European Journal of Operational Research 296 (2022) 3–21

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Invited Review

Arc flow formulations based on dynamic programming: Theoretical foundations and applications

Vinícius L. de Lima[a,*], Cláudio Alves[b], François Clautiaux[c], Manuel Iori[d], José M. Valério de Carvalho[b]

[a] Instituto de Computação, Universidade Estadual de Campinas, Brazil
[b] Escola de Engenharia / Centro Algoritmi, Universidade do Minho, Portugal
[c] Université de Bordeaux, IMB UMR CNRS 5251, Inria Bordeaux Sud-Ouest, France
[d] DISMI, Università di Modena e Reggio Emilia, Italy

ARTICLE INFO

Article history:
Received 9 July 2020
Accepted 15 April 2021
Available online 24 April 2021

Keywords:
Combinatorial optimization
Arc flow
Dynamic programming
Acyclic network
Pseudo-polynomial

ABSTRACT

Network flow formulations are among the most successful tools to solve optimization problems. Such formulations correspond to determining an optimal flow in a network. One particular class of network flow formulations is the arc flow, where variables represent flows on individual arcs of the network. For $\mathcal{NP}$-hard problems, polynomial-sized arc flow models typically provide weak linear relaxations and may have too much symmetry to be efficient in practice. Instead, arc flow models with a pseudo-polynomial size usually provide strong relaxations and are efficient in practice. The interest in pseudo-polynomial arc flow formulations has grown considerably in the last twenty years, in which they have been used to solve many open instances of hard problems. A remarkable advantage of pseudo-polynomial arc flow models is the possibility to solve practical-sized instances directly by a Mixed Integer Linear Programming solver, avoiding the implementation of complex methods based on column generation.

In this survey, we present theoretical foundations of pseudo-polynomial arc flow formulations, by showing a relation between their network and Dynamic Programming (DP). This relation allows a better understanding of the strength of these formulations, through a link with models obtained by Dantzig-Wolfe decomposition. The relation with DP also allows a new perspective to relate state-space relaxation methods for DP with arc flow models. We also present a dual point of view to contrast the linear relaxation of arc flow models with that of models based on paths and cycles. To conclude, we review the main solution methods and applications of arc flow models based on DP in several domains such as cutting, packing, scheduling, and routing.

1. Introduction

Optimization problems can assume different characterizations, each allowing the reduction of the original problem to other optimization problems, leading to, possibly, different solution methods. One of such characterizations is based on a network, that is, a directed graph with costs on the arcs, and is the base of the well-known network flow problems (see, e.g., Ahuja, Magnanti, & Orlin, 1993). A *network flow problem* is an optimization problem that requires to determine an optimal flow on a network, by satisfying flow conservation on each node and possible additional side constraints on the flow on the arcs. The list of real-life problems that can be solved as network flow problems is extensive, including not only direct applications (i.e., the network is an input of the problem), as vehicle routing, telecommunication network planning, and train scheduling (see, e.g., Gouveia, Leitner, & Ruthmair, 2019, Minoux, 2006, and Cacchiani & Toth, 2012), but also indirect ones (i.e., the network must be constructed), as cutting and packing, scheduling, and project management (see, e.g., Delorme & Iori, 2020, Kramer, Iori, & Lacomme, 2021, and Riedler, Jatschka, Maschler, & Raidl, 2020).

In general, any formulation that corresponds to solving a network flow problem is called *network flow formulation*. Following Ahuja et al. (1993), the two main classes of network flow formulations are the path (and cycles) flow formulations and the arc flow formulations. *Path (and cycles) flow formulations* (called path flow formulations for short in the following) have variables

* Corresponding author.
E-mail addresses: v.loti@ic.unicamp.br (V.L. de Lima), claudio@dps.uminho.pt (C. Alves), francois.clautiaux@math.u-bordeaux.fr (F. Clautiaux), manuel.iori@unimore.it (M. Iori), vc@dps.uminho.pt (J.M. Valério de Carvalho).

https://doi.org/10.1016/j.ejor.2021.04.024
0377-2217/

corresponding to flow on paths and cycles of the network, and are frequently associated with set-covering, set-packing, or set-partitioning models. In contrast, *arc flow formulations* have variables corresponding to flow on individual arcs of the network. The Flow Decomposition Theorem by Ahuja et al. (1993) guarantees that the two formulations produce equivalent models when based on the same network, in the sense that any solution of one model can be mapped into a solution of the other model.

Arc flow formulations have a linear number of variables with respect to the number of arcs in the network. Often, such formulations can be used to solve medium-sized instances directly by a commercial Mixed Integer Linear Programming (MILP) solver, avoiding the implementation of complex methods. On the other hand, path flow formulations may have a much larger number of variables, as the number of paths and cycles can be exponential with respect to the number of arcs of the network. Such exponential path flow formulations are typically solved by sophisticated methods based on column generation and branch-and-price algorithms (see, e.g., Desaulniers, Desrosiers, and Solomon 2006 and Sadykov & Vanderbeck 2013).

Many $\mathcal{NP}$-hard problems can be formulated as compact (i.e., polynomial-sized) models. Although their size is favorable, these models are usually associated with weak linear relaxations and very symmetrical solution spaces, leading to an overly extensive enumeration on branch-and-bound algorithms. To overcome such inefficiency, one may rely on the Dantzig-Wolfe (DW) decomposition (see Dantzig & Wolfe 1961), which can lead to models with stronger linear relaxations and less symmetry. The models resulting from DW decomposition usually have exponentially more variables than the original model and are solved by column generation. When the associated pricing problem is solved by *Dynamic Programming* (DP), the model can be seen as a path flow formulation, where each variable corresponds to a path in the network inherent from the DP problem. From this observation, one can devise an equivalent arc flow model based on the DP network. Originated by a DW decomposition, the resulting arc flow model will, possibly, have a strong linear relaxation and a less symmetrical solution space, compared to the original polynomial-sized model.

To obtain strong models by a DW decomposition, one may have to pay the price of having an increased complexity on the pricing problem, as such complexity usually becomes pseudo-polynomial or exponential. When the size of the DP network from the pricing problem is pseudo-polynomial, the resulting arc flow model can still be used in practice to solve medium-sized instances by means of a MILP solver. On the other hand, when the DP network is too large (possibly exponential), one can obtain smaller (yet still strong) pseudo-polynomial arc flow models relying on a state-space relaxation of the DP. Different state-space relaxations lead to arc flow models with different sizes and different strength, allowing a flexible possibility to balance size and strength of arc flow models.

Seminal works and previous surveys

To the best of our knowledge, Ford and Fulkerson (1958b) were the first to propose a pseudo-polynomial arc flow model, which was based on a time-expanded network and was used to solve the maximal dynamic flow problem. Already in the sixties, Shapiro, (1968) studied the relation between DP and Integer Programming, by modeling the knapsack problem as a pseudo-polynomial network flow problem. More than ten years later, Wolsey (1977) proposed a general methodology to build packing and covering models from the set of feasible solutions of DP problems. As an application, he introduced a pseudo-polynomial arc flow model based on a DP network of the knapsack problem for the cutting stock problem (CSP), a strongly $\mathcal{NP}$-hard problem. Due to the context, Wolsey (1977) did not present computational experiments for the model, and such formulations did not get much attention for decades. It was more than twenty years later that Valério de Carvalho (1999) independently proposed this model, used it to solve to (integer) optimality open instances in the CSP literature, and showed that it is equivalent to the Gilmore-Gomory model (Gilmore & Gomory, 1961; 1963), a path flow model obtained from a DW decomposition.

Since the work by Valério de Carvalho (1999), the popularity of pseudo-polynomial arc flow formulations based on DP has grown considerably, and several different applications to $\mathcal{NP}$-hard problems have appeared in the literature. Large instances have been solved for the first time to proven optimality by arc flow models in several cutting, packing, and scheduling problems. In addition, arc flow formulations have effectively modeled complex problems involving, for instance, multiple-stage cutting processes or the presence of setups.

Some previous surveys considered arc flow models on specific applications (see, e.g., Valério de Carvalho, 2002 and Delorme, Iori, & Martello, 2016 for bin packing and cutting stock problems, and Cattaruzza, Absi, & Feillet, 2016 for vehicle routing problems with multiple trips).

Skutella (2009) presented a survey on problems based on dynamic flow, which is also called "flows over time". One of the main formulations for this class of problems is the arc flow model based on time-expanded networks, introduced by Ford and Fulkerson (1958b), which is derived from DP graphs of pseudo-polynomial size. A generalization of such networks is the layered graph approach which considers, for instance, capacity-indexed networks for vehicle routing problems. Modeling techniques and efficient solution methods for pseudo-polynomial arc flow formulations based on layered graphs were surveyed by Gouveia et al. (2019). Other surveys, such as that by Sadykov and Vanderbeck (2013), focused instead on how arc flow formulations can be used to devise effective branch-and-price algorithms. Conforti, Cornuéjols, and Zambelli (2010) and Lancia and Serafini (2014) studied techniques to transform large extended formulations into compact ones. In the case of network flow formulations, this occurs when path flow models are transformed into equivalent arc flow models with (possibly exponentially) fewer variables.

Contents

In this survey, we extend the previous studies to provide a base for some reasons behind the (primal) strength of pseudo-polynomial arc flow formulations. Based on the Flow Decomposition Theorem, we show how DW decomposition can derive arc flow models with a strong linear relaxation. We also discuss how the state-space relaxation method may allow one to obtain a balance between strength and complexity when constructing arc flow models. This relation between state-space relaxation and arc flow formulations can implicitly relate different arc flow models in the literature (see Section 4). We provide a dual insight that may explain arc flow models' practical computational efficiency over their equivalent path flow models, which is the richer description of the dual space. The relevance is that a better description of the dual space leads to less primal degeneracy, which is headwind for Linear Programming (LP) simplex (vertex) algorithms used in branch-and-bound techniques. We also discuss the main solution methods to solve large-scale arc flow models and present the main applications studied in the literature.

The remainder of this paper is organized as follows. Section 2 presents the theoretical foundations of network flow formulations and DP. The relationship between arc flow models with a strong relaxation and DW decomposition is presented in Section 3. In Section 4, we present a discussion on how the state-space relaxation can be used to obtain smaller arc flow models. In Section 5, we show how arc flow formulations present a more

detailed dual space than path flow formulations, which leads to a better convergence of the LP solution. The state-of-the-art methods to solve large-scale arc flow models and the main applications are shown in Sections 6 and 7, respectively. Finally, our concluding thoughts are presented in Section 8.

2. Network flow formulations and dynamic programming

A *network* $\mathcal{N}$ is composed of a directed graph $G = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of nodes (vertices) and $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of arcs, and of a cost function $c : \mathcal{A} \to \mathbb{R}$ that maps each arc (u, v) to a cost $c(u, v)$. In addition, a network has two special nodes in $\mathcal{V}$, called *source* (v_{source}) and *sink* (v_{sink}), such that no arcs in $\mathcal{A}$ enters v_{source} or leaves v_{sink}. Depending on the context, we may denote the set of nodes and the set of arcs of a network $\mathcal{N}$ as $\mathcal{V}(\mathcal{N})$ and $\mathcal{A}(\mathcal{N})$, respectively.

A *flow* in a network is a function $f : \mathcal{A} \to \mathbb{R}_+$ that satisfies the flow conservation constraints:

$$\sum_{(u,v)\in\mathcal{A}} f((u,v)) = \sum_{(v,w)\in\mathcal{A}} f((v,w)), \quad \forall v \in \mathcal{V} \setminus \{v_{source}, v_{sink}\}, \quad (1)$$

i.e., the flow entering a node is equal to the flow leaving it, except for v_{source} and v_{sink}, where there is only outgoing and incoming flow, respectively. By the flow conservation constraints, the flow leaving v_{source} is equal to the flow entering v_{sink}.

Definition 1. A network flow problem is an optimization problem which requires to determine a flow f that satisfies side constraints and optimizes $\sum_{(u,v)\in\mathcal{A}} c(u, v) f((u, v))$. In addition, any formulation that corresponds to solving a network flow problem is referred to as a network flow formulation.

A *path* in a graph is a sequence of arcs which joins a sequence of nodes. A *cycle* is a path in which the first and the last nodes are the same. We denote by $\mathcal{P}$ the set of all paths from v_{source} to v_{sink}, and by $\mathcal{A}(p)$ the set of arcs of a path $p \in \mathcal{P}$. The cost of a path $p \in \mathcal{P}$ is given by $\tilde{c}_p = \sum_{(u,v)\in\mathcal{A}(p)} c(u, v)$. In practice, the solution of network flow problems can be given as a set of paths and cycles with a positive flow or as the flow on each arc of the network. Following Ahuja et al. (1993), this gives rise to two different classes of network flow formulations: path flow formulations and arc flow formulations.

Definition 2. Path flow formulations are network flow formulations where decision variables correspond to the non-negative flow on each path and each cycle of the network.

Path flow formulations are also referred to in the literature as *path-based formulations*. In path flow formulations, the flow conservation is implicitly imposed on the variables. Since $|\mathcal{P}(\mathcal{N})|$ can be exponential with respect to $|\mathcal{A}(\mathcal{N})|$, path flow formulations usually have a huge number of variables. Nonetheless, such formulations have been successfully solved in the literature by column generation based algorithms (see, e.g., Poggi & Uchoa, 2014).

Definition 3. Arc flow formulations are network flow formulations where decision variables correspond to the non-negative flow on each arc of the network.

In contrast to path flow formulations, arc flow formulations impose the flow conservation explicitly as linear constraints. For instance, let variable $\varphi_{(u,v)} \in \mathbb{R}_+$, for each arc $(u, v) \in \mathcal{A}$, represent the flow on arc (u, v), and let variable $z \in \mathbb{R}_+$ represent the total flow on the network. Flow conservation constraints can be formulated as:

$$\sum_{(u,v)\in\mathcal{A}} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \quad \forall v \in \mathcal{V}. \quad (2)$$

The underlying matrix from (2) is totally unimodular (see, e.g., Nemhauser & Wolsey, 1988), which is an interesting property, as LP models with a totally unimodular constraint matrix have the *integrality property*, i.e., every vertex of the corresponding polytope is integer, implying the existence of an integer optimal solution if the right-hand side of the constraints is integer.

Although flow conservation constraints (2) must be explicitly included, differently from path flow formulations, arc flow formulations have a polynomial number of variables with respect to network size. An important result that relates these two classes of formulations is the following:

Theorem 1 (Flow Decomposition Theorem Ahuja et al., 1993)**.** *Every path and cycle flow has a unique representation as non-negative arc flows. Conversely, every non-negative arc flow can be represented as a path and cycle flow (though not necessarily uniquely) with the following two properties:*

(a) *Every directed path with a positive flow connects the source to the sink;*
(b) *At most $|\mathcal{V}| + |\mathcal{A}|$ paths and cycles have nonzero flow; out of these, at most $|\mathcal{A}|$ cycles have nonzero flow.*

Theorem 1 proves the equivalence between arc flow and path flow formulations: an arc flow solution can be transformed into positive flow on a set of paths and cycles, and a solution of path flow formulations can be decomposed into flow on individual arcs. Consequently, arc flow and path flow formulations based on the same network produce the same linear relaxation bounds for integer problems.

A fundamental network flow problem is the *longest path problem* (LPP): find a longest path, i.e., a unitary flow of maximum cost, from v_{source} to v_{sink}. When the network does not have cycles of negative cost, a longest path can be found in polynomial time with respect to the network size. In particular, when the network is acyclic, a longest path can be found in $O(|\mathcal{A}|)$ by a topological ordering of the nodes (see, e.g., Ahuja et al. 1993). The LPP often appears in solution methods for more complex network flow problems. Such problems have, for instance, side constraints characterized by generalized upper and lower bounds on the flow on subsets of arcs (see, e.g., Clautiaux, Hanafi, Macedo, Voge, & Alves 2017). In the remainder of this section, we formally define Dynamic Programming and its relation to acyclic network flow formulations and the LPP.

