J. W., 2004. A stochastic integer programming approach to solving a synchronous optical network ring design problem. Networks 44 (1), 12–26.

Sutter, A., Vanderbeck, F., Wolsey, L. A., 1998. Optimal placement of add/drop multiplexers: Heuristic and exact algorithms. Operations Research 46 (5), 719–728.

Thorsteinsson, E. S., 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Walsh, T. (Ed.), CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming. Springer, pp. 16–30.

van Beek, P., 2006. Backtracking search algorithms. In: Handbook of Constraint Programming. Elsevier.