Journal of Computational Mathematics
Vol.37, No.6, 2019, 866–888.

http://www.global-sci.org/jcm
doi:10.4208/jcm.1905-m2019-0055

TACKLING INDUSTRIAL-SCALE SUPPLY CHAIN PROBLEMS BY MIXED-INTEGER PROGRAMMING*

Gerald Gamrath, Ambros Gleixner, Thorsten Koch and Matthias Miltenberger,
Zuse Institute Berlin, Department Optimization
Email: gamrath@zib.de, gleixner@zib.de, koch@zib.de, miltenberger@zib.de
Dimitri Kniasew and Dominik Schlögel
SAP SE, SAP Optimization
Email: dimitri.kniasew@sap.com, dominik.schloegel@sap.com
Alexander Martin and Dieter Weninger
Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Mathematics
Email: alexander.martin@math.uni-erlangen.de, dieter.weninger@math.uni-erlangen.de

Abstract

The modeling flexibility and the optimality guarantees provided by mixed-integer programming greatly aid the design of robust and future-proof decision support systems. The complexity of industrial-scale supply chain optimization, however, often poses limits to the application of general mixed-integer programming solvers. In this paper we describe algorithmic innovations that help to ensure that MIP solver performance matches the complexity of the large supply chain problems and tight time limits encountered in practice. Our computational evaluation is based on a diverse set, modeling real-world scenarios supplied by our industry partner SAP.

Mathematics subject classification: 90B06, 90C05, 90C06, 90C11, 90C90.
Key words: Supply chain management, Supply network optimization, Mixed-integer linear programming, Primal heuristics, Numerical stability, Large-scale optimization.

1. Introduction

In the late 1990s, the need for advanced business software to face the challenges of ongoing globalization had become ubiquitous and that need has been continuously increasing ever since then. Thus, various software vendors began to offer so-called advanced planning systems (APS) that helped companies plan and coordinate their growing supply chains and avoid potential bottlenecks in their resources such as labor, material, and machinery.

Many of the underlying questions can naturally be phrased as complex mathematical optimization problems. This article focuses on supply network planning, an optimization task that amounts to the computation of medium- and long-term plans for material procurement, production, transportation, demand fulfillment, stock keeping, and resource capacity utilization over large time horizons and various organizational units such as raw material suppliers, plants, warehouses, and transportation facilities. Some of the quantities may be produced and transported only in discrete lots. The goal is the minimization of the overall business-related costs, which consist of actual costs, e.g., for stock keeping, production, and transportation, as well as penalty costs for missing demand fulfillment [1].

* Received February 26, 2019 / Revised version received May 5, 2019 / Accepted July 19, 2019 / Published online September 13, 2019 /

Mathematically, the resulting optimization problems can be formulated as mixed-integer linear programs (MIPs), which take the general form

$$\min\left\{c^T x : Ax \geq b, x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}\right\},$$

where $A \in \mathbb{R}^{(n_1+n_2)\times m}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^{n_1+n_2}$. This has many benefits. The expressivity and flexibility of MIP allows to represent many different use cases in one general model. Additionally, solvers provide guarantees on the quality of the supply network plans produced even if the problems are too hard to be solved to optimality. The high feature complexity and level of detail present in real-world supply network problems, however, pose challenges to current state-of-the-art optimization software.

In this article, we describe how these challenges can be addressed even for large-scale real-world supply chain problems in a software product delivered to SAP® users worldwide. In particular, we want to emphasize how the choice for using general MIP solvers as the underlying optimization engine allows to achieve an integrated and general handling of models for different supply chain structures that can easily be adapted and extended for future requirements. On the mathematical side, we describe algorithmic techniques developed inside the academic MIP solver SCIP[1)] [2] in order to improve computational performance on MIP formulations with supply chain structure.

The paper is organized as follows. In Sections 2 and 3, we present the basic MIP models solved and discuss the benefits and challenges of the MIP approach. In Section 4, we explain high-level decomposition techniques that are applied in order to break down large MIP formulations to sizes that can be handled by a general MIP solver within the running time requirements imposed by practitioners. Sections 5 to 8 are dedicated to the algorithmic advances inside the MIP solver SCIP that have been developed to alleviate the computational challenges. In Section 9, we demonstrate their performance impact on a test set derived from diverse real-world supply chain scenarios. Section 10 summarizes our findings.