2.1. Dynamic programming and arc flow formulations

(Discrete) Dynamic Programming (DP) is a well-known method, proposed in the fifties (see, e.g., Bellman 1957), to solve combinatorial optimization problems that can be decomposed into a sequence of decisions, often represented by *stages*, each corresponding to a decision step. DP models are defined by a state space $\mathcal{S}$, where each state $s \in \mathcal{S}$ is characterized by a set of entities. Each state is defined, for instance, by the subset of clients already visited in routing problems (see, e.g., Desrochers, Desrosiers, & Solomon 1992) or by the level of usage of a given resource in, e.g., cutting, packing or scheduling problems (see, e.g., Christofides & Hadjiconstantinou 1995).

A key aspect in DP modeling is the fact that the description of a state $s \in \mathcal{S}$ should be sufficient to recognize the admissible decisions for s, which is coined as the *no-memory property of DP*. Then, when stages are considered, problems are divided into a sequence of sub-problems, so that the solution of a sub-problem depends only on the sub-problems from the previous stage. Nowadays, some authors changed the perspective of DP from the stage-dependent description to a table-filling method, as described by, e.g., Ahuja et al. (1993).

A DP model can be represented by a recursive function, in which the value of a state is only based on the values of the precedent states and the cost (contribution) of the decisions leading to that state. Formally, for each state $s \in \mathcal{S}$, let $\Delta(s)$ be the set of precedent states, and $c(r,s) \in \mathbb{R}$ be the cost required to move from $r \in \Delta(s)$ to s. The state space $\mathcal{S}$ contains two special states, s_0 and s^*, representing the initial condition of the recursion and the optimal solution of the problem, respectively. No state precedes s_0 ($\Delta(s_0) = \emptyset$) and s^* does not precede any other state ($s^* \notin \Delta(s), \forall s \in \mathcal{S}$). The DP recursion we consider is given by:

$$f_{DP}(s) = \begin{cases} \max_{\{r \in \Delta(s)\}} \{f_{DP}(r) + c(r,s)\}, & \text{if } s \neq s_0, \\ 0, & \text{if } s = s_0, \end{cases} \quad \forall s \in \mathcal{S}. \quad (3)$$

A priori, the DP recursion (3) denotes a maximization problem, but it can easily be considered as a minimization problem by inverting the sign of all decision costs. Based on recursion (3), a DP model can be defined on a directed acyclic graph (DAG), where vertices and arcs correspond to states and decisions, respectively. In this characterization, the DP solution is given by a longest path, that is, a path from s_0 to s^* with maximum total cost (profit). In some cases, the description of the recursion produces states that are not in any solution from the initial state to the goal state, and such states can be disregarded.

The previous characterization of DP is a classical definition, and one of the most used by the integer programming literature. Nevertheless, other definitions of DP were proposed. For instance, Martin, Rardin, & Campbell, (1990) defined DP over a directed acyclic hypergraph, to obtain a better polyhedral characterization of a class of combinatorial optimization problems. This generalization was later used to solve network flow problems (see, e.g., Clautiaux, Sadykov, Vanderbeck, & Viaud 2018).

Following our DP definition, the DAG from a DP model produces the *dynamic programming network* $\mathcal{N}_{DP}$. Each node is associated with a state, that is, $\mathcal{V}(\mathcal{N}_{DP}) \equiv \mathcal{S}$, where states s_0 and s^* correspond to nodes v_{source} and v_{sink}, respectively. The set of arcs is defined by $\mathcal{A}(\mathcal{N}_{DP}) = \{(r,s) \mid s \in \mathcal{S}, r \in \Delta(s)\}$, and the cost $c(r,s)$ of an arc $(r,s) \in \mathcal{A}$ is equivalent to the decision cost $c(r,s)$ for moving from state r to state s.

The fact that DP can be seen as a LPP in the DP network provides a generic recipe to transform a DP model into the arc flow model given by:

$$\max \sum_{(u,v) \in \mathcal{A}(\mathcal{N}_{DP})} c(u,v)\varphi_{(u,v)}, \quad (4)$$

$$\text{s.t.:} \sum_{(u,v) \in \mathcal{A}(\mathcal{N}_{DP})} \varphi_{(u,v)} - \sum_{(v,w) \in \mathcal{A}(\mathcal{N}_{DP})} \varphi_{(v,w)} = \begin{cases} -1, & \text{if } v = v_{source}, \\ 1, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \quad \forall v \in \mathcal{V}(\mathcal{N}_{DP}), \quad (5)$$

$$\varphi_{(u,v)} \in \{0,1\}, \quad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{DP}). \quad (6)$$

Model (4)–(6) considers flow conservation constraints with a unitary flow ($z = 1$) and side constraints impose that the flow on each arc is binary. The objective function is to maximize the cost of the selected path, given by the sum of the contributions of the state decisions. As this model has only flow conservation constraints, it has the integrality property, because, as previously discussed, its underlying matrix is totally unimodular. This property is linked with the fact that the LPP on a DAG can be solved in linear time with respect to the number of arcs (see, e.g., Bellman 1957).

In strongly $\mathcal{NP}$-hard problems, DP often provides pseudo-polynomial algorithms to solve sub-problems that determine partial plans. Examples of such sub-problems are: finding a single cutting/packing pattern on cutting and packing problems (see, e.g., Valério de Carvalho 2002); obtaining a schedule in a single machine in scheduling problems (see, e.g., Kramer et al. 2021); determining a route of a single vehicle on vehicle routing problems (see, e.g., Poggi & Uchoa 2014).

The DP network from the sub-problems (partial plans) may be used to derive arc flow models to solve the main problem (global plan). If the global plan is a strongly $\mathcal{NP}$-hard problem (which is often the case), these models do not have the integrality property, unless $\mathcal{P} = \mathcal{NP}$, but they usually provide strong relaxations. We provide examples of DP networks of two problems that are often associated with partial plans.

2.2. Example on the knapsack problem

Let us consider an example on the *knapsack problem* (KP): given a knapsack with capacity $W \in \mathbb{Z}_+$ and a set $I = \{1, 2, \ldots, n\}$ of items, each item $i \in I$ with a weight $w_i \in \mathbb{Z}_+$, a profit $p_i \in \mathbb{Z}_+$, and a maximum number of copies $d_i \in \mathbb{Z}_+$, determine the number x_i (with $0 \leq x_i \leq d_i$) of copies of each item $i \in I$ in the solution, so that $\sum_{i \in I} w_i x_i \leq W$ and $\sum_{i \in I} p_i x_i$ is maximum (see, e.g., Martello & Toth 1990). The optimal solution value $f_{KP}(i, W')$ of a knapsack sub-problem that considers the items $\{1, 2, \ldots, i\}$ and a partial capacity W' can be recursively computed by:

$$f_{KP}(i,W') = \begin{cases} \max_{l \in \{0,1,\ldots,\min\{d_i, \lfloor W'/w_i \rfloor\}\}} \{f_{KP}(i-1, W' - w_i l) + p_i l\}, & \text{if } i > 0 \text{ and } W' > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

The optimal KP solution value is given by $OPT(I,W) = \max_{\{0 < W' \leq W\}} f_{KP}(|I|, W')$ and it is associated with a DP model where the state space $\mathcal{S}$ is a subset of $\{(i,W') \mid i \in I, W' \in \{0,1,\ldots,W\}\} \cup \{s^*\}$, each partial set of items corresponds to a stage, the initial condition state is $s_0 = (0,0)$, and the goal is represented by a dummy state s^*, so that the precedent states in $\Delta(s^*)$ are related to the recursion of $OPT(I,W)$. The set of precedent states of each state $(i,W') \in \mathcal{S} \setminus \{s^*\}$, where $i > 0$, is $\Delta((i,W')) = \{(i-1, W' - w_i l) \mid w_i l \leq W', l \in \{0,1,\ldots,d_i\}\}$, and the decision cost for reaching (i,W') from a precedent state $(i-1, W' - w_i l) \in \Delta((i,W'))$ is $c((i-1, W' - w_i l), (i,W')) = p_i l$. The set of precedent states of s^* is $\Delta(s^*) = \{(|I|, W') \mid W' \in \{0,1,\ldots,W\}\}$ and the cost to move from a precedent state $(|I|, W') \in \Delta(s^*)$ to s^* is $c((|I|, W'), s^*) = 0$.

Let $\mathcal{N}_{KP}$ be a DP network for the KP. The set of nodes $\mathcal{V}(\mathcal{N}_{KP}) \equiv \mathcal{S}$ corresponds to the set of states, where v_{source} and v_{sink} are associated with s_0 and s^*, respectively. The set of arcs is $\mathcal{A}(\mathcal{N}_{KP}) = \cup_{\{i \in I, l \in \{0,1,\ldots,\min\{d_i, \lfloor W'/w_i \rfloor\}\}\}} \mathcal{A}_{il} \cup \mathcal{A}_S$. For each $i \in I$, the set $\mathcal{A}_{il} = \{((i-1, W' - w_i l), (i,W')) \mid (i,W') \in \mathcal{S}, (i-1, W' - w_i l) \in \Delta((i,W'))\}$ contains the arcs corresponding to the decision of choosing l copies of i in the solution, and these arcs have profit $p_i l$. The set $\mathcal{A}_S = \{((n,W'), v_{sink}) \mid (n,W') \in \mathcal{S}\}$ contains arcs, called *loss arcs*, that link the nodes from the last stage to v_{sink}, and their profit is 0.

As an example, Fig. 1 shows the network for a KP with capacity $W = 8$, and three items having $w_1 = 4$, $w_2 = 3$, $w_3 = 2$, $p_1 = 8$, $p_2 = 5$, $p_3 = 4$, and $d_1 = d_2 = d_3 = 1$. There is a node s^* representing v_{sink}, a node (i,W') for each state, and $v_{source} = (0,0)$. Arcs $\mathcal{A}_S$ are depicted as dotted links, and for every item $i \in I$, the arcs in $\mathcal{A}_{i1}$ and $\mathcal{A}_{i0}$ are depicted as full and dashed links, respectively. The arcs that do not belong to any path from (0,0) to s^* are disregarded. By

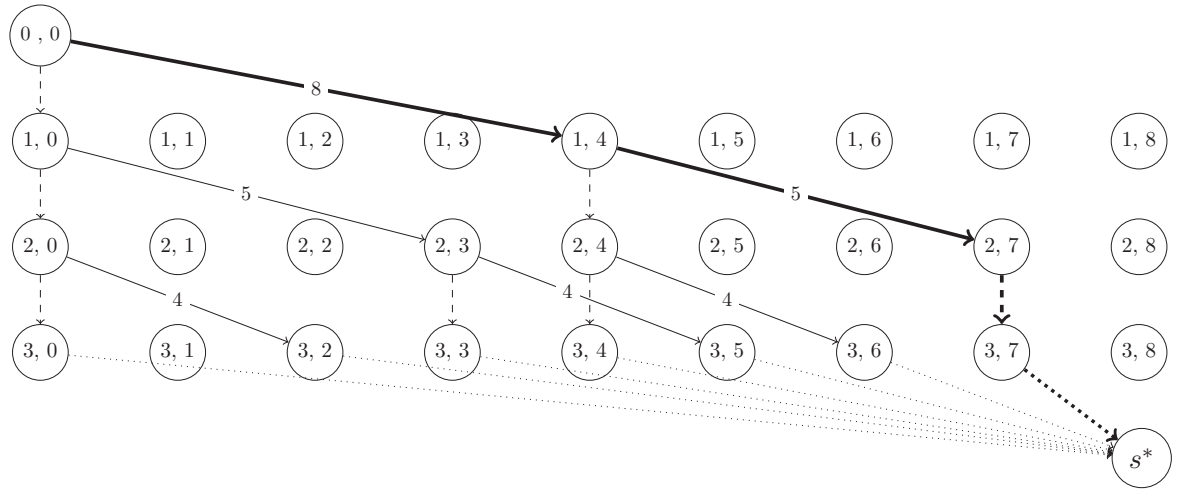

Fig. 1. Example of network $\mathcal{N}_{KP}$ for the knapsack problem.

definition, the only non-zero profit arcs are the ones from $\mathcal{A}_{i1}$, and their profits are shown in the figure. The bold arcs are from the longest path, i.e., the optimal solution, which contains items 1 and 2 and has total profit 13.

The network $\mathcal{N}_{DP}$ can be used in the generic recipe previously discussed, to derive an arc flow model for the KP with $O(|I|W)$ nodes and arcs. Although the resulting arc flow model has pseudo-polynomial size and has the integrality property, solving it with LP algorithms is usually not as fast as solving the problem by DP. However, the LP model can be a base for interesting results. For instance, (Boyd, 1992) extended this model to solve the knapsack separation problem, which is a generic tool to generate cutting planes for MILP formulations.

2.3. Example on the elementary shortest path problem with resource constraints

The *elementary shortest path problem with resource constraints* (ESPPRC) is linked with determining the best route of a single vehicle in vehicle routing problems (see, e.g., Irnich & Desaulniers 2005). In the ESPPRC there is a resource capacity and a directed graph. Each vertex is associated with a resource consumption and each arc is associated with a cost. The objective is to find a path with minimum cost such that: (i) each vertex is visited at most once (elementary constraint); and (ii) the total resource consumption from the vertices in the path does not exceed the resource capacity (resource constraint).

In the context of vehicle routing, the vertices are given by $I = \{0, 1, \ldots, n\}$, where 0 represents the depot and the other n vertices represent the clients. The solution must start and finish at the depot. Generally, the ESPPRC might consider a set of resources, but in this example we are concerned only with a single resource W, which is associated with the load capacity of a vehicle. Let $e_{(i,j)} \in \mathbb{R}$ be the cost of arc $(i, j) \in I \times I$ and $w_i \in \mathbb{R}_+$ be the resource consumption of $i \in I$.

Given a set of clients $S \subseteq I \setminus \{0\}$ that satisfies the resource constraint (i.e., $\sum_{j \in S} w_j \leq W$), we denote by $f_{ESPPRC}(S, i)$ the cost of a path of minimum cost from the depot 0 to a client $i \in S$ that visits every client in S exactly once. A client $j \in S \setminus \{i\}$ that precedes i in an optimal path related to $f_{ESPPRC}(S, i)$ is one that minimizes the sum of the cost $e_{(j,i)}$ from j to i and the cost $f_{ESPPRC}(S \setminus \{i\}, j)$ of a path of minimum cost that visits every client in $S \setminus \{i\}$ exactly once and finishes in $j \in S \setminus \{i\}$. This relation can be mapped into the following recursion:

$$f_{ESPPRC}(S, i) = \begin{cases} \min_{j \in S \setminus \{i\}} \{f_{ESPPRC}(S \setminus \{i\}, j) + e_{(j,i)}\}, & \text{if } S \neq \emptyset \text{ and } i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

The optimal ESPPRC solution value is given by $OPT(I, e, r) = \min\{f_{ESPPRC}(S, i) + e_{(i,0)} \mid S \subseteq I, i \in S, \sum_{j \in S} w_j \leq W\}$. The recursion can be transformed into a maximization problem, by inverting the sign of each $e_{(i,j)}$, resulting in a DP model. The state space $\mathcal{S}$ is a subset of $\{(S, i) \mid S \subseteq I, i \in S, \sum_{j \in S} w_j \leq W\} \cup \{s^*\}$, where the initial state is $s_0 = (\emptyset, 0)$ and the goal state s^* has precedent states related to the recursion of $OPT(I, e, r)$. For each state $(S, i) \in \mathcal{S} \setminus \{s^*\}$, the set of precedent states is $\Delta((S, i)) = \{(S \setminus \{i\}, j) \mid j \in S \setminus \{i\}\}$, and the decision cost to move from state $(S \setminus \{i\}, j) \in \Delta((S, i))$ to (S, i) is $c((S \setminus \{i\}, j), (S, i)) = e_{(j,i)}$. The set of precedent states of s^* is $\Delta(s^*) = \mathcal{S} \setminus \{s^*\}$, and the cost to move from state $(S, i) \in \Delta(s^*)$ to s^* is $c((S, i), s^*) = e_{(i,0)}$.