2. A Mixed-integer Model for Optimal Supply Network Planning

In the following, we detail the base formulation of a mixed-integer model used to optimize medium- or long-term plans for general supply networks including typical supply chain processes such as procurement, production, transportation, and customer-demand fulfillment. This model is an integral part of the Supply Network Planning Optimization (SNP Optimization) function delivered worldwide to users of the SAP® Advanced Planning and Optimization component.[2)] The supply chain plans may cover a time interval of several years and include various organizational units of the supply network (locations) such as raw material suppliers, plants, warehouses, and transportation facilities (see [1] for details). The objective of the model is to minimize the overall business-related costs incurred by stock keeping, production, transport, or missing demand fulfillment. Furthermore, it considers scarce resource capacities required by production and transport activities. Large-scale scenarios may contain up to several thousand products and hundreds of locations.

The basic decisions of the supply chain model to be made are the quantities of material procurement, production, transportation, demand fulfillment, stock keeping, and resource capacity utilization. Some quantities may be produced and transported only in discrete lots. The

[1)] See http://scip.zib.de/

[2)] SAP is a registered trademark of SAP SE in Germany and in several other countries.

granularity of these quantities is determined according to the coarse temporal structure of the planning interval (horizon), which is subdivided into periods, called time buckets or simply buckets. Typically buckets represent days, weeks, or months. The optimization has to make the above decisions for every bucket where possible. Since early decisions often have to be more fine-grained and precise than later decisions, many practitioners prefer using a so-called telescopic bucket scheme. As an example, a telescopic scheme may divide the horizon into daily buckets for short-term decisions, weekly buckets for mid-term decisions, and monthly buckets for late-term decisions. In many scenarios the planning horizon comprises one or two years and may consist of 25 to 100 buckets.

The supply chain structure itself yields numerous different constraints. These emerge from restrictions entered by the user like limited resource capacities but also from essential requirements as stock balance consistency. We first state the core of the model. Refinements and extensions are provided in the next section.

2.1. The core model

The formulation is based on the index sets, parameters, and variables summarized in Table 2.1. All variables are given a default lower bound of zero. Upper bounds are implied by the constraints.

The main goal of supply chain management is to satisfy requested demands. However, at this stage there are no cost-relevant benefits or rewards generated by satisfying a demand. Instead, in order to initiate activity within the supply chain, despite the costs of production and transportation, the model incorporates non-delivery penalties.

A demand d of product p may allow a late delivery by at most $\delta_{dp}(t)$ time buckets. To obtain the non-delivered amount regarding some demand in bucket t, all deliveries for that demand, including delayed deliveries, are subtracted from the original demand quantity in that bucket:

$$x_{dp}(t) = \alpha_{dp}(t) - \sum_{t'=t}^{t+\delta_{dp}(t)} y_{dp}(t,t'), \quad \forall d \in D, \ \forall p \in P, \ \forall t \in T. \tag{2.1}$$

The stock level balance equation contains all information on input and output to and from a location by production, transport, or procurement, as well as on stock quantities from the previous time bucket. Consider time buckets $\tau, t \in T$ with $\tau \leq t$. Possible late deliveries to satisfy demands of product p from earlier time buckets $\tau < t$ as well as on time delivery $\tau = t$, need also to be taken into account in the stock level balance equation, as long as their maximum allowed lateness $\delta_{dp}(\tau)$ permits a delivery up to bucket t:

$$\begin{aligned} s_{lp}(t) = s_{lp}(t-1) + v_{lp}(t) &+ \sum_{o \in O_l} \pi^{+}_{op}(t) \cdot u_o(t) + \sum_{a \in A^l} z_{ap}(t) \\ &- \sum_{o \in O_l} \pi^{-}_{op}(t) \cdot u_o(t) - \sum_{a \in A_l} z_{ap}(t) - \sum_{d \in D_l} \sum_{\tau \leq t} y_{dp}(\tau, t), \\ &\forall l \in L, \ \forall p \in P_l, \ \forall t \in T. \end{aligned} \tag{2.2}$$

Resource restrictions can be considered in different activities. For simplicity, we refer only to production as a showcase. Resource capacities in production usually represent allocatable time on production machines or available man hours. Therefore different product lines may share the same resources. Usually there are no costs for resource consumption. Production

Table 2.1: Notation for index sets, parameters, and decision variables of the model.