Let $\mathcal{N}_{ESPPRC}$ be the DP network for the ESPPRC obtained from the state space $\mathcal{S}$ presented above. The set of nodes $\mathcal{V}(\mathcal{N}_{ESPPRC}) \equiv \mathcal{S}$ corresponds to the state space, where v_{source} and v_{sink} are associated with $(\emptyset, 0)$ and $OPT(I, e, r)$, respectively. The set of arcs $\mathcal{A}(\mathcal{N}_{ESPPRC}) = \cup_{i \in I} \mathcal{A}_i$ contains arcs $\mathcal{A}_i = \{((S \setminus \{i\}, j), (S, i)) \in \mathcal{V} \times \mathcal{V}\}$ related to the decision of visiting client $i \in I \setminus \{0\}$, and arcs $\mathcal{A}_0 = \{((S, j), v_{sink}) \in \mathcal{V} \times \mathcal{V}\}$ related to the decision of returning to the depot.

Fig. 2 illustrates the network for an example with $W = 6$ and 4 clients, with $w_1 = 1$, $w_2 = 2$, $w_3 = 2$, and $w_4 = 5$. In this example, the costs are the following: $c_{(0,1)} = -5$, $c_{(0,2)} = -3$, $c_{(0,3)} = 1$, $c_{(0,4)} = -2$, $c_{(1,0)} = 0$, $c_{(1,2)} = 6$, $c_{(1,3)} = 2$, $c_{(1,4)} = -3$, $c_{(2,0)} = 0$, $c_{(2,1)} = -7$, $c_{(2,3)} = -1$, $c_{(2,4)} = 5$, $c_{(3,0)} = 0$, $c_{(3,1)} = 2$, $c_{(3,2)} = -5$, $c_{(3,4)} = 5$, $c_{(4,0)} = 0$, $c_{(4,1)} = -2$, $c_{(4,2)} = 5$, $c_{(4,3)} = 5$. On each arc, we present its associated cost, except for the arcs that enter s^*, which for the sake of conciseness we consider their cost to be 0 and present such arcs as dotted lines. The arcs of the optimal solution are highlighted in bold, and they correspond to visit the sequence of clients 3, 2, and 1, and returning to the depot. The optimal solution has value -11.

The network $\mathcal{N}_{ESPPRC}$ can be the base of an arc flow model for the ESPPRC, following the generic recipe previously presented. However, the number of subsets $S \subseteq I$ such that $\sum_{j \in S} w_j \leq W$ is exponential, implying that the state space presented for the ESPPRC is also exponential. Consequently, the corresponding arc flow

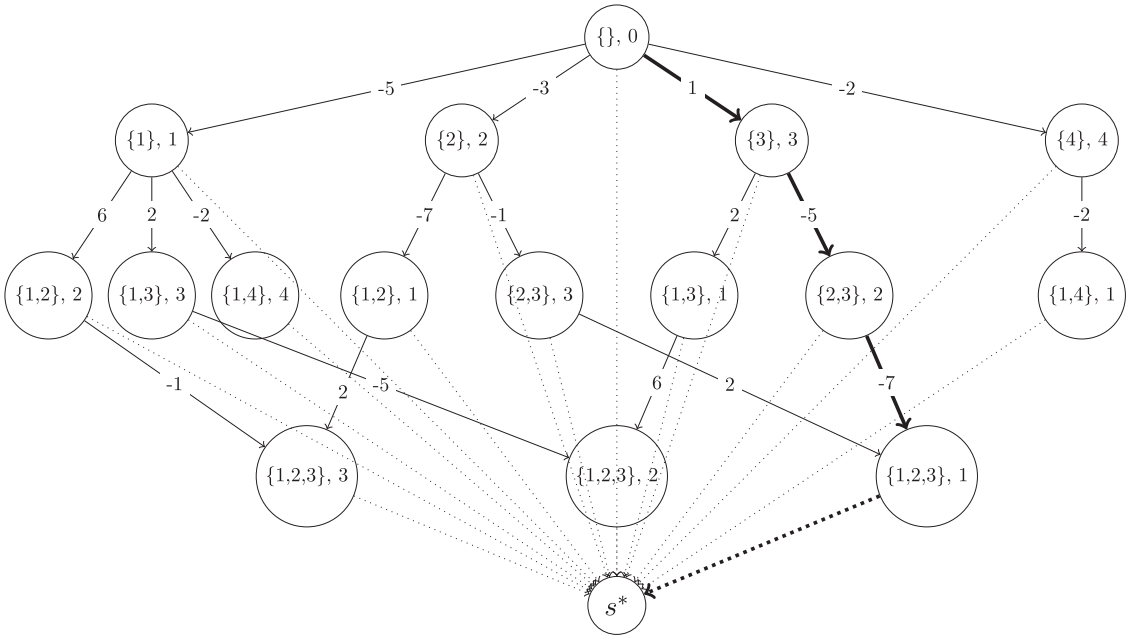

Fig. 2. Example of network $\mathcal{N}_{ESPPRC}$ for the elementary shortest path problem with resource constraints..

model has an exponential number of variables and constraints, and it is not practical to solve it directly. Practical techniques that can be used to overcome this inconvenience are described in Section 4. In particular, Section 4.2 presents a network of pseudo-polynomial size for a version of the ESPPRC where the elementary constraints are partially relaxed.

3. Dantzig-Wolfe decomposition and network flow formulations

Several challenging problems admit compact MILP models that, although having a reasonable (polynomial) size, are usually associated with weak linear relaxations. In such cases, reformulation methods can obtain models with stronger linear relaxation. One of such methods is the well-known *Dantzig-Wolfe* (DW) *decomposition* (see Dantzig & Wolfe 1961), which has been successfully used in many applications. The resulting models may be stronger, but they usually have an exponential number of variables, and complex methods are required to solve them in practice. In the following, we show how models obtained from DW decomposition can be the base to derive equivalent arc flow models with smaller (and possibly practical) size.

First, we show how the DW decomposition can be applied to Linear Programming (LP) models. Let us express the feasible solution region of the LP model as:

$$X_{LP} = \{x \in \mathbb{R}^n_+ \mid Ax \geq b, x \in X\}, \tag{9}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and X is a bounded convex polytope of a set of constraints. According to the Minkowski's Theorem, any point in X can be represented as a convex combination of the vertices (extreme points) of X (for the general case of an unbounded polytope, it is also necessary to consider a non-negative linear combination of the extreme rays of the polytope, see, e.g., Nemhauser & Wolsey 1988). Then, by defining $\mathcal{Q}(X)$ as the set of vertices of X, it follows:

$$X = \{x \in \mathbb{R}^n_+ \mid x = \sum_{q \in \mathcal{Q}(X)} q\lambda_q, \sum_{q \in \mathcal{Q}(X)} \lambda_q = 1, \lambda_q \geq 0, \forall q \in \mathcal{Q}(X)\}. \tag{10}$$

By substituting x as in (10) in (9), the feasible region in the space of the variables of the reformulated model becomes:

$$X_{LPDW} = \{\lambda \in \mathbb{R}_+^{|\mathcal{Q}(X)|} \mid \sum_{q \in \mathcal{Q}(X)} (Aq)\lambda_q \geq b, \sum_{q \in \mathcal{Q}(X)} \lambda_q = 1\}. \tag{11}$$

The reformulated problem, called *DW model*, has a variable associated with each vertex of X, and a new set of constraints, usually referred to as convexity constraints (see, e.g., Lübbecke & Desrosiers 2005). The DW model is just another way of expressing the same solution space. Therefore, the decomposition applied to LP models preserves the optimal solution value.

However, in the context of MILP, to obtain linear models stronger than the original linear relaxation, the integrality constraints must be implicitly considered in the reformulated variables. A possibility is to impose the integrality constraints just in the set X, reformulating over the vertices of $Conv\{x \in X \text{ and integer}\}$, the convex hull of the integer points in X (see, e.g., Nemhauser & Wolsey 1988), to optimize over the set:

$$X_{DW} = \{x \in \mathbb{R}^n_+ | Ax \geq b, x \in Conv\{x \in X \text{ and integer}\}\}. \tag{12}$$

Note indeed that $X_{DW} \subseteq X_{LP}$, and the relation can be strict when the set X does not have the integrality property. When that happens, the reformulated model is stronger. The strength of a model can have a strong impact on the search for the optimal integer solution. Typically the length of the search is smaller with stronger models, leading to smaller computational times. For further details, the reader is referred to Nemhauser and Wolsey (1988), which discusses at length the importance of deriving stronger models in the context of MILP.

A typical issue from the DW decomposition is that the resulting model may have an exponential number of variables. This issue is usually addressed by relying on the column generation method, a technique to solve LP models with a large number of variables (see, e.g., Lübbecke & Desrosiers 2005). The method solves a restricted version of the LP model with only a subset of variables by iteratively solving a *pricing problem*. The pricing generates non-basic variables (columns) that are candidates to improve the current restricted model. If no such variable exists, then the current basis of the restricted model is optimal for the original model, and the method halts. Column generation algorithms are complex, but

it pays off to solve models resulting from a DW decomposition, because they may be stronger when the set X does not have the integrality property and integrality is enforced in the pricing problem.

When the pricing problem of a DW model can be solved by DP, each column of the model can be associated with a path in the DP network, and the model can be seen as a path flow formulation. In such cases, the coefficients of a column are given as contributions from the arcs of the corresponding path. In addition, due to the no-memory property of the DP, for each column, the contribution $\ell^k_{(u,v)} \in \mathbb{R}$ of an arc (u,v) to a row coefficient $k \in \mathcal{C}$, where $\mathcal{C}$ is the set of constraints, is independent of other arcs. Then, given a DP network $\mathcal{N}$, and $d_k \in \mathbb{R}$, for every constraint $k \in \mathcal{C}$, we obtain a general path flow formulation:

$$\min \sum_{p \in \mathcal{P}(\mathcal{N})} \tilde{c}_p \lambda_p, \tag{13}$$

$$\text{s.t.:} \sum_{p \in \mathcal{P}(\mathcal{N})} \sum_{(u,v) \in \mathcal{A}(p)} \ell^k_{(u,v)} \lambda_p \geq d_k, \qquad \forall k \in \mathcal{C}, \tag{14}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in \mathcal{P}(\mathcal{N}). \tag{15}$$

The objective function (13) minimizes the total cost from the paths in the solution, and (14) is a set of general linear constraints. As discussed previously, due to the flow decomposition theorem, the path flow formulation (13)–(15) can be reformulated as the following arc flow formulation:

$$\min \sum_{(u,v) \in \mathcal{A}(\mathcal{N})} c(u,v) \varphi_{(u,v)}, \tag{16}$$

$$\text{s.t.:} \sum_{(u,v) \in \mathcal{A}(\mathcal{N})} \varphi_{(u,v)} - \sum_{(v,w) \in \mathcal{A}(\mathcal{N})} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \qquad \forall v \in \mathcal{V}(\mathcal{N}), \tag{17}$$

$$\sum_{(u,v) \in \mathcal{A}(\mathcal{N})} \ell^k_{(u,v)} \varphi_{(u,v)} \geq d_k, \qquad \forall k \in \mathcal{C}, \tag{18}$$

$$z \geq 0, \tag{19}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \qquad \forall (u,v) \in \mathcal{A}(\mathcal{N}). \tag{20}$$

The objective function (16) minimizes the total cost from the arcs in the solution, (17) are the flow conservation constraints, and (18) are general linear constraints, adapted from (14).

DW decomposition can generate path flow formulations that are usually associated with strong linear relaxations, and equivalent arc flow formulations can be derived from the DP network of the underlying pricing problem. As pointed before, to obtain stronger relaxations, one should reformulate over the convex hull of the integer points in a polytope that does not have the integrality property. Although a stronger linear relaxation is obtained, this kind of reformulation leads to pricing problems that are $\mathcal{NP}$-hard.

On the other hand, for arc flow models, the price to pay to obtain a stronger relaxation (i.e., to obtain a set equivalent to X_{DW}) is to use a set of constraints that defines a DP network with a pseudo-polynomial or an exponential number of arcs.

Usually, arc flow models of practical size can be derived when the resulting pricing problem has pseudo-polynomial complexity. However, when the pricing problem is strongly $\mathcal{NP}$-hard, the resulting arc flow model is generally too large to be solved in practice. When the pricing problem from path flow formulations is too difficult in practice, one may rely on relaxation methods. We show in Section 4 one of such relaxation methods, which can be used to derive smaller networks and obtain arc flow formulations that are still strong and have practical size. In the next sections, we present two examples of arc flow formulations derived from DW decomposition, with pseudo-polynomial and exponential size, respectively.

3.1. Example on the cutting stock problem

We present an example of an arc flow model derived from a DW decomposition for the *cutting stock problem* (CSP). In the CSP, a roll of width W and a set $I = \{1, 2, \ldots, n\}$ of items are given, such that each item $i \in I$ has a positive demand d_i and a width w_i. The objective is to cut the demand of all items from the minimum number of rolls.

Given an upper bound K on the minimum number of rolls, following Martello and Toth (1990), the CSP can be formulated as:

$$\min \sum_{j=1}^{K} y_j, \tag{21}$$

$$\text{s.t.:} \sum_{i=1}^{n} w_i x_{ij} \leq W y_j, \qquad j = 1, 2, \ldots, K, \tag{22}$$

$$\sum_{j=1}^{K} x_{ij} \geq d_i, \qquad \forall i \in I, \tag{23}$$

$$y_j \in \{0, 1\}, \qquad j = 1, 2, \ldots, K, \tag{24}$$

$$x_{ij} \in \mathbb{Z}_+, \qquad \forall i \in I, j = 1, 2, \ldots, K. \tag{25}$$

Each roll $j = 1, 2, \ldots, K$ is associated with a binary variable y_j that is equal to 1 if and only if roll j is used in the solution. The integer variable x_{ij} represent the number of copies of item $i \in I$ that is cut from roll j. The objective function (21) minimizes the number of rolls cut in the solution. Constraints (22) ensure that the size of each roll is satisfied and that no item is cut from an unused roll, whereas constraints (23) ensure that the demand of each item is satisfied.

According to Martello and Toth (1990), the optimal solution value of the linear relaxation of (21)–(25) is equivalent to the continuous lower bound that is obtained from the minimum length required to cut all of the demand from a stock with unlimited width, i.e., $\sum_{i \in I} w_i d_i / W$. This bound is known to be weak in practice, and its worst-case performance ratio asymptotically tends to $1/2$. To obtain a stronger bound, a DW decomposition can be applied to the model above (see, e.g., Vance 1998).

Constraints (22) correspond to K identical fractional knapsack polytopes. When integrality constraints (24) and (25) are taken into account, the K identical polytopes correspond to K identical sets P_{CSP} of integer knapsack solutions. A DW decomposition that includes constraints (22)–(25) in the sub-problem results in the following set-covering model:

$$\min \sum_{p \in P_{CSP}} \lambda_p, \tag{26}$$

$$\text{s.t.:} \sum_{p \in P_{CSP}} a_{ip} \lambda_p \geq d_i, \qquad \forall i \in I, \tag{27}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in P_{CSP}. \tag{28}$$

In the CSP, the integer knapsack solutions from P_{CSP} are called *cutting patterns* (or just patterns, for short, in the following), and

each of them represents the cutting of a single piece of stock. The objective function (26) minimizes the number of cutting patterns. Constraints (27) ensure that the demand of each item is satisfied, where a_{ip} is the number of copies of item i in pattern p.