Symbol	Index set
T	Time buckets
L	Locations
D	Demands
D_t	Demands in time bucket $t \in T$
D_l	Demands at location $l \in L$
A	Arcs
A^l	Arcs with location $l \in L$ as destination
A_l	Arcs with location $l \in L$ as origin
P	Products
P_l	Products handled at location $l \in L$
P_a	Products transported on arc $a \in A$
O_l	Production models at location $l \in L$
R_l	Resources at location $l \in L$
Symbol	Parameter
$\alpha_{dp}(t)$	Quantity of demand d for product p in bucket t
$\delta_{dp}(t)$	Maximum allowed lateness for demand d for product p in bucket t
$\beta_{dp}(t)$	Non-delivery costs for demand d for product p in bucket t
$\gamma_{dp}(t,t')$	Late delivery costs for delivering demand $d \in D_t$ for product p in bucket $t' > t$, i.e., $(t'-t)$ buckets late
$\zeta_{ap}(t)$	Costs for transporting one unit of product p via arc a in bucket t
$\eta_o(t)$	Costs of applying production model $o \in O_l$ once at location l in bucket t
$\theta_{lp}(t)$	Costs of procuring product p at location l in bucket t
$\pi^+_{op}(t)$	Output quantity of product p from production model o in bucket t
$\pi^-_{op}(t)$	Input quantity of product p to production model o in bucket t
$\sigma_{lp}(t)$	Costs of storing product p at location l in bucket t
$\rho_{or}(t)$	Requirement of resource r for production model o in bucket t
$\psi_r(t)$	Capacity of resource r in bucket t
$\varphi_o(t)$	Quantity of minimum lot size
M	Large constant used in big-M method
Symbol	Decision variable
$x_{dp}(t)$	Quantity not delivered for demand d of product p in bucket t
$y_{dp}(t,t')$	Quantity delivered in bucket t' for demand d of product p in bucket t
$z_{ap}(t)$	Quantity of product p transported on arc a in bucket t
$u_o(t)$	Number of applications of production model $o \in O_l$ at location l in bucket t
$v_{lp}(t)$	Quantity procured of product p at location l in bucket t
$s_{lp}(t)$	Stock level at location l of product p in bucket t
$w_o(t)$	Binary indicator variable for minimum lot sizes

resource capacity restrictions are realized by

$$\sum_{o \in O_l} \rho_{or}(t) \cdot u_o(t) \leq \psi_r(t), \quad \forall l \in L, \ \forall r \in R_l, \ \forall t \in T. \tag{2.3}$$

Note, because of $r \in R_l$ the capacity $\psi_r(t)$ of resource r in bucket t is related to location l.

The minimum lot size for production describes the minimum number of times a production model should be executed in case a production takes place. Two constraints are needed to model minimum lot sizes. First for the decision if production takes place or not, which is modeled with the big-M method, and second for the minimum quantity requirement:

$$u_o(t) - M \cdot w_o(t) \leq 0, \qquad \forall l \in L,\ \forall o \in O_l,\ \forall t \in T, \tag{2.4}$$

$$\varphi_o(t) \cdot w_o(t) - u_o(t) \leq 0, \qquad \forall l \in L,\ \forall o \in O_l,\ \forall t \in T. \tag{2.5}$$

Finally, the objective function accumulates all emerging costs:

$$\begin{aligned} \min_{\substack{x,y,z\\u,v,w,s}} \sum_{t \in T} \Bigg(& \sum_{d \in D_t} \sum_{p \in P} \sum_{t' > t}^{t+\delta_{dp}(t)} \big(\gamma_{dp}(t,t') \cdot y_{dp}(t,t') + \beta_{dp}(t) \cdot x_{dp}(t)\big) \\ & + \sum_{a \in A} \sum_{p \in P_a} \zeta_{ap}(t) \cdot z_{ap}(t) + \sum_{l \in L} \sum_{o \in O_l} \eta_o(t) \cdot u_o(t) \\ & + \sum_{l \in L} \sum_{p \in P_l} \big(\theta_{lp}(t) \cdot v_{lp}(t) + \sigma_{lp}(t) \cdot s_{lp}(t)\big) \Bigg). \end{aligned} \tag{2.6}$$

Note that, in practice, variables are generated sparsely, i.e., only for relevant index combinations for which their value can be non-zero. For notational convenience and readability, however, the model above is formulated densely, e.g., using variables $x_{dp}(t)$ for all $p \in P$ and $t \in T$ although one demand may only involve a subset of products and time buckets.

2.2. Further features

The variables and constraints described above comprise the basic core of the mathematical model. In addition, the SNP Optimization supports further features that cannot be discussed here in detail due to lack of space, for instance:

- fixed lot sizes: the requirement to produce only multiples of a fixed quantity
- safety stock: try to keep material inventory at a certain minimum level
- shelf life: limit material inventory not to exceed a certain maximum level
- extension capacity: extend resource capacity at some costs
- cost functions: convex and concave piecewise linear cost functions on basic decisions such as production, transport, or inventory
- setup costs: fixed costs for production of a material, independent of the quantity produced
- substitution: satisfy product demands by substitute products
- quota arrangements: try to keep quota arrangements between alternative sources of supply
- subcontracting: consider external suppliers of certain products
- fixed material flows: some production may incur fixed material flows independent of the number of produced lots
- fixed resource consumption: some production may incur fixed resource capacity consumption, independent of the number of produced lots

3. Benefits and Challenges of the MIP Approach

To find a feasible or even optimal solution for the supply chain problem, business-related data is mapped into a general mathematical model as described in the previous section. After building the mathematical model, a MIP solver is invoked for optimization. The result is then converted back into a supply chain plan proposed to the practitioner. In practice, some practitioners create remarkably large scenarios, resulting in MIP models with up to 30 million variables and constraints, where up to 500,000 of the variables can be integer.

3.1. MIP vs. heuristics

Despite the large complexity of the resulting optimization problems, it was decided from the very beginning of the SNP Optimization development to employ an exact MIP approach instead of heuristic algorithms. The main reasons were threefold:

- the feature complexity of real-world supply chain problems,
- the extensibility of MIP models, and
- the possibility to evaluate feasibility and solution quality.

In the face of the feature complexity of the supply chain problems to be solved, one strong argument for MIP was the expressiveness of general MIP models. Besides multiple stages in production, capacity constraints, and discrete lot sizes, many supply chain-specific hard or soft restrictions may occur in practice. Some examples are fulfillment of safety stock, product interchangeability, or the consideration of shelf life. Although many heuristics exist for supply chain planning, they usually are specialized on a subset of the features above. To the best of our knowledge, a generalized heuristic that would consider all those aspects is neither available on the software market nor described in literature. However, for practitioners it is often crucial to be able to address all planning features and business requirements in a single optimization framework.

Another essential advantage of the MIP approach that is related, but slightly different, is the extensibility of MIP models. Since the market launch of SNP Optimization, SAP had to address numerous user requests for additional planning capabilities. Generally, extending a sophisticated heuristic that is already based on multiple interdependent and calibrated algorithms often requires significant redesign and testing effort. Consequently, with an increasing number of additional features the heuristic is likely to collapse under its growing complexity. However, with the MIP approach, adding new features is far easier to accomplish by incrementally adding further constraints and variables to the existing model, thereby keeping its correctness and stability.

Last but not least, a major benefit of MIP solvers is their capability to evaluate whether the supply chain problem is solvable at all and to estimate the gap between a best known and the best possible solution, i.e., a global optimum [1]. If the model formulation is too restrictive or contradictory, that is, if it contains a combination of constraints that can never be fulfilled, the MIP solver can prove this infeasibility. In our experience, this often succeeds quickly before the solver starts the branch-and-bound search for a solution. Thus, it is possible to inform the planner about the infeasibility of the input data.

If, on the contrary, the model formulation is correct but the MIP solver fails to find the global optimum within the given time limit, it will at least provide a guarantee about the actual solution quality. This gives the supply chain planner a quantitative basis to decide whether to optimize with an increased time limit or whether the solution is already good enough. Most heuristic approaches lack all these benefits.

3.2. Numerical stability

Despite the advantages described above, the MIP approach may exhibit some drawbacks. One typical issue that can have a significant impact on the solver performance is numerical instability of the model. If the optimization model is directly derived from a practitioner's business data, it usually contains a wide range of coefficients and constants.