In cutting and packing problems, a *proper pattern* is a pattern that respects the maximum number of copies of the items, whereas a *non-proper pattern* does not. The set-covering model by Gilmore and Gomory (1961, 1963) for the CSP is similar to (26)–(28), but it includes non-proper patterns that are obtained from the polytope of an *unbounded knapsack problem* (UKP), a variant of the KP where each item has an unlimited number of copies. The linear relaxation of (26)–(28), often referred to as *proper relaxation* (see, e.g., Kartak, Ripatti, Scheithauer, & Kurz 2015), is stronger than the linear relaxation of the model by Gilmore and Gomory (1961, 1963), but the optimal solution values of the corresponding MILP models are equal.

The linear relaxation of (26)–(28) is strong and often the number of rolls in the optimal solution is the rounded up optimal solution value from this relaxation. In fact, there is a conjecture related to the strength of this relaxation (see, e.g., Caprara, Dell'Amico, Díaz-Díaz, Iori, and Rizzi 2015 and Kartak et al. 2015):

Conjecture 1 (Modified Integer Round-Up Property (MIRUP))**.** *The difference between the optimal solution value of the CSP and the rounded-up solution value of the linear relaxation of (26)–(28) is at most one.*

The pricing problem of the set-covering model is a KP (where each item has d_i copies), which, as shown in Section 2.2, can be solved by DP, implying on the existence of a DP network $\mathcal{N}_{KP}$. As every column from the set-covering model can be represented as a path in $\mathcal{N}_{KP}$, this network can be the base of an arc flow model for the CSP:

$$\min z, \tag{29}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{KP})} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}(\mathcal{N}_{KP})} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \quad \forall v \in \mathcal{V}(\mathcal{N}_{KP}), \tag{30}$$

$$\sum_{(u,v)\in\mathcal{A}_i(\mathcal{N}_{KP})} \ell^i_{(u,v)} \varphi_{(u,v)} \geq d_i, \quad \forall i \in I, \tag{31}$$

$$z \in \mathbb{Z}_+, \tag{32}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \quad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{KP}). \tag{33}$$

The objective function (29) minimizes the total flow. Constraints (30) impose the flow conservation and constraints (31) are related to the demand of each item. Following the definitions in Section 2.2, we further define $\mathcal{A}_i(\mathcal{N}_{KP}) = \cup_{l\in\{0,1,\ldots,d_i\}}\mathcal{A}_{il}(\mathcal{N}_{KP})$, for each $i \in I$, and the contribution $\ell^i_{(u,v)}$ of arc $(u,v) \in \mathcal{A}_i(\mathcal{N}_{DP})$ is the number of copies of i associated with (u,v). The arc flow model (29)–(33), corresponds to the arc flow model proposed by Cambazard and O'Sullivan (2010), under the name *DP-flow*, for the *bin packing problem* (BPP), a particular case of the CSP where $d_i = 1$ for every $i \in I$. The DP-flow is equivalent to the set-covering model (26)–(28) (see, e.g., Delorme & Iori, 2020), and it follows the proper relaxation.

In this example, we started from a model that is associated with a weak linear relaxation. A DW decomposition resulted in a model with a stronger relaxation, but with the drawback of having a potentially exponential number of variables. Then, we showed how the DP network related to the pricing problem of the exponential model derives an equivalent pseudo-polynomial network flow model. Practical successful applications of this idea are shown in Section 7.

3.2. Example on the capacitated vehicle routing problem

In the *capacitated vehicle routing problem* (CVRP), we are given a set K of identical vehicles and a set $I = \{0, 1, \ldots, n\}$ of vertices where vertex 0 corresponds to a depot, and vertices $1, \ldots, n$ correspond to n clients. Each vehicle has a load capacity W, each client $i \in I \setminus \{0\}$ has a non-negative demand w_i, and each pair of vertices $i, j \in I$ is associated with a cost $c(i, j)$. The CVRP aims at determining a routing plan with the minimum total cost, such that: (i) exactly $|K|$ routes are considered; (ii) each route starts and ends at the depot; (iii) each client is visited exactly once; and (iv) the total demand from the clients of a route does not exceed the load capacity W.

In this example, we apply a DW decomposition on a compact arc flow model for the CVRP, obtaining a set partitioning model that derives an equivalent arc flow model of exponential size.

Let $I^* = \{I \setminus \{0\}\} \cup \{0^+, 0^-\}$ be the set of clients with two additional vertices 0^+ and 0^-, which are copies of the depot, each corresponding to the source and the sink of a route, respectively. Two additional sets of arcs, $\{(0^+, j) \mid j \in \{I \setminus \{0\}\}\}$ and $\{(i, 0^-) \mid i \in \{I \setminus \{0\}\}\}$, are considered. The following compact arc flow model, also known as three-index (vehicle-flow) formulation (see, e.g., Irnich, Toth, & Vigo, 2014), solves the CVRP:

$$\min \sum_{k\in K} \sum_{i\in I^*} \sum_{j\in I^*\setminus\{i\}} c(i,j)\phi^k_{(i,j)}, \tag{34}$$

$$\text{s.t.:} \sum_{j\in I^*\setminus\{i\}} \phi^k_{(i,j)} - \sum_{j\in I^*\setminus\{i\}} \phi^k_{(j,i)} = \begin{cases} -1, & \text{if } i = 0^+, \\ 1, & \text{if } i = 0^-, \\ 0, & \text{otherwise}, \end{cases} \quad \forall k \in K, i \in I^*, \tag{35}$$

$$\sum_{k\in K} \sum_{j\in I^*\setminus\{i\}} \phi^k_{(i,j)} = 1, \quad \forall i \in I \setminus \{0\}, \tag{36}$$

$$\omega_{ik} - \omega_{jk} + W\phi^k_{(i,j)} \leq W - w_j, \quad \forall i, j \in I^*, i \neq j, k \in K, \tag{37}$$

$$\omega_{ik} \leq W, \quad \forall i \in I^*, k \in K, \tag{38}$$

$$\phi^k_{(i,j)} \in \{0, 1\}, \quad \forall i, j \in I^*, k \in K, \tag{39}$$

$$\omega_{ik} \geq 0, \quad \forall i \in I^*, k \in K. \tag{40}$$

Model (34)–(40) considers a copy of the original graph for each vehicle $k \in K$. The binary variable $\phi^k_{(i,j)}$ is equal to 1 if and only if vehicle k moves from client i to client j, and the variable ω_{ik} indicates the accumulated demand already distributed by the vehicle k when arriving at client i. The objective function (34) minimizes the total cost of the routes. Constraints (35) guarantee the conservation of a unitary flow (representing a single route) for each of the $|K|$ vehicles. Constraints (36) guarantee that each client is visited exactly once. Constraints (37) and (38) model the propagation of the accumulated demand of the load capacity of each vehicle, and they are also used to ensure that the route of each vehicle is a single connected component (path).

Although model (34)–(40) has a polynomial number of variables, it has a large number of symmetries that makes it inefficient in practice (see, e.g., Irnich et al., 2014). Each possible route can be attributed to any vehicle, so each solution (routing plan) has $|K|!$ equivalent permutations. Next, we present a DW decomposition that eliminates this symmetry.

Flow conservation constraints (35) correspond to $|K|$ identical polytopes, each containing all paths from 0^+ to 0^-, whose vertices are always integer. By including constraints (37) and (38), each of the $|K|$ resulting polytopes contains all paths from 0^+ to 0^- satisfying the load capacity, but the integrality property is lost. Let us consider the DW decomposition over the set P_{CVRP} of vertices of the convex hull of constraints (35), (37), (38), and (40), that represent the set of integer paths that satisfy the load capacity. The resulting DW model is:

$$\min \sum_{p \in P_{CVRP}} \tilde{c}_p \lambda_p, \tag{41}$$

$$\text{s.t.:} \sum_{p \in P_{CVRP}} a_{pi} \lambda_p = 1, \qquad \forall i \in I \setminus \{0\}, \tag{42}$$

$$\sum_{p \in P_{CVRP}} \lambda_p = |K|, \tag{43}$$

$$\lambda_p \in \{0,1\}, \qquad \forall p \in P_{CVRP}. \tag{44}$$

The set partitioning model (41)–(44) provides a strong lower bound and it is among the most successful formulations to solve the CVRP in practice (see, e.g., Poggi & Uchoa, 2014 and Pecin, Pessoa, Poggi, & Uchoa, 2017). This model associates with each path $p \in P_{CVRP}$ a variable λ_p. The binary coefficient a_{ip} is equal to 1 if and only if client $i \in I \setminus \{0\}$ is visited in p. The objective function (41) minimizes the total cost, where the cost $\tilde{c}_p$ of a path p is the sum of the costs of the arcs in $\mathcal{A}(p)$. Constraints (42) guarantee that each client is visited by exactly one route. As each variable represents a unique path, the symmetry from the model (34)–(40) is eliminated.

The pricing problem associated with model (41)–(44) is equivalent to the ESPPRC. In Section 2.3, we presented the exponential DP network $\mathcal{N}_{ESPPRC}$ for the ESPPRC. Based on this network, and recalling that $\mathcal{A}_i(\mathcal{N}_{ESPPRC})$ is the set of arcs that visit $i \in I$, we can reformulate the set partitioning model into the following arc flow model:

$$\min \sum_{(u,v) \in \mathcal{A}(\mathcal{N}_{ESPPRC})} c(u,v) \varphi_{(u,v)}, \tag{45}$$

$$\text{s.t.:} \sum_{(u,v) \in \mathcal{A}(\mathcal{N}_{ESPPRC})} \varphi_{(u,v)} - \sum_{(v,w) \in \mathcal{A}(\mathcal{N}_{ESPPRC})} \varphi_{(v,w)} = \begin{cases} -|K|, & \text{if } v = v_{source}, \\ |K|, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \qquad \forall v \in \mathcal{V}(\mathcal{N}_{ESPPRC}), \tag{46}$$

$$\sum_{(u,v) \in \mathcal{A}_i(\mathcal{N}_{ESPPRC})} \varphi_{(u,v)} = 1, \qquad \forall i \in I \setminus \{0\}, \tag{47}$$

$$\varphi_{(u,v)} \in \{0,1\}, \qquad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{ESPPRC}). \tag{48}$$

The objective function (45) minimizes the total cost of the routing plan. Constraints (46) impose the flow conservation, and constraints (47) imply that each client is visited exactly once.

As network $\mathcal{N}_{ESPPRC}$ is possibly exponential, the arc flow model (45)–(48) may be too large to be used in practice. Nonetheless, this model is presented so as to provide an example in Section 4.2 on how a relaxation for the pricing problem of (41)–(44) derives a pseudo-polynomial arc flow model for the CVRP.

4. State-space relaxation on arc flow formulations

Dynamic programming is often an efficient tool to solve hard problems. However, when the number of states from a DP model is very large, it is not practical to solve it by enumerating all states, and more sophisticated methods are needed. From that observation, Christofides, Mingozzi, and Toth (1981) proposed a general relaxation procedure for DP, called *state-space relaxation*, which aggregates subsets of states in order to obtain a smaller state space which provides a bound for the original problem. Such relaxation can be embedded in exact solution methods to find the optimal solution for the original state space by (partially) disaggregating the relaxed state space during the search for feasible solutions. Since a state space may allow many different relaxations, it is desirable to determine relaxations that provide a good balance between number of states and strength of the relaxation.

Formally, a state-space relaxation consists of a mapping function $g : \mathcal{S} \to \mathcal{G}$ between two state spaces, where $|\mathcal{G}| < |\mathcal{S}|$. For every $s \in \mathcal{S}$ and $r \in \Delta(s)$, function g must guarantee that $g(r) \in \Delta(g(s))$, and the decision cost of going from $g(r)$ to $g(s)$ is defined as $\bar{c}_{(g(r),g(s))} = \max_{\{p,q \in \mathcal{S}\}} \{c(p,q) \mid g(p) = g(r), g(q) = g(s)\}$. The DP recursion of the resulting state-space relaxation is given by:

$$f_{SSR}(g(s)) = \begin{cases} \max_{\{p \in \Delta(g(s))\}} (f_{SSR}(p) + \bar{c}_{(p,g(s))}), & \text{if } s \neq s_0, \\ 0, & \text{if } s = s_0, \end{cases} \qquad \forall s \in \mathcal{S}. \tag{49}$$

As a relaxation, recursion (49) guarantees that $f_{SSR}(g(s)) \geq f_{DP}(s)$, i.e., it produces an upper bound for the original recursion f_{DP}. In the context of arc flow formulations induced by DP, the size of the LP formulation is strictly related to the size of the state space. In this case, smaller arc flow formulations can be obtained from state-space relaxations. Solutions obtained with the state-space relaxation may be unfeasible for the original problem. However, there are many cases in which arc flow formulations based on networks from state-space relaxations are guaranteed to produce optimal integer solutions that are feasible for the original problem. The main drawback of state-space relaxations is that they lead to arc flow formulations with weaker linear relaxation. This weakness occurs as the relaxation can profit from paths that are not feasible in the original network, generating, for instance, non-proper patterns in cutting and packing problems or non-elementary routes in vehicle routing problems. However, in many cases, the reduction on the size of the model pays off the loss in the linear relaxation strength.

Strong pseudo-polynomial arc flow formulations can be obtained from state-space relaxations of both pseudo-polynomial and exponential state spaces. Motivated by vehicle routing problems, Gouveia et al. (2019) studied modeling and solution methods of a class of pseudo-polynomial arc flow formulations obtained from state-space relaxation of exponential state-spaces, named by the authors as *layered graph formulations*.

Next, we present two examples of pseudo-polynomial arc flow formulations obtained from state-space relaxations over a pseudo-polynomial and an exponential state space.

4.1. Example on the state space for the cutting stock problem

In Section 2.2, we presented recursion (7) to solve the KP. This recursion produces network $\mathcal{N}_{KP}$, which has $O(|I|W)$ nodes and $O(|I|W)$ arcs. Then, in Section 3.1, network $\mathcal{N}_{KP}$ was the base of arc flow model (29)–(33) for the CSP, i.e., the DP-Flow model of Cambazard and O'Sullivan (2010).

Let $\mathcal{N}_{KP-SSR}$ be the network of a state-space relaxation of $\mathcal{N}_{KP}$ based on the mapping function $g((i,W')) = (W')$, for every state $(i,W') \in \mathcal{V}(\mathcal{N}_{KP})$. This mapping function disregards the dimension related to the partial set of items and merges states representing

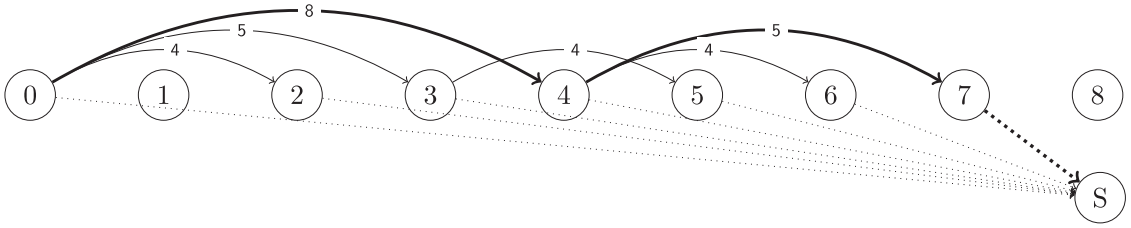

Fig. 3. Example of network $\mathcal{N}_{KP-SSR}$ obtained from a state-space relaxation of $\mathcal{N}_{KP}$.

the same partial capacities, into a single state. Network $\mathcal{N}_{KP-SSR}$ is smaller than $\mathcal{N}_{KP}$, with $O(W)$ nodes, but it may contain paths that violate the maximum number of copies of each item. Then, an arc flow model for the KP, based on $\mathcal{N}_{KP-SSR}$, needs additional side constraints to impose limits on the number of copies of items, while in $\mathcal{N}_{KP}$, such constraints are implicitly imposed by the configuration of the network.