Very large coefficients may occur in the objective function acting as pseudo-hard penalties for not delivering a customer demand or for missing a minimum stock level (safety stock). Other examples are variables or constraints limited by extremely high bounds to represent an almost unlimited production or transport decision. Very small coefficients often occur in the objective function representing low costs or in the constraints representing material flow coefficients, stemming from internal unit conversions. In typical scenarios the coefficients in the model span from 0.001 to 10^7, i.e., ten orders of magnitude.

As frequently described in the literature, too wide a range of numbers inside a single MIP model is likely to deteriorate the numeric stability and significantly reduce the solver performance (see for example [3]).

3.3. Scalability

Besides numerical issues, the major challenge of the MIP approach is to ensure high scalability of running time and solution quality as the size of the model increases. As pointed out, the supply chains of some practitioners lead to MIP models with up to 30 million variables and constraints, where up to 500,000 of the variables may be integer. Given the high complexity of MIP, no generic state-of-the-art solver can guarantee to find optimal solutions efficiently, say within predictable running times growing linearly with the size of the model. In practice, this indicates the risk of either long solving times or low solution quality within an acceptable time frame. In some extreme cases, a MIP solver will not find any solution at all after many hours or even days. However, the solving time expected by users is typically limited to few hours. When running the tool manually on smaller scenarios, users usually expect a solving time of few minutes. In the following, we describe techniques to tackle these challenges.

4. Decomposition Techniques

One classic possibility to address the scalability challenge is to apply decomposition techniques. Here, the idea is to divide the optimization problem into several subproblems and solve these separately instead of solving the problem as a whole. The solutions of the subproblems are then combined to form the overall solution. Hence, decomposition techniques overcome the performance bottleneck of large problems and often yield feasible and good solutions within an acceptable time frame while keeping the memory consumption low.

In our framework, decomposition is applied both inside the MIP solver as well as a priori before passing problems to the MIP solvers. While the techniques inside the MIP solver, which will be described later, are all exact, some of the high-level decomposition techniques on the modeling level are heuristic in the sense that they may compromise solution quality in order to deal with the large dimension of some supply chain problems. In the following, we describe the decomposition techniques applied in SNP Optimization outside of the MIP solver.

4.1. Mathematical decomposition

The simplest way to decompose the MIP model is when completely independent subproblems can be identified, i.e., disjoint groups of variables that are not linked by any constraints. In this case, the objective function values and objective bounds of the subproblems (here: lower bounds for a minimization problem) can be summed up after solving them individually to obtain the final objective value and bound without losing solution quality.

However, even when this case is detected there is a risk that the model is not well decomposable, i.e., that some of the independent subproblems turn out to be significantly larger than the remaining subproblems and cannot be split up further. This happens when variables occur in multiple constraints, for example, variables representing stock levels of raw materials, which are used throughout the whole supply chain. In this case, one would expect only a minimal performance improvement from this decomposition technique.

4.2. Decomposition under business aspects

To lower the risk of inhomogeneous subproblems described above, SNP Optimization offers optional decomposition techniques that allow subproblems to overlap to a limited extent, thereby considering additional business-related knowledge of the supply chain:

- Decisions in earlier periods are often more critical than decisions in later periods.
- Some demands are more important than others and should be planned initially.
- Some product lines are more important than others and should be planned primarily.

The first aspect is covered by the so-called time decomposition, which solves the supply chain problem in separate overlapping time windows, thereby gliding forwards in time. The second aspect is exploited by the so-called priority decomposition. It first solves the supply chain problem containing only demands of the highest priority. Afterwards it solves the same problem with demands of the second-highest priority, while keeping the first solution fixed, and so on. The third aspect is addressed by the product decomposition, which will be discussed in detail in the following subsection.

Note that the price of these decomposition techniques is the loss of our capability to find the optimal solution and to estimate the lower cost bound. The final solution will be feasible but we cannot guarantee optimality. The lower bounds of the individual subproblems cannot be combined to estimate the overall lower cost bound. Despite these limitations many practitioners often prefer decomposition, in particular when planning large supply chains. The availability of these decomposition approaches in conjunction with an exact model enable the practitioner to evaluate the trade-off between running time and solution quality for typical input data in an *a priori* study.