In the CSP, as each demand constraint (31) imposes only a minimum and not a maximum number of items, there is no problem in considering a network that has paths representing cutting patterns with more copies of a single item than the required one. Hence, the arc flow model obtained by substituting $\mathcal{N}_{KP}$ by $\mathcal{N}_{KP-SSR}$ on model (29)–(33) still solves the CSP, because unnecessary items copies, if any, can be removed from the solution without affecting its cost. The resulting model is equivalent to the arc flow model for the CSP proposed by Valério de Carvalho (1999).

The model with $\mathcal{N}_{KP-SSR}$ is smaller than the original model, with $O(|I|+W)$ constraints instead of $O(|I|W)$, but its linear relaxation is weaker. Nonetheless, $\mathcal{N}_{KP-SSR}$ was obtained by Valério de Carvalho (1999) after reducing the DP network of the UKP, implying the graph of $\mathcal{N}_{KP-SSR}$ is a subgraph of the underlying graph from a DP for the UKP. This implies that the linear relaxation of the model with $\mathcal{N}_{KP-SSR}$ is at least as strong as the relaxation of the model by Gilmore and Gomory (1961, 1963), whose sub-problem is the UKP. Thus, the resulting linear relaxation still follows the MIRUP conjecture, implying that it is still strong. In practice, the arc flow model by Valério de Carvalho (1999), which is associated with $\mathcal{N}_{KP-SSR}$, is preferable than the model by Cambazard and O'Sullivan (2010), which is associated with $\mathcal{N}_{KP}$, as its linear relaxation is still strong and it is substantially smaller.

To exemplify, Fig. 3 shows the network $\mathcal{N}_{KP-SSR}$ obtained from the example of Fig. 1. We conclude that state-space relaxation can be considered to reduce the size of pseudo-polynomial arc flow formulations and obtain models with a linear relaxation that is still strong. This modeling technique to derive smaller, but yet efficient, pseudo-polynomial models from the relaxation of pseudo-polynomial state spaces has been mainly used, even without mentioning it, to model one- and two-dimensional cutting and packing problems and scheduling problems (see Section 7).

4.2. Example on the state space for the capacitated vehicle routing problem

In Section 3.2, we presented a DW decomposition for the CVRP that resulted in the path flow model (41)–(44), which is known to have a strong linear relaxation. Then, we presented the corresponding arc flow model (45)–(48), which is based on the network $\mathcal{N}_{ESPPRC}$ for the ESPPRC and has exponential size. In the following, we present a state-space relaxation for $\mathcal{N}_{ESPPRC}$ that leads to a pseudo-polynomial arc flow model for the CVRP.

A pseudo-polynomial state space $\mathcal{N}_{ESPPRC-SSR}$ can be obtained from a state-space relaxation of $\mathcal{N}_{ESPPRC}$ based on the mapping function $g((S,i)) = (\sum_{j\in S} r_j, i)$, for every state $(S,i) \in \mathcal{N}_{ESPPRC}$. This mapping function (originally proposed by Christofides et al. (1981)) merges states with the same total load from the visited clients into a single state. The resulting network $\mathcal{N}_{ESPPRC-SSR}$, which has $O(|I|W)$ nodes and $O(|I|^2W)$ arcs, preserves the resource constraints of the ESPPRC, but it may consider non-elementary paths (clients may be visited more than once).

In the exponential arc flow model (45)–(48) for the CVRP, the side constraints guarantee that clients are visited exactly once. Hence, by changing network $\mathcal{N}_{ESPPRC}$ by $\mathcal{N}_{ESPPRC-SSR}$ in model (45)–(48), the resulting model still solves the CVRP and has pseudo-polynomial size. This resulting model is often referred to as capacity-indexed formulation (see, e.g., Poggi & Uchoa, 2014), and it has been studied as a layered graph formulation by Gouveia et al. (2019).

Fig. 4 illustrates the network $\mathcal{N}_{SSR-ESPPRC}$ obtained from the state-space relaxation of the network from Fig. 2. It can be noticed that nodes $(\{1,2\},1)$ and $(\{1,3\},1)$ have been aggregated into a single node (3,1). As a result, $\mathcal{N}_{SSR-ESPPRC}$ has one node and one arc less than $\mathcal{N}_{ESPPRC}$. This is a minimal example of a reduction provided by $\mathcal{N}_{SSR-ESPPRC}$ that could be presented within the limits of this paper. However, due to the contrast of the exponential size of $\mathcal{N}_{ESPPRC}$ and the pseudo-polynomial size of $\mathcal{N}_{SSR-ESPPRC}$, there are many practical instances where the state-space relaxation provides a huge reduction on the size of the network.

This example shows how state-space relaxation derives pseudo-polynomial arc flow models from a network of exponential size. This kind of modeling technique is often used in vehicle routing problems, as the combinatorial structure of such problems usually leads to exponential state spaces in DW decompositions (see, e.g., Righini & Salani, 2008).

5. Dual insight

Research has shown that controlling the values of the dual-variables when using the primal simplex algorithm may improve computational times substantially. This happens, for instance, by modifying the model adding extra primal variables (corresponding to extra dual constraints) that must take null values at the end of the solution process. The use of this strategy may help in reducing difficulties related with instability of dual variables, primal degeneracy and long-tail effects, which are known to occur in column generation.

Other than the considerably smaller size, the use of more dual information is another advantage of the arc flow formulations over path flow formulations. Dual constraints (i.e., dual optimality conditions) of path flow formulations are non-negative combinations of dual constraints of arc-flow formulations. From a primal point of view, this competitive advantage can be interpreted as resulting from the possible recombination of basic variables to generate different paths in column generation algorithms (see, e.g., Sadykov & Vanderbeck, 2013).

5.1. On the dual space of network flow formulations

Many state-of-the-art algorithms to solve LP models are based on iteratively pivoting from one vertex of the LP polytope to a

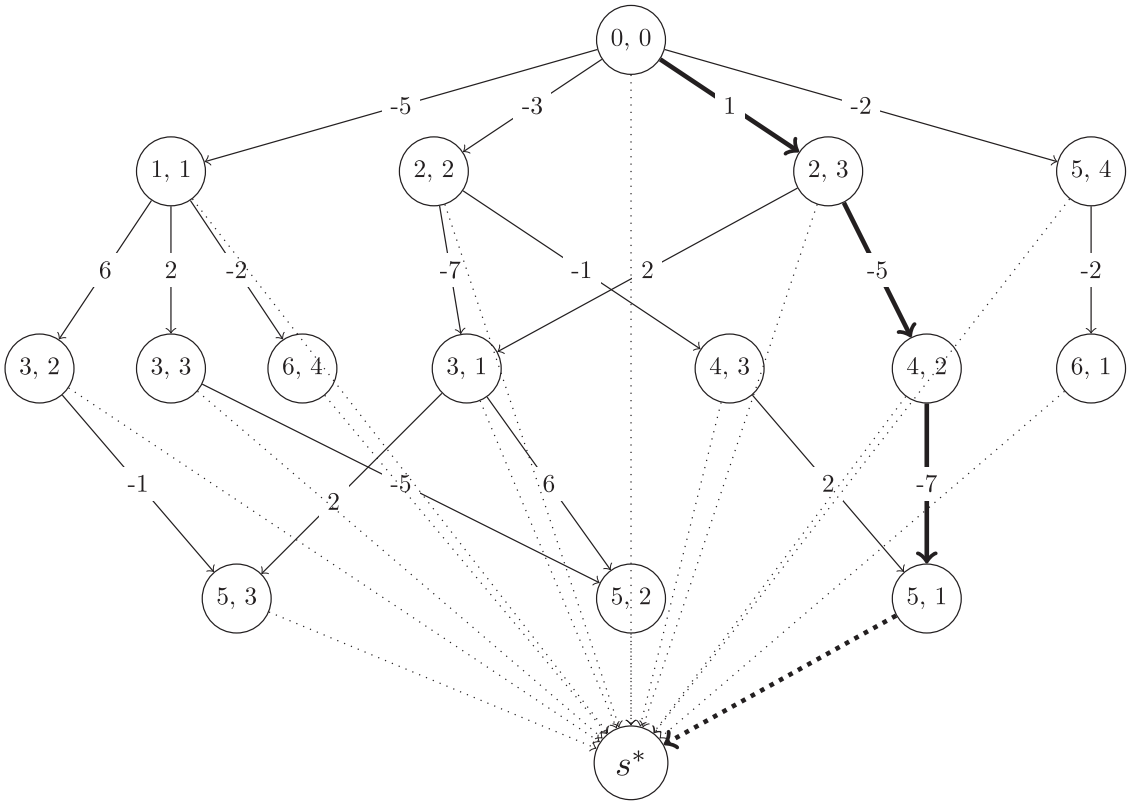

Fig. 4. Example of network $\mathcal{N}_{SSR-ESPPRC}$ obtained from a state-space relaxation of $\mathcal{N}_{ESPPRC}$.

better neighboring vertex until an optimal solution is found. An issue that may be critical for the computational time is degeneracy. A solution is degenerate when there are basic variables with a null value; in such cases, there may be degenerate pivots that lead to the same degenerate vertex, with no change in the primal solution, nor improvement in the objective function. Hard combinatorial optimization problems are often associated with highly degenerate models, and stalling, which is a sequence of degenerate pivots, occurs in practice (see, e.g., Bazaraa, Jarvis, & Sherali 2011). In fact, in a degenerate pivot, there is no change in the primal solution, but the new set of basic variables yields a change in the dual solution, leading to an alternative dual solution, often with a high oscillation of the values of the dual variables.

This instability often happens with master problems of DW reformulations, which may have a huge number of dual solutions associated to each primal solution. Several strategies showed that controlling the dual-variable values in the primal simplex algorithm may make the column generation procedure more efficient. For instance, du Merle, Villeneuve, Desrosiers, & Hansen, (1999) introduced a stabilization procedure, combining a perturbation method and a penalty method, that amounts to penalizing dual variables when they lie outside a predefined box. Wentges, (1997) searches dual solutions in the neighborhood of the best dual solution found so far, thus reducing instability. Other strategies include aggregating primal constraints (aggregation may change dynamically along the solution process), which enables transferring degeneracy to a complementary problem that is able to select a more central dual solution, as in Elhallaoui, Metrane, Desaulniers, and Soumis (2011); Elhallaoui, Villeneuve, Soumis, and Desaulniers (2005). Further improvements aim at identifying a set of non-basic variables that are pivoted together into the basis, avoiding degeneracy and strictly decreasing the objective function value, by solving a problem, coined as complementary problem, in Bouarab, El Hallaoui, Metrane, and Soumis (2017). For other strategies and insights on overcoming instability in combinatorial optimization algorithms, the reader is referred to, e.g., Lemaréchal (2001).

Another strategy to deal with primal degeneracy and instability is to develop strong models with a more restricted dual space. When comparing different LP models for the same problem with the same primal strength, the one with a tighter description of the dual space eliminates alternative dual solutions, potentially reducing degeneracy. This concept has been used in column generation algorithms by adding dual cuts that preserve all dual optimal solutions or even at least just one dual optimal solution, leading to significant speed-ups and reductions in the number of degenerate pivots (see, e.g., Valério de Carvalho, 2005, Lübbecke & Desrosiers, 2005, and Ben Amor, Desrosiers, & Valério de Carvalho, 2006).

LP solvers rely on the dual solution to prove optimality, and each dual constraint is an optimality condition. In the following, we show that arc flow formulations provide a tighter description of the dual space than their corresponding path flow formulations, with a richer description of the optimality conditions of the LP models. For instance, consider the dual of the linear relaxation of the path flow formulation (13)–(15), given by:

$$\max \sum_{k \in \mathcal{C}} d_k \beta_k, \tag{50}$$

$$\text{s.t.:} \sum_{k \in \mathcal{C}} \sum_{(u,v) \in \mathcal{A}(p)} \ell^k_{(u,v)} \beta_k \leq \tilde{c}_p, \qquad \forall p \in \mathcal{P}(\mathcal{N}), \tag{51}$$

$$\beta_k \geq 0, \qquad \forall k \in \mathcal{C}, \tag{52}$$

where dual variable β_k correspond to the constraints (14). Now, consider the dual of the linear relaxation of the corresponding arc flow formulation (16)–(20), given by:

$$\max \sum_{k \in \mathcal{C}} d_k \beta_k, \tag{53}$$

$$\text{s.t.: } \alpha_v - \alpha_u + \sum_{k\in\mathcal{C}} \ell^k_{(u,v)}\beta_k \le c(u,v), \qquad \forall (u,v) \in \mathcal{A}(\mathcal{N}), \tag{54}$$

$$\alpha_{source} - \alpha_{sink} \le 0, \tag{55}$$

$$\alpha_v \ge 0, \qquad \forall v \in \mathcal{V}(\mathcal{N}), \tag{56}$$

$$\beta_k \ge 0, \qquad \forall k \in \mathcal{C}. \tag{57}$$

Each dual variable α_v corresponds to the flow conservation of node $v \in \mathcal{V}(\mathcal{N})$, and each dual variable β_k corresponds to the side constraint $k \in \mathcal{C}$. For a given path $p \in \mathcal{P}(\mathcal{N})$, by performing a non-negative linear combination of the dual constraints (54) of every arc on $\mathcal{A}(p)$, we obtain:

$$\sum_{(u,v)\in\mathcal{A}(p)} (\alpha_v - \alpha_u) + \sum_{(u,v)\in\mathcal{A}(p)} \sum_{k\in\mathcal{C}} \ell^k_{(u,v)}\beta_k \le \sum_{(u,v)\in\mathcal{A}(p)} c(u,v) = \tilde{c}_p. \tag{58}$$

Note that, each node $v \in p$, except v_{source} and v_{sink}, is the head of an arc $(u,v) \in \mathcal{A}(p)$ and the tail of an arc $(v,w) \in \mathcal{A}(p)$, producing in the first summation, respectively, the terms α_v and $-\alpha_v$ that cancel each other. Then, (58) can be rewritten as:

$$(\alpha_{sink} - \alpha_{source}) + \sum_{k\in\mathcal{C}} \sum_{(u,v)\in\mathcal{A}(p)} \ell^k_{(u,v)}\beta_k \le \tilde{c}_p, \tag{59}$$

which is equivalent to the dual constraint from (51) for the path p, with an additional term $(\alpha_{sink} - \alpha_{source})$. From (55), we know that this term is always non-negative, implying a tighter constraint. Thus, we conclude that every dual constraint of the path flow formulation is a redundant dual constraint for the arc flow formulation, implying that the arc flow formulation provides a tighter dual space than the path flow formulation.

Generally, when the linear relaxation of either path flow or arc flow models is solved by simplex algorithms, the basis at each iteration is associated with a set of paths forming a primal-feasible solution. This set of paths is unique in the case of path flow, but not necessarily unique in the case of arc flow. If a same set of paths is considered to form the basis of an arc flow and of a path flow model, the basis of the former will be larger, due to the additional flow conservation constraints and the fact that each path is decomposed in a set of arcs in this model. But, in fact, the arc flow basis can be seen as obtained from a disaggregation of the path flow basis, which directly implies more optimality conditions. This guarantees a better description of the dual space, which, as already discussed, provides a number of practical benefits.

Concluding, in practice, each simplex iteration of an arc flow model can be more expensive when compared to a path flow model (due to the larger basis), but the number of pricing iterations needed to reach proven optimality can be substantially smaller, which in many cases is a significant advantage.

5.2. Example on the cutting stock problem

We present a numerical example to compare the dual of the classical arc flow model from Valério de Carvalho (1999) (see Section 4.1) to solve the CSP and the dual of its corresponding path flow model. The dual of the linear relaxation of this arc flow model is given by:

$$\max \sum_{i\in I} \beta_i, \tag{60}$$

$$\text{s.t.: } -\alpha_v + \alpha_{v+w_i} + \beta_i \le 0, \quad \forall i \in I, (v, v+w_i) \in \mathcal{A}_i(\mathcal{N}_{KP-SSR}), \tag{61}$$

$$\alpha_{source} - \alpha_{sink} \le 1, \tag{62}$$

$$\alpha_v \ge 0, \qquad \forall v \in \mathcal{V}(\mathcal{N}_{KP-SSR}), \tag{63}$$

$$\beta_i \ge 0, \qquad \forall i \in I. \tag{64}$$

Variables α_v are related to the flow conservation constraints of each $v \in \mathcal{V}(\mathcal{N}_{KP-SSR})$, and variables β_i are related to the demand constraint of each $i \in I$. Constraints (61) are related to the arc variables, and constraint (62) is related to the flow variable. The dual of the linear relaxation corresponding path flow model is given by:

$$\max \sum_{i\in I} \beta_i, \tag{65}$$

$$\text{s.t.: } \sum_{i\in I} a_{ip}\beta_i \le 1, \qquad \forall p \in \mathcal{P}(\mathcal{N}_{KP-SSR}), \tag{66}$$

$$\beta_i \ge 0, \qquad \forall i \in I. \tag{67}$$

The variables β_i are related to the demand constraints of each $i \in I$, and constraints (66) are related to each path (cutting pattern) p of $\mathcal{N}_{KP-SSR}$, where a_{ip} is an integer coefficient representing the number of times item i is cut from pattern p.

Consider again the example from Fig. 3, with bin capacity 8 and three items having $w_1 = 4$, $w_2 = 3$, $w_3 = 2$, and $d_1 = d_2 = d_3 = 1$. Tables 1 and 2 present, respectively, model (60)–(64) and (65)–(67) for this example. In the dual of the linear relaxation of the arc flow model, v_{source} and v_{sink} are represented by 0 and S, respectively. In this example, the path flow model is relatively smaller than the arc flow model, which is not common for practical instances, as the former may have exponentially more primal variables. However, our goal here is only to exemplify how the dual constraints (optimality conditions) of the path flow model are redundant dual constraints for the arc flow model. This can be observed as each dual constraint of the path flow model related to a pattern can be obtained by a non-negative linear sum of the dual constraints of the arc flow model related to the arcs that form this pattern. In particular: pattern $\{1,2\}$ can be formed by arcs (0,4), (4,7), $(7,S)$ and $(S,0)$; pattern $\{1,3\}$ by arcs (0,4), (4,6), $(6,S)$ and $(S,0)$; pattern $\{2,3\}$ by arcs (0,3), (3,5), $(5,S)$ and $(S,0)$; pattern $\{1\}$ by arcs (0,4), $(4,S)$ and $(S,0)$; pattern $\{2\}$ by arcs (0,3), $(3,S)$ and $(S,0)$; pattern $\{3\}$ by arcs (0,2), $(2,S)$ and $(S,0)$.

6. General solution methods

As previously discussed, an advantage of arc flow formulations over their equivalent path flow formulations is that they can be much smaller and often can be solved directly by a MILP solver (which is not practical for path flow models). However, pseudo-polynomial arc flow models can still be too large, depending on the size of the parameters of an instance. In such cases, one has to rely on more sophisticated methods to solve these models. An advantage of arc flow models derived from DW decompositions is that, since their networks are based on the underlying pricing problem, it is not always necessary to load the full network in the computer memory. Instead, one can use the structure of the pricing problem to derive methods based on column generation or iterative aggregation/disaggregation to solve the problem to integer optimality while avoiding to generate the full network. Such methods can lead to an increase in practical efficiency and avoid memory overflow when solving instances associated with huge networks.

Table 1
Example of a dual arc flow formulation for the CSP.

		α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_S	β_1	β_2	β_3	
arc	(0,4)	-1				1						1			≤ 0
	(0,3)	-1			1								1		≤ 0
	(4,7)					-1			1				1		≤ 0
	(0,2)	-1		1										1	≤ 0
	(3,5)				-1		1							1	≤ 0
	(4,6)					-1		1						1	≤ 0
	$(0,S)$	-1									1				≤ 0
	$(2,S)$			-1							1				≤ 0
	$(3,S)$				-1						1				≤ 0
	$(4,S)$					-1					1				≤ 0
	$(5,S)$						-1				1				≤ 0
	$(6,S)$							-1			1				≤ 0
	$(7,S)$								-1		1				≤ 0
flow	$(S,0)$	1									-1				≤ 1
max												1	1	1	

Table 2
Example of a dual path formulation for the CSP.

		β_1	β_2	β_3	
patterns	{1, 2}	1	1		≤ 1
	{2, 3}		1	1	≤ 1
	{1, 3}	1		1	≤ 1
	{1}	1			≤ 1
	{2}		1		≤ 1
	{3}			1	≤ 1
max		1	1	1	

Column generation

The column generation method (introduced in Section 3) solves the linear relaxation of models with a large number of variables, and is a popular tool to solve path flow models. The method was proposed by Ford and Fulkerson (1958a) to solve a path flow model for a maximal multi-commodity network flow problem and three years later generalized by Dantzig and Wolfe (1961) to solve the models resulting from DW-decompositions. Gilmore and Gomory (1961, 1963) solved a path flow model for the CSP by a column generation algorithm and were the first to show the practical efficiency of the method. Since then, column generation has been the base of several methods to solve path flow models. For references on column generation algorithms not strictly related to arc flow formulations, we refer the interested reader to the survey by Lübbecke and Desrosiers (2005) and the book by Desaulniers et al. (2006).

Column generation was applied to solve an arc flow model for the first time by Valério de Carvalho (1999), who proposed a column-and-row generation algorithm. In the context of arc flow models, column-and-row generation iteratively generates arcs to enter the simplex base, while the flow conservation constraints (rows) are restricted to nodes where there exists at least one incoming and one outgoing arc in the restricted problem. Sadykov and Vanderbeck (2013) studied the column-and-row generation method and experimentally compared the solution of path flow models by column generation, with the solution of equivalent arc flow models by column-and-row generation, and observed a faster convergence of the latter, which follows the discussion in Section 5.

In column generation algorithms (and lagrangian relaxations where only flow conservation constraints are left in the master) to solve arc flow models, an LPP on the underlying network is iteratively solved, and its computational efficiency is strictly related to the network size. In this context, when the network is too large, one may rely on dynamic graph generation methods to solve the LPP, as proposed by, e.g., Fischer and Helmberg (2014), to solve LPPs that arise as sub-problems from arc flow models based on large-scale time-expanded networks.

Iterative aggregation/disaggregation method

Several authors have studied state-space relaxation techniques to reduce the size of arc flow formulations (see, e.g., Macedo, Alves, Valério de Carvalho, Clautiaux, & Hanafi, 2011, Clautiaux et al., 2017, Voge & Clautiaux, 2012, Boland, Hewitt, Marshall, & Savelsbergh, 2017a, Boland & Savelsbergh, 2019, Riedler, Ruthmair, & Raidl, 2019). These techniques are equivalent to applying a surrogate relaxation to the flow conservation constraints related to subsets of nodes. Theoretically, this only reduces the number of constraints. Practically speaking, after an aggregation, many arcs (variables) associated with the same decision become equivalent in the reduced network and can be merged. From an initial relaxation, these methods use iterative techniques in which the relaxation is refined, typically by splitting nodes that have been aggregated, until the solution produced by the relaxation is feasible for the original problem, or its value is equal to a known primal bound.

Macedo et al. (2011) were the first to use these techniques in pseudo-polynomial arc flow models. The authors used two aggregations: one produces a relaxation, the other a heuristic solution. These results were later generalized by Voge and Clautiaux (2012) and Clautiaux et al. (2017), who studied the difficulty of the different sub-problems and the performance ratio obtained by an aggregated model. More efficient refining strategies are studied in Riedler et al. (2019). The authors show that their path-based techniques are more effective and underline the importance of heuristic methods in the algorithm. In early works, the elements to be aggregated were decided beforehand. In Boland and Savelsbergh (2019), the authors introduced the paradigm of *dynamic discretization discovery*, in which the discretization is constructed on the fly, producing a better relaxation, by using information from the network construction process.

Such aggregation techniques have been generalized to flows in hypergraphs by Benkirane, Clautiaux, Damay, and Detienne (2019) to solve a joint rolling-stock and train selection problem for the French railway company. The authors show that even for hypergraphs, one can safely use reduced-cost filtering on aggregated variables. Another type of relaxation is used in Nadarajah and Cire (2017) to deal with problems where several constraints can be reformulated as network-flow constraints. In their model, a network is created for each constraint, and the flow conservation constraints of all networks but one are relaxed in a Lagrangian way.

Graph reduction methods

An important element of efficient solution methods for arc flow models is to determine arcs that can be removed from the network without losing optimality. Removing redundant arcs is important, as a smaller network may lead to a reduction in symmetry, a tighter relaxation, and a smaller branch-and-bound tree. Many techniques to reduce the number of arcs are problem-dependent, usually based on dominance criteria of the underlying DP, and we discuss some of them in Section 7, under specific applications. Nonetheless, general reduction techniques have been used to enhance arc flow models, as, for instance, the reduced-cost variable-fixing method (see, e.g., Pessoa, Uchoa, de Aragão, & Rodrigues, 2010 and Kramer, Lalla-Ruiz, Iori, & Voß, 2019b).

Given a MILP minimization model, the reduced-cost variable-fixing method performs a domain propagation based on a dual-feasible solution of the corresponding linear relaxation and an upper bound value z_{ub} corresponding to an available feasible solution. The idea is that, given the objective value z_{lb} of the dual-feasible solution, any integer variable with a reduced cost greater than or equal to $z_{ub} - z_{lb}$ can be removed from the model. Irnich, Desaulniers, Desrosiers, and Hadjar (2010) proposed an efficient reduced-cost variable fixing algorithm for eliminating arcs in network flow models that computes a bound on the reduced cost of the arcs in arc flow models based on a dual solution of the linear relaxation of its equivalent path flow model. This method was later extended by Desaulniers, Gschwind, and Irnich (2020) to determine and efficiently handle pairs of sequential arcs that cannot be in the same path in an optimal solution. The impact of different dual solutions in reduced-cost variable-fixing for network flow models has been recently discussed by de Lima, Iori, and Miyazawa (2021).

7. Successful applications of pseudo-polynomial arc flow models

In this section, we discuss the main applications of arc flow models, and discuss problem-dependent solution methods and reduction criteria.

7.1. Cutting and packing problems

Arc flow models have been used in a variety of cutting and packing problems, both in one and multiple dimensions.

One-dimensional problems

The classical arc flow model for the CSP in Valério de Carvalho (1999) has a node for each partial stock size, the arcs relate the items with cut positions (item arcs) or represent loss stock (loss arcs), and the combination of arcs into paths represents cutting patterns. To solve this model to integer optimality, Valério de Carvalho (1999) proposed a branch-and-price algorithm based on column-and-row generation. To accelerate the column generation's convergence, the pricing problem generates paths (instead of single arcs). Valério de Carvalho (1999) proved that the arc flow model is equivalent to the path (set-covering) model by Gilmore and Gomory (1961, 1963), whereas Martinovic, Scheithauer, and Valério de Carvalho (2018) and Delorme and Iori (2020) proved that the arc flow model is equivalent to the one-cut model by Rao (1976) and Dyckhoff (1981). The one-cut is a pseudo-polynomial model where variables represent cutting operations on the roll.

To reduce the number of arcs, Valério de Carvalho (1999) constructs the graph considering that items of a single roll can always be ordered by non-increasing width. The resulting graph independently follows the state-space relaxation discussed in Section 4.1. To obtain even smaller networks, Côté & Iori, (2018) proposed the meet-in-the-middle technique: each path representing a cutting pattern can be transformed into an equivalent one by left aligning the items whose left border is at the left of a given threshold parameter t, and right aligning the remaining items. Recently, de Lima et al. (2021) proposed a new way to reduce the graph in Valério de Carvalho (1999) by considering a maximum waste for each roll. They developed a branch-and-price framework in which the branching produces a series of small arc flow models which are solved one at a time by a general purpose MILP solver.

Cambazard and O'Sullivan (2010) proposed the DP-flow, an arc flow model for the CSP (already introduced in Section 3.1) based on the DP network of the KP. Differently from the classical arc flow model for the CSP, which considers a node for each partial stock size, the DP-flow considers a node for each pair of items and partial stock size. The network has a level for each item, and each feasible path visits the level of each item exactly once. Although this modeling technique can substantially increase the number of nodes, it allows one to consider only proper patterns. On the other hand, the classical arc flow model is smaller, but it cannot distinguish between proper and non-proper patterns, so its relaxation can be weaker than the one of the DP-flow.

A generalization of the classical arc flow model for the CSP was proposed by Brandão and Pedroso (2015). This generalization can model related problems, such as the vector BPP, the BPP with conflicts, the cardinality constrained BPP and CSP, and the graph coloring problem. The resulting network may be significantly large, but several techniques (referred to as graph compression) are proposed to reduce the network size. To break symmetry, the authors proposed a modeling technique similar to the graph construction of the DP-flow, considering an extra dimension, related to the items, on the nodes.

Delorme and Iori (2020) proposed the *reflect formulation*, a pseudo-polynomial model for the CSP that considers nodes and arcs only from half of the stock size. In practice, this formulation is significantly smaller than the classical arc flow model. Besides, the reflect formulation has been proven to be as strong as the classical arc flow model without reduction criteria. Other than the CSP, Delorme and Iori (2020) extended the reflect formulation to solve the variable-sized BPP and the BPP with item fragmentation. Dell'Amico, Delorme, Iori, and Martello (2019) adapted the reflect formulation to solve a feasibility problem in a Benders' decomposition algorithm for the multiple knapsack problem.

Alves and Valério de Carvalho (2008) presented a branch-and-price-and-cut algorithm for the multiple length CSP that solves the Gilmore and Gomory (1963) machine balance model, which is a path flow formulation, using the original arc flow model to generate attractive columns in a single subproblem and the variables of the arc flow model to implement a branching scheme, by expressing the branching constraints in terms of the Gilmore and Gomory model variables. The equivalence of the path and arc flow models ensures a correct transferral of dual information. Valid dual inequalities are used to stabilize and accelerate the search in the entire branch-and-bound tree.

Recently, arc flow models were proposed for the *skiving stock problem* (SSP), which is strongly related to the dual BPP. In the SSP, we are given a set of items, each having a length and a maximum number of copies, to be combined into the maximum number of larger items of minimum length W. Martinovic and Scheithauer (2016b) proposed an arc flow model for the SPP where nodes represent the length sum of combinations of items, arcs represent the positioning of the items in a combination, and paths represent a combination of items. This model, which is similar to the classical arc flow model for the CSP, cannot distinguish between proper and non-proper patterns. Martinovic and Scheithauer (2016a) proposed an arc flow model that considers only proper patterns, which, similarly to the DP-flow model, considers a node for each pair of items and possible length sum, and the network

has a level for each item. Martinovic, Delorme, Iori, Scheithauer, and Strasdat (2020) proposed two arc flow models for the SSP: one considers reversed loss arcs, which lead to a reduction of the worst-case number of nodes from $2W$ to W, and the other is based on the reflect model for the CSP.

Multi-dimensional problems

Macedo, Alves, and Valério de Carvalho (2010) extended the classical arc flow model to minimize the number of bins in the two-stage two-dimensional guillotine CSP. Their model has a one-dimensional arc flow graph for the first stage cuts (which cut the bins to obtain strips), and a one-dimensional arc flow graph for each possible strip size to determine the second stage cuts (which produce the items from the strips, possibly admitting a final trim loss cut). To reduce symmetry, the flow in the second stage graph of a given strip size is equal to the sum of the flows for that strip size in the first stage graph. Note that these strips may belong to the same bin or different bins. Procedures to reduce the size of the graph and a new family of cutting planes based on the height of the items were proposed. The arc flow model in Macedo et al. (2010) was later adapted by Mrad (2015) to solve the two-stage two-dimensional guillotine strip packing problem.