4.3. Product decomposition

The product decomposition is the most commonly used decomposition technique in practice. It identifies independent product-line structures within the given supply chain, extracts them into subproblems, and solves them sequentially in a specific order. In practice, it is often successful in breaking large scenarios into several hundred subproblems.

The subproblems of the product decomposition can overlap, since different product lines may require common resource capacity, provided by machines of a factory or labor, for instance. These capacity conflicts are resolved on a first-come, first-served basis: once a resource capacity has been consumed within an early subproblem, any later subproblem has to respect that consumption as a fixed boundary condition and may only utilize the remaining capacity of this resource.

Hence, the solving order of the subproblems influences the final solution in the overlapping case. To avoid the effect that earlier subproblems fully utilize capacities of shared resources, thereby leaving little or no capacity for later subproblems, the product decomposition can optionally apply a technique called preallocation. Here, as a heuristic preprocessing step, the linear programming relaxation of the whole optimization problem is solved, which ignores any discrete constraints. The resource occupancies of the linear solution then serve as further boundary conditions for the subproblems. Naturally, this option is applicable only for supply chain models with binary or integer-valued variables.

The product decomposition can succeed only if the underlying MIP solver solves all subproblems successfully. If it fails to find a feasible solution for only one subproblem, the whole approach fails. Consequently, the distribution of the overall user-defined run-time to the individual subproblems is critical for the success of this decomposition technique.

The simplest way is to evenly assign the run-time to the subproblems. This strategy works for most problems in practice. A more complex way is to assign time slots proportionally to the estimated complexity of the subproblems. SNP Optimization offers various methods to estimate problem complexity based on the input data. If a subproblem is not solved within its time slot, the remaining time will be used until the first solution is found. Afterwards, the residual time will be redistributed to the remaining subproblems according to their complexity estimation. Besides choosing the time-distribution strategy, the user may also choose among several strategies for ordering the subproblem solving in order to avoid possible time bottlenecks and to improve the solution quality.

Note that product decomposition is not suitable for very dense supply chain models. If, for instance, a subproblem covers 90% of the supply chain model, product decomposition would not yield a significant performance improvement. In this case, other decomposition techniques, such as time decomposition, might be more suitable.

While the outlined decomposition techniques are applied on the business side with supply-chain-specific knowledge at hand, the following sections are devoted to the algorithmic improvements that have been implemented inside of the MIP solver SCIP that receives only the abstract mathematical MIP model.

5. Engineering the MIP Solver I: Presolving

Presolving is a collection of algorithms that reduce the size and, more importantly, improve the formulation of a given model. They aim at shrinking and tightening the linear programming (LP) relaxation such that it better describes the convex hull of the underlying mixed-integer solutions and becomes easier to solve. Applied in multiple rounds before starting the branch-and-bound search, these reductions help to decrease the number of nodes that need to be explored later and speed up their processing time. It has been shown that presolving is a powerful key factor in modern mixed-integer programming solvers [4, 5].

Supply chain instances are often designed in a way that they are convenient for applying presolving techniques: The underlying constraint matrix is mostly very sparse and equality constraints, e.g., from modeling stock level conditions, can be used to aggregate variables profitably. Many constraints consist only of continuous variables, where presolving techniques from the linear programming literature take effect. In addition, the regarded instances often contain independent components, i.e. parts of the original problem which share no common variables and constraints.

In the following, we describe three presolving techniques and accent frequently occurring substructures in supply chain instances predestinated for their employment. Furthermore, these substructures appear in many other mixed-integer problems as well, e.g., production planning and network design. Therefore, the following descriptions are a tutorial for identifying related substructures in other applications or designing similar custom-made presolving algorithms. Mathematical details of the presolving algorithms can be found in [6].

5.1. Singleton column stuffing

Convex piecewise linear functions play an important role in modeling cost structures with coefficients depending on the stock level value. They are frequently used, e.g., for modeling safety stock violation penalties, stock keeping costs, and maximum stock violation penalties. After applying aggregation-related presolving techniques, models of such functions very often yield continuous singleton columns.

A singleton column is a column of the constraint matrix with only one non-zero coefficient, i.e., it appears only in one row of the constraint matrix. During singleton column stuffing we determine for every row of the constraint matrix a set of continuous singleton columns. Then we consider each variable of such a set in a suitable order that is determined by the ratio of the objective function coefficient and the coefficient in the row and try to fix the corresponding variable at a bound. This approach can be seen as solving a linear subproblem with one constraint and it can be proven that at least one optimal solution must satisfy this fixing.

In Section 2.2, we have already listed some extensions of the core model. An obvious and widespread extension is to integrate the modelling of convex piecewise linear cost functions depending on some stock level value s. Instead of pointing out a detailed model description we illustrate the idea with a small example. Consider a function $f(s) : [0, 5] \to \mathbb{R}$ defined by

$$f(s) = \begin{cases} -2s + 3, & s \in [0, 1], \\ -s + 2, & s \in [1, 2], \\ \frac{1}{2}s - 1, & s \in [2, 4], \\ 3s - 11, & s \in [4, 5], \end{cases} \tag{5.1}$$

and illustrated in Figure 5.1.

This could be the stock cost for a material with safety stock 2 and storage capacity 4. Between 2 and 4, stock keeping costs are applied, while a stock level larger than the capacity causes large maximum stock violation penalties. On the other hand, a stock level below the safety stock causes safety stock violation penalties. Missing the safety stock by only a bit causes a penalty cost of 1 per unit, while the penalty increases to 2 per unit when the safety stock is missed by more than 1. In the mathematical model, we employ a variable $p_i, i = 1, \ldots, 4$ for

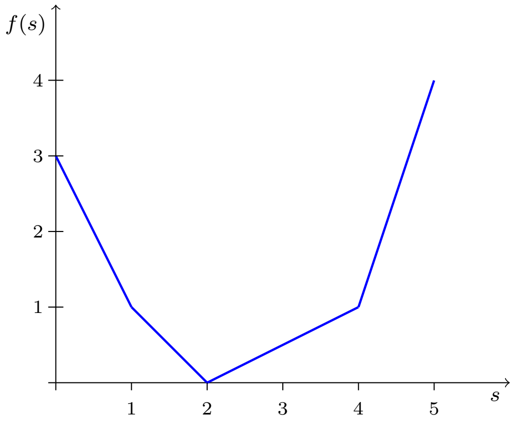

Fig. 5.1. A convex piecewise linear cost function.

each linear segment and obtain the following optimization problem:

$$\begin{aligned} \min \quad & f(s) = 3 - 2p_1 - 1p_2 + \tfrac{1}{2}p_3 + 3p_4 \\ \text{s.t.} \quad & s = p_1 + p_2 + p_3 + p_4 \\ & \qquad s \in [0,5],\ p_1 \in [0,1],\ p_2 \in [0,1],\ p_3 \in [0,2],\ p_4 \in [0,1]. \end{aligned}$$

The slopes in (5.1) correspond to the coefficients in the objective function and the bounds of the variables p_i result from the corresponding interval widths. Due to the increasing objective coefficients, an optimal solution will assign a non-zero value to p_i only if p_j is set to its maximum value for all $j < i$. Singleton column stuffing uses this argument. If, for example, the bounds of s are restricted to $s \in [1.5, 3]$, singleton column stuffing can fix p_1 to 1 and p_4 to 0.

5.2. Dominated columns

The dominated columns presolving algorithm combines two features: implied variable bounds and a dominance relation between two columns of the constraint matrix. Let two variables with the same type, i.e., continuous or integer, be given. Furthermore, let us assume that we want to minimize a linear objective function and all constraints have a less equal relation. Equations can be represented as two inequalities. If the coefficient in the objective function of the first variable is less than or equal to the coefficient of the second variable and if for each row the coefficient of the first variable in that row is less than or equal to that of the second variable, then the first variable dominates the second variable.

A lower or upper bound on a variable derived by bound propagation techniques that is finite and is equal to or tighter than the explicitly stated lower or upper bound, is called implied lower bound or implied upper bound, respectively. Consider two variables, where the first variable dominates the second variable. If the first variable has an implied upper bound, then we can fix the second variable to its lower bound and remove it from the remaining problem. Otherwise, if the second variable has an implied lower bound, then we can set the first variable to the corresponding upper bound and remove this variable from the problem. By simple transformations in the argumentation it is possible to transfer this idea to some other cases.