Nesello, Delorme, Iori, and Subramanian (2018) proposed an arc flow model, similar to the one by Macedo et al. (2010), to solve a three-stage two-dimensional strip packing problem where a limit is imposed on the number of shelves and setup times between items must be taken into account. Delorme, Iori, and Martello (2017) adapted the classical arc flow model for the CSP to solve the one-dimensional contiguous bin packing problem, which often appears as a sub-problem in two-dimensional cutting and packing solution methods. The authors used it to solve the sub-problem of a Benders' decomposition algorithm for the two-dimensional strip packing with item rotations and for the pallet loading problem.

Clautiaux et al. (2018) solved the four-stage two-dimensional guillotine bounded knapsack problem with a network flow model based on a directed acyclic hypergraph. They compared the efficiency of several algorithms based on this representation, including a MILP model and an iterative state-space relaxation based on the corresponding DP.

7.2. Scheduling problems

In the scheduling field, both time-indexed (see, e.g., Sousa & Wolsey, 1992) and path flow (see, e.g., van den Akker, Hurkens, & Savelsbergh, 2000) formulations have been widely used to solve a variety of optimization problems. In a closely related context, polynomial arc flow models have been proposed more than three decades ago by Eppen and Martin (1987) to solve a lot-sizing problem. However, pseudo-polynomial arc flow formulations have only been adopted for scheduling problems during the last decade. It is worth mentioning that time-indexed formulations can be used to derive arc flow models of the same strength, which are associated to a sparser constraint matrix. The equivalence between time-indexed and arc flow formulations has been shown in different contexts starting from Valério de Carvalho (2002), who proved such equivalence by a unimodular transformation, in the context of the CSP.

Pessoa et al. (2010) developed a branch-and-cut-and-price algorithm for the problem of minimizing weighted tardiness on identical parallel machines (denoted as $P||\sum w_j T_j$ in the three-field classification of Graham et al. (1979)). Their algorithm is based on an arc-time-indexed formulation and is improved with a number of combinatorial techniques, including variable fixing by reduced costs, extended capacity cuts, dual stabilization, and the direct solution of the formulation by a MILP solver if the fixing procedure had consistently reduced the number of variables. The method in Pessoa et al. (2010) was later extended by Bulhões, Sadykov, Subramanian, and Uchoa (2020), who proposed a branch-and-cut-and-price algorithm to solve a path flow formulation for parallel machine scheduling in which the branching is based on the variables of the arc flow model.

Lancia, Rinaldi, and Serafini (2011) developed a branch-and-price algorithm for the job shop problem with a general min-sum objective function. Their algorithm is based on the solution of an arc flow model in which a path has to be chosen for each job (which is composed of multiple operations). The model was strengthened by clique inequalities.

Ratli, Benmansour, Macedo, Hanafi, and Wilbaut (2013) presented a comprehensive list of mathematical models for scheduling jobs on a single machine by minimizing weighted earliness and tardiness. The problem, denoted as $1||\sum \alpha_j E_j + \sum \beta_j T_j$, is relevant in the context of just-in-time production. In computational tests on random instances, the arc flow model achieved the lowest optimality gaps.

Mrad and Souayah (2018) presented a direct extension of the arc flow formulation by Valério de Carvalho (1999) to the problem of scheduling jobs on identical parallel machines with the objective of minimizing the makespan ($P||C_{\max}$).

Kramer, Dell'Amico, and Iori (2019a) considered again the problem of scheduling jobs on identical parallel machines, but focused on the minimization of the weighted sum of the completion times ($P||\sum w_j C_j$). They presented an arc flow model, and then enhanced it by grouping jobs having the same weight and processing time and creating time windows for each group by considering job priorities. A computational comparison showed that the enhanced arc flow performed very well compared to other time-indexed, convex integer quadratic programming, and path flow models.

Kramer et al. (2021) extended the work in Kramer et al. (2019a) to deal with the case of family setup times ($P|s_i|\sum w_j C_j$). The authors proposed three different arc flow models. In the most efficient one, the network is divided into a set of layers, one layer per family. Arcs within the same layer considered only the processing time of a job, whereas arcs connecting two layers considered both setup and processing times. Computational results showed that setup times worsen the linear relaxation value of the arc flow models, which outperformed a path flow model only on a handful of instances.

A further generalization of Kramer et al. (2019a) was provided in Kramer, Dell'Amico, Feillet, and Iori (2020a), who considered the case of release dates ($P|r_j|\sum w_j C_j$) and obtained stricter time windows for the jobs. The resulting arc flow model obtained better results than a branch-and-price based on the path flow formulation on instances having jobs of small and moderate processing time.

We also mention that interesting integrations between scheduling and other combinatorial problems have been tackled in the literature. Cappanera and Scutellà (2015) considered a joint assignment, scheduling, and routing problem arising in home care optimization. An arc flow model was used for the generation of patterns, which represent feasible combinations of the three decision levels of the problem. Cire, Diamant, Yunes, and Carrasco (2019) proposed an arc flow formulation for a rotation assignment and scheduling problem arising in the context of clinical rotations. They demonstrate that the network model was computationally superior to a classical MILP model on a real-world set of instances.

Braga, Alves, Macedo, and Valério de Carvalho (2016) compared two formulations for a combined cutting stock and scheduling problem: a compact formulation strengthened with knapsack inequalities and an arc flow formulation. Using a revised version of the latter formulation based on aggregated time periods, they derived a heuristic solution procedure that proved to be effective for the solution of medium size instances. Rietz, Alves, Braga, and Valério de Carvalho (2016) proposed and analyzed an arc

flow formulation for a combined cutting stock and scheduling problem on parallel machines. Different strategies to simplify the formulation by reducing the number of arcs were presented.

Trindade, de Araújo, and Fampa (2020) solved scheduling problems with batch processing machines. In such problems, the jobs are grouped into batches to be scheduled in machines, where the batches have a capacity to be respected by the size of its grouped items. The author proposed arc flow models where nodes represent a discretization of the capacity of a batch and arcs represent either a scheduled job in a batch or an unused capacity.

7.3. Routing problems

In routing problems, the input is usually based on a network, where clients and depots are given as nodes and arcs are related to transportation between the nodes (see, e.g., Toth & Vigo, 2014). For such problems, arc flow models based on the input network lead to compact models that are small but may have weak relaxations and too much symmetry to be solved in practice. Examples of such compact models are the classical MTZ model by Miller, Tucker, and Zemlin (1960) for the traveling salesman problem (TSP) and the three-index formulation (see Section 3.2) for the CVRP. In many routing problems, the ability to avoid non-elementary routes is an important aspect to determine the strength of a relaxation. For this reason, the best solution methods for many variants, especially the ones with multiple vehicles, are usually based on path flow formulations (see, e.g., Poggi & Uchoa, 2014 and Pessoa, Sadykov, Uchoa, & Vanderbeck, 2019), as they can handle non-elementary routes more easily than arc flow formulations. However, pseudo-polynomial arc flow formulations have still been used to solve open problems of a number of variants, and to enhance state-of-the-art methods based on path flow formulations.

Pseudo-polynomial arc flow formulations for vehicle routing problems were first introduced by Godinho, Gouveia, Magnanti, Pesneau, and Pires (2007) and Pessoa, de Aragão, and Uchoa (2008), which, inspired by the early work by Picard and Queyranne (1978) for single machine scheduling with minimum tardiness, proposed the capacity-indexed formulation for the CVRP. The capacity-indexed formulation, which follows the network from the state-space relaxation in Section 4.2, has a node (i, q) for each pair of client (or depot) i and partial capacity q, where the partial capacities group the nodes into levels (layers). Then, each path arriving at a node (i, q) represents a route that finishes in client i and has total load q.

The capacity-indexed formulation is based on a network where paths may be associated with non-elementary routes (which weakens its linear relaxation), and may be too large to be solved in practice (see, e.g., Pessoa et al., 2008). The main interest in this formulation is that its variables can be used to define cutting planes (e.g., the extended capacity cuts) to strengthen path flow formulations for vehicle routing problems (see, e.g., Poggi & Uchoa, 2014). According to Uchoa (2012), cutting planes based on the capacity-indexed formulation have been successfully used in other vehicle routing problems, and even in parallel machine scheduling problems.

Macedo et al. (2011) proposed an iterative aggregation/disaggregation algorithm to solve an arc flow formulation for the vehicle routing problem with time windows and multiple routes. In the underlying network, each node corresponds to a time instant, and each arc corresponds to a possible subtour. Aggregation based on rounding procedures was used to make the routes with non-integer travel times fit the model's graph. Whenever an infeasible solution was found, the nodes involved in the infeasibility were disaggregated, and the resulting model was solved again.

Braga, Alves, and Macedo (2017) proposed an arc flow formulation for the multi-trip inventory routing problem where vehicles can perform more than a single route per time period. Following Macedo et al. (2011), nodes and arcs of the underlying network correspond to, respectively, time instants and routes that may be assigned to a vehicle.

Algorithms based on iterative aggregation/disaggregation were later proposed to solve time-expanded arc flow formulations for the TSP with time windows (TSPTW). The network of time-expanded formulations has a node (i, t) for each client i and time instant t. Each path from the source to a node (i, t) represents a path arriving at client i at time t. Boland, Hewitt, Vu, and Savelsbergh (2017b) and Riedler et al. (2019) proposed iterative aggregation/disaggregation algorithms to solve a time-expanded model for the TSPTW, starting with a reduced network that is sufficient to produce a bound for the problem and is iteratively refined to find an optimal solution. The method in Boland et al. (2017b) was later extended by Vu, Hewitt, Boland, and Savelsbergh (2020) to solve the generalization where travel times are time-dependent. This kind of approach was also used by Boland et al. (2017a) to solve a continuous-time service network design problem without the need to approximate the solution by a discretization of the time-horizon.

7.4. Miscellaneous

Earth Observation Satellite scheduling requires to determine the pictures to be taken by a set of satellites in a given time period, so as to satisfy side constraints and optimize an objective function. Pseudo-polynomial arc flow models have been proposed to solve such problems by Gabrel and Murat (2003) and Wang, Wu, Xing, and Pedrycz (2020). In these models, the nodes are related to a discretization of the time horizon, and the arcs are related to the decisions on the pictures to be taken.

Kramer et al. (2019b) solved the dynamic berth allocation problem, which aims at allocating vessels into quays that are divided into berths, while optimizing an objective function based on the service time of each vessel. The authors proposed an arc flow model where nodes are associated with time instants and arcs are related to vessels serving. Problem-dependent reduction criteria and a reduced-cost variable-fixing algorithm were proposed to improve the solution time.

In the capacitated p-center problem, a set of customers must be attributed to capacitated facility locations, minimizing the maximum distance between each client and its facility. To solve this problem, Kramer, Iori, and Vidal (2020b) proposed an arc flow model where the underlying graph has a component for each location, in which nodes correspond to partial filling of the facility capacity, and arcs correspond to customers and unused capacity.

Ramos, Alves, and Valério de Carvalho (2020) described an arc flow formulation for the multi-trip production, inventory, distribution, and routing problem with time windows. Nodes and arcs represent time instants and vehicle routes, respectively.

Train timetabling problems require to determine a periodic timetable for a set of trains that satisfies operational constraints and optimizes an objective function (see, e.g., Cacchiani & Toth, 2012). The input for such problems is based on a graph where nodes represent stations and arcs represent tracks. Train timetabling problems have been successfully solved by arc flow models based on time-expanded networks by Caprara, Fischetti, and Toth (2002) and Fischer and Helmberg (2014).

Recently, van Hoeve (2020) proposed an arc flow model to solve the graph coloring problem, which asks to partition a graph into the minimum number of independent sets. The proposed arc flow model is based on decision diagrams, which are closely related to DP (see, e.g., Hooker, 2013) and result in acyclic graphs.

The resulting network in van Hoeve (2020) is equivalent to a DP network to solve the maximum independent set problem, which is usually the pricing problem of path (set-partitioning) formulations for the graph coloring problem (see, e.g., Gualandi & Malucelli, 2012). An iterative refinement method is proposed by the authors to deal with the exponential size of the model.

8. Conclusion and future research directions

In this survey, we reviewed over one hundred papers related to arc flow formulations. Many of these papers present arc flow models having pseudo-polynomial size and a strong linear relaxation. The number of applications of pseudo-polynomial arc flow formulations has grown considerably since the work by Valério de Carvalho (1999), making them a valid alternative to path flow formulations. For many combinatorial optimization problems, path flow formulations are still the best alternative, because they can embed difficult constraints in the subproblem solved to build the paths. However, arc flow formulations have many positive aspects, among which we highlight that: they are powerful modeling tools that allow one to model complex issues from real systems; pseudo-polynomial arc flow models are often related to DW decompositions, providing strong linear relaxations; differently from path flow formulations, they provide models that usually have a number of variables which allows practical solutions directly by general MILP solvers, avoiding complex implementations; compared to equivalent path flow formulations, they have a richer description of the dual space, leading to a faster convergence of simplex-based methods; pseudo-polynomial arc flow models can be derived from state-space relaxations from the underlying DP network from path flow models.

This survey is a contribution to a systematic study of arc flow formulations, but we would like to point out that there are many open questions and research lines to be pursued. Some of them are the following.

One of the advantages of arc flow formulations is that they provide a tighter description of the dual feasible space keeping the primal strength. Besides the methodologies presented in this paper, are there any general hints on how to do both primal and dual strengthening in arc flow models? For example, in extended formulations, using new sets of variables may strengthen the primal model and, as new (primal) variables are dual cuts, there is also a richer description of the dual feasible space.

Models presented in this survey explore solution spaces that are convex combinations of flows, each corresponding to an $s-t$ path, which is a sequence of arcs and vertices. Are there other types of structures (e.g., involving sequences of operations) that also lead to models with strong bounds? In fact, there are pseudo-polynomial models that, instead of using a sequence of arcs to form a path (which is an extremal solution), use a sequence of operations to form a solution that is extremal. Examples are Dyckhoff (1981) for the CSP, and Silva, Alvelos, and Valério de Carvalho (2010) and Furini, Malaguti, and Thomopoulos (2016) for the two-dimensional CSP with guillotine cuts. In these cases, one-cut operations are/have to be combined to form a cutting pattern (the extremal solution).

There are pseudo-polynomial models that do not provide LP lower bounds as strong as those of column generation (e.g., position indexed models for two-dimensional non-guillotine CSP). Is there any structural (extremal) property, in these cases, that can be explored and may lead to models with stronger bounds?

Several solution methods for large-scale arc flow models, like the ones presented in Section 6, are general and can be applied to any arc flow model based on DP. A software/library containing such general tools to solve general large-scale arc flow models would be an interesting contribution to the optimization and operations research community.

Acknowledgment

We thank three anonymous referees for their careful reviews. Their comments helped improve the clarity of the presentation and the overall quality of the paper. The first and fourth authors have been supported by FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo (under grant numbers 2017/11831-1 and 2019/12728-5). The second and the fifth authors have been supported by FCT - Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Prentice-Hall.

van den Akker, J. M., Hurkens, C. A. J., & Savelsbergh, M. W. P. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing, 12*(2), 111–124.

Alves, C., & Valério de Carvalho, J. M. (2008). A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research, 35*(4), 1315–1328.

Bazaraa, M. S., Jarvis, J. J., & Sherali, H. D. (2011). *Linear programming and network flows*. John Wiley & Sons.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Ben Amor, H., Desrosiers, J., & Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research, 54*(3), 454–463.

Benkirane, M., Clautiaux, F., Damay, J., & Detienne, B. (2019). A hypergraph model for the rolling stock rotation planning and train selection. working paper or preprint, december. URL https://hal.inria.fr/hal-02402447.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2017a). The continuous-time service network design problem. *Operations Research, 65*(5), 1303–1321.

Boland, N., Hewitt, M., Vu, D. M., & Savelsbergh, M. (2017b). Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin, & M. Lombardi (Eds.), *CPAIOR: Integration of AI and OR techniques in constraint programming* (pp. 254–262). Springer International Publishing.