This algorithm is suited for fixing both continuous and integer variables. Often it removes continuous variables representing a quantity delivered for covering the demand or a variable constituting a specific stock level of a product at a location. Although it is sometimes useful to fix continuous variables, it is more important to reduce the number of discrete variables.

In our context, it particularly helps to remove discrete variables describing the transport of a product along an arc of the supply chain network. If there are different options how a product $p \in P$ can be transported from location $l_1 \in L$ to location $l_2 \in L$, this is typically modeled via parallel arcs. In case no additional restrictions are applied to the transport options, the transport variables $z_{a_1p}(t)$ and $z_{a_2p}(t)$ have the same coefficients in all constraints, but may have different capacities and costs in the time buckets $t \in T$. Now, if presolving identifies an implied upper bound on $z_{a_1p}(t)$ and $\zeta_{a_1p}(t) \leq \zeta_{a_2p}(t)$ applies for the coefficients in the objective function (2.6), $z_{a_2p}(t)$ can be fixed to its lower bound.

5.3. Disconnected components

Although a well-modeled problem should not contain disconnected components, we observe that they occur frequently in our supply chain instances. This happens for example if some region is independent of the others, i.e., has its own, exclusive set of customers without transportation to or from other regions and no common capacity restrictions. In the software environment providing the model, an integrated treatment may be easier to accomplish for the user than setting up independent business cases. More important, we have observed that even a fully connected supply chain problem may split up into independent components after some rounds of presolving. In this case, the MIP solver can employ decomposition techniques that are not applicable at the high-level modeling level explained beforehand.

Mathematically, a disconnected component corresponds to a set of decision variables that do not share a common constraint with any variable from outside the set. Solving the disconnected components individually is equivalent to solving the problem as one piece. The components presolver identifies disconnected components and tries to solve them to optimality. After one component is solved, the constraints and variables therein can be removed from the remaining problem. This approach can strongly speed up LP solving and reduce the total number of branch-and-bound nodes explored in the search trees of the subproblems.

The disconnected components can be identified by first transferring the constraint matrix into an undirected graph which is constructed as follows: For every variable a node is created, and for each constraint we add edges to the graph such that the variables with non-zero coefficients in the constraint are connected. The latter is realized by connecting all nodes of the induced subgraph of a constraint by a simple path. In this case the size of the graph is linear in the number of variables and non-zeros. Finally, we apply depth first search to compute the disconnected components.

Consider the stock level balance equation (2.2) and let us assume that all transportation decision variables $z_{ap}(t)$ for $a \in A^l \cup A_l$ at a specific location $l \in L$ and $p \in P_l$ can be fixed for all time buckets $t \in T$ to certain values determined for example by bound propagation techniques. This gives us a disconnected component for location l, which can be solved individually.

6. Engineering the MIP Solver II: Primal Heuristics

Many supply network planning models in practice, including many that serve as basis of our evaluation in Section 9, tend to be hard to solve to optimality within the given time limit.

If optimality is not reached, practitioners are mainly interested in the value of the best primal solution obtained, while the dual bound is of small interest. In this situation, the use of effective primal heuristics within the MIP solver is essential. They support the branch-and-bound search by trying to construct feasible solutions in a short time. Although they can neither guarantee a certain quality of the generated solution nor the successful construction of any feasible solution at all, they have been proven to be very effective on many problem instances by providing solutions of good quality in a reasonable amount of time. Note that these heuristics work on a general MIP instance and do not use any additional problem information provided by the user. They may, however, identify and exploit certain structures, as it is done in some of the heuristics discussed in the following. The importance of primal heuristics within the exact MIP solver SCIP is also emphasized by the fact that the latest release, SCIP 3.2.1, contains 44 primal heuristics implementing various approaches.

Given this variety of primal heuristics and the need to find good solutions in a short amount of time in the supply chain context we decided to put more emphasis on heuristics by increasing running them more often within the branch-and-bound search and enabling some additional heuristics. More important, however, we developed new general MIP heuristics that are motivated by supply chain problems.

6.1. Shift-and-propagate

Shift-and-propagate [7] is a pre-root heuristic which aims at constructing a feasible solution even before the initial root LP is solved. The heuristic starts with a trivial solution that fulfills variable bounds but potentially violates some constraints. Then it iteratively selects one integer variable and shifts it to a new value such that the number of infeasible constraints