Boland, N. L., & Savelsbergh, M. W. P. (2019). Perspectives on integer programming for time-dependent models. *TOP, 27*, 147–173.

Bouarab, H., El Hallaoui, I., Metrane, A., & Soumis, F. (2017). Dynamic constraint and variable aggregation in column generation. *European Journal of Operational Research, 262*(3), 835–850.

Boyd, E. A. (1992). A pseudopolynomial network flow formulation for exact knapsack separation. *Networks, 22*(5), 503–514.

Braga, N., Alves, C., & Macedo, R. (2017). Exact solution of the multi-trip inventory routing problem using a pseudo-polynomial model. In *Proceedings of the sixth international conference on operations research and enterprise systems: vol. 2* (pp. 250–257).

Braga, N., Alves, C., Macedo, R., & Valério de Carvalho, J. (2016). Combined cutting stock and scheduling: A matheuristic approach. *International Journal of Innovative Computing and Applications, 7*(3), 135–146.

Brandão, F., & Pedroso, J. P. (2015). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research, 69*, 56–67.

Bulhões, T., Sadykov, R., Subramanian, A., & Uchoa, E. (2020). On the exact solution of a large class of parallel machine scheduling problems. *Journal of Scheduling, 23*, 411–429.

Cacchiani, V., & Toth, P. (2012). Nominal and robust train timetabling problems. *European Journal of Operational Research, 219*(3), 727–737.

Cambazard, H., & O'Sullivan, B. (2010). Propagating the bin packing constraint using linear programming. In D. Cohen (Ed.), *Principles and practice of constraint programming* (pp. 129–136). Berlin Heidelberg: Springer.

Cappanera, P., & Scutellà, M. G. (2015). Joint assignment, scheduling, and routing models to home care optimization: A pattern-based approach. *Transportation Science, 49*(4), 830–852.

Caprara, A., Dell'Amico, M., Díaz-Díaz, J. C., Iori, M., & Rizzi, R. (2015). Friendly bin packing instances without integer round-up property. *Mathematical Programming, 150*, 5–17.

Caprara, A., Fischetti, M., & Toth, P. (2002). Modeling and solving the train timetabling problem. *Operations Research, 50*(5), 851–861.

Cattaruzza, D., Absi, N., & Feillet, D. (2016). Vehicle routing problems with multiple trips. *4OR, 14*(3), 223–259.

Christofides, N., & Hadjiconstantinou, E. (1995). An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. *European Journal of Operational Research, 83*(1), 21–38.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks, 11*(2), 145–164.

Cire, A. A., Diamant, A., Yunes, T., & Carrasco, A. (2019). A network-based formulation for scheduling clinical rotations. *Production and Operations Management, 28*(5), 1186–1205.

Clautiaux, F., Hanafi, S., Macedo, R., Voge, M.-E., & Alves, C. (2017). Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research, 258*(2), 467–477.
Clautiaux, F., Sadykov, R., Vanderbeck, F., & Viaud, Q. (2018). Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization, 29*, 18–44.
Conforti, M., Cornuéjols, G., & Zambelli, G. (2010). Extended formulations in combinatorial optimization. *4OR, 8*, 1–48.
Côté, J.-F., & Iori, M. (2018). The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing, 30*(4), 646–661.
Dantzig, G. B., & Wolfe, P. (1961). The decomposition algorithm for linear programs. *Econometrica, 29*(4), 767–778.
Dell'Amico, M., Delorme, M., Iori, M., & Martello, S. (2019). Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research, 274*(3), 886–899.
Delorme, M., & Iori, M. (2020). Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing, 32*(1), 101–119.
Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research, 255*(1), 1–20.
Delorme, M., Iori, M., & Martello, S. (2017). Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research, 78*, 290–298.
Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2006). *Column generation*. Springer Science & Business Media.
Desaulniers, G., Gschwind, T., & Irnich, S. (2020). Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. *Transportation Science, 54*(5), 1153–1438.
Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research, 40*(2), 342–354.
Dyckhoff, H. (1981). A new linear programming approach to the cutting stock problem. *Operations Research, 29*(6), 1092–1104.
Elhallaoui, I., Metrane, A., Desaulniers, G., & Soumis, F. (2011). An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing, 23*(4), 569–577.
Elhallaoui, I., Villeneuve, D., Soumis, F., & Desaulniers, G. (2005). Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research, 53*(4), 632–645.
Eppen, G. D., & Martin, R. K. (1987). Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research, 35*(6), 832–848.
Fischer, F., & Helmberg, C. (2014). Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming, 143*, 257–297.
Ford, L. R., Jr., & Fulkerson, D. R. (1958a). A suggested computation for maximal multi-commodity network flows. *Management Science, 5*(1), 97–101.
Ford, L. R., Jr., & Fulkerson, D. R. (1958b). Constructing maximal dynamic flows from static flows. *Operations Research, 6*(3), 419–433.
Furini, F., Malaguti, E., & Thomopoulos, D. (2016). Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing, 28*(4), 736–751.
Gabrel, V., & Murat, C. (2003). Mathematical programming for earth observation satellite mission planning. In T. A. Ciriani, G. Fasano, S. Gliozzi, & R. Tadei (Eds.), *Operations research in space and air* (pp. 103–122). Springer US.
Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research, 9*(6), 849–859.
Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting stock problem - part II. *Operations Research, 11*(6), 863–888.
Godinho, M. T., Gouveia, L., Magnanti, T., Pesneau, P., & Pires, J. (2007). On time-dependent models for unit demand vehicle routing problems. In *International network optimization conference (INOC)*.
Gouveia, L., Leitner, M., & Ruthmair, M. (2019). Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research, 102*, 22–38.
Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Hammer, P. L., Johnson, E. L., & Korte, B. H. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Discrete optimization II*. In *Annals of Discrete Mathematics: 5* (pp. 287–326). Elsevier.
Gualandi, S., & Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing, 24*(1), 81–100.
van Hoeve, W. J. (2020). Graph coloring lower bounds from decision diagrams. In D. Bienstock, & G. Zambelli (Eds.), *Integer programming and combinatorial optimization* (pp. 405–418). Springer International Publishing.
Hooker, J. N. (2013). Decision diagrams and dynamic programming. In *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems* (pp. 94–110). Springer.
Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, & M. M. Solomon (Eds.), *Column generation* (pp. 33–65). Boston, MA: Springer US.
Irnich, S., Desaulniers, G., Desrosiers, J., & Hadjar, A. (2010). Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing, 22*(2), 297–313.
Irnich, S., Toth, P., & Vigo, D. (2014). The family of vehicle routing problems. In P. Toth, & D. Vigo (Eds.), *Vehicle routing: Problems, methods, and applications, chapter 1* (pp. 1–33). SIAM.
Kartak, V. M., Ripatti, A. V., Scheithauer, G., & Kurz, S. (2015). Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics, 187*, 120–129.
Kramer, A., Dell'Amico, M., Feillet, D., & Iori, M. (2020a). Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time. *Computers & Operations Research, 123*. Article 105018
Kramer, A., Dell'Amico, M., & Iori, M. (2019a). Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research, 275*(1), 67–79.
Kramer, A., Iori, M., & Lacomme, P. (2021). Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. *European Journal of Operational Research, 289*(3), 825–840.
Kramer, A., Lalla-Ruiz, E., Iori, M., & Voß, S. (2019b). Novel formulations and modeling enhancements for the dynamic berth allocation problem. *European Journal of Operational Research, 278*(1), 170–185.
Kramer, R., Iori, M., & Vidal, T. (2020b). Mathematical models and search algorithms for the capacitated-center problem. *INFORMS Journal on Computing, 32*(2), 444–460.
Lancia, G., Rinaldi, F., & Serafini, P. (2011). A time-indexed LP-based approach for min-sum job-shop problems. *Annals of Operations Research, 186*, 175–198.
Lancia, G., & Serafini, P. (2014). Deriving compact extended formulations via LP-based separation techniques. *4OR, 12*(3), 201–234.
Lemaréchal, C. (2001). Lagrangian relaxation. In M. Jünger, & D. Naddef (Eds.), *Computational combinatorial optimization: Optimal or provably near-optimal solutions* (pp. 112–156). Berlin, Heidelberg: Springer Berlin Heidelberg.
de Lima, V. L., Iori, M., & Miyazawa, F. K. (2021). New exact techniques applied to a class of network flow formulations. In M. Singh, & D. P. Williamson (Eds.), *Integer Programming and Combinatorial Optimization* (pp. 178–192). Springer International Publishing. Forthcoming
Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research, 53*(6), 1007–1023.
Macedo, R., Alves, C., & Valério de Carvalho, J. M. (2010). Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research, 37*(6), 991–1001.
Macedo, R., Alves, C., Valério de Carvalho, J. M., Clautiaux, F., & Hanafi, S. (2011). Solving exactly the vehicle routing problem with time windows and multiple routes using a pseudo-polynomial model. *European Journal of Operational Research, 214*(3), 536–545.
Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, Inc..
Martin, R. K., Rardin, R. L., & Campbell, B. A. (1990). Polyhedral characterization of discrete dynamic programming. *Operations Research, 38*(1), 127–138.
Martinovic, J., Delorme, M., Iori, M., Scheithauer, G., & Strasdat, N. (2020). Improved flow-based formulations for the skiving stock problem. *Computers & Operations Research, 113*. Article 104770
Martinovic, J., & Scheithauer, G. (2016a). The proper relaxation and the proper gap of the skiving stock problem. *Mathematical Methods of Operations Research, 84*(3), 527–548.
Martinovic, J., & Scheithauer, G. (2016b). Integer linear programming models for the skiving stock problem. *European Journal of Operational Research, 251*(2), 356–368.
Martinovic, J., Scheithauer, G., & Valério de Carvalho, J. M. (2018). A comparative study of the arcflow model and the one-cut model for one-dimensional cutting stock problems. *European Journal of Operational Research, 266*(2), 458–471.
du Merle, O., Villeneuve, D., Desrosiers, J., & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics, 194*(1), 229–237.
Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM, 7*(4), 326–329.
Minoux, M. (2006). Multicommodity network flow models and algorithms in telecommunications. In M. G. C. Resende, & P. M. Pardalos (Eds.), *Handbook of optimization in telecommunications* (pp. 163–184). Springer US.
Mrad, M. (2015). An arc flow-based optimization approach for the two-stage guillotine strip cutting problem. *Journal of the Operational Research Society, 66*(11), 1850–1859.
Mrad, M., & Souayah, N. (2018). An arc-flow model for the makespan minimization problem on identical parallel machines. *IEEE Access, 6*, 5300–5307.
Nadarajah, S., & Cire, A. A. (2017). Network-based approximate linear programming for discrete optimization. *SSRN Electronic Journal*.
Nemhauser, G., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. Wiley & Sons.
Nesello, V., Delorme, M., Iori, M., & Subramanian, A. (2018). Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society, 69*(3), 326–339.
Pecin, D., Pessoa, A., Poggi, M., & Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation, 9*, 61–100.
Pessoa, A., de Aragão, M. P., & Uchoa, E. (2008). Robust branch-cut-and-price algorithms for vehicle routing problems. In B. Golden, S. Raghavan, & E. Wasil (Eds.), *The vehicle routing problem: Latest advances and new challenges* (pp. 297–325). Springer US.
Pessoa, A., Sadykov, R., Uchoa, E., & Vanderbeck, F. (2019). A generic exact solver for vehicle routing and related problems. In A. Lodi, & V. Nagarajan (Eds.), *Integer programming and combinatorial optimization* (pp. 354–369). Springer International Publishing.
Pessoa, A., Uchoa, E., de Aragão, M. P., & Rodrigues, R. (2010). Exact algorithm

over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation, 2*, 259–290.
Picard, J.-C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research, 26*(1), 86–110.
Poggi, M., & Uchoa, E. (2014). New exact algorithms for the capacitated vehicle routing problem. In P. Toth, & D. Vigo (Eds.), *Vehicle routing: Problems, methods, and applications, chapter 3* (pp. 59–86). SIAM.
Ramos, B., Alves, C., & Valério de Carvalho, J. (2020). An arc flow formulation to the multitrip production, inventory, distribution, and routing problem with time windows. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.12765. Forthcoming
Rao, M. R. (1976). On the cutting stock problem. *Journal of the Computer Society of India, 7*, 35–39.
Ratli, M., Benmansour, R., Macedo, R., Hanafi, S., & Wilbaut, C. (2013). Mathematical programming and heuristics for scheduling problems with early and tardy penalties. In B. Jarboui, P. Siarry, & J. Teghem (Eds.), *Metaheuristics for production scheduling, chapter 8* (pp. 183–223). John Wiley & Sons, Ltd.
Riedler, M., Jatschka, T., Maschler, J., & Raidl, G. R. (2020). An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research, 27*(1), 573–613.
Riedler, M., Ruthmair, M., & Raidl, G. R. (2019). Strategies for iteratively refining layered graph models. In M. J. Blesa Aguilera, C. Blum, H. Gambini Santos, P. Pinacho-Davidson, & J. Godoy del Campo (Eds.), *Hybrid metaheuristics* (pp. 46–62). Springer International Publishing.
Rietz, J., Alves, C., Braga, N., & Valério de Carvalho, J. M. (2016). An exact approach based on a new pseudo-polynomial network flow model for integrated planning and scheduling. *Computers & Operations Research, 76*, 183–194.
Righini, G., & Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks, 51*(3), 155–170.
Sadykov, R., & Vanderbeck, F. (2013). Column generation for extended formulations. *EURO Journal on Computational Optimization, 1*, 81–115.
Shapiro, J. F. (1968). Dynamic programming algorithms for the integer programming problem-I: The integer programming problem viewed as a knapsack type problem. *Operations Research, 16*(1), 103–121.
Silva, E., Alvelos, F., & Valério de Carvalho, J. M. (2010). An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research, 205*(3), 699–708.
Skutella, M. (2009). An introduction to network flows over time. In W. Cook, L. Lovász, & J. Vygen (Eds.), *Research trends in combinatorial optimization: Bonn 2008* (pp. 451–482). Berlin, Heidelberg: Springer.
Sousa, J. P., & Wolsey, L. A. (1992). A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming, 54*(1), 353–367.
Toth, P., & Vigo, D. (2014). *Vehicle routing* (2nd). SIAM. https://doi.org/10.1137/1.9781611973594.
Trindade, R. S., de Araújo, O. C. B., & Fampa, M. (2020). Arc-flow approach for parallel batch processing machine scheduling with non-identical job sizes. In M. Baïou, B. Gendron, O. Günlük, & A. R. Mahjoub (Eds.), *Combinatorial optimization* (pp. 179–190). Springer International Publishing.
Uchoa, E. (2012). Cuts over extended formulations by flow discretization. In A. R. Mahjoub (Ed.), *Progress in combinatorial optimization, chapter 8* (pp. 255–282). Wiley.
Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research, 86*, 629–659.
Valério de Carvalho, J. M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research, 141*(2), 253–273.
Valério de Carvalho, J. M. (2005). Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing, 17*(2), 175–182.
Vance, P. H. (1998). Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications, 9*(3), 211–228.
Voge, M.-E., & Clautiaux, F. (2012). Theoretical investigation of aggregation in pseudo-polynomial network-flow models. In A. R. Mahjoub, V. Markakis, I. Milis, & V. T. Paschos (Eds.), *Combinatorial optimization* (pp. 213–224). Springer Berlin Heidelberg.
Vu, D. M., Hewitt, M., Boland, N., & Savelsbergh, M. (2020). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science, 54*(3), 703–720.
Wang, X., Wu, G., Xing, L., & Pedrycz, W. (2020). Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. *IEEE Systems Journal*. https://doi.org/10.1109/jsyst.2020.2997050. Forthcoming
Wentges, P. (1997). Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research, 4*(2), 151–162.
Wolsey, L. A. (1977). Valid inequalities, covering problems and discrete dynamic programs. *Annals of Discrete Mathematics, 1*, 527–538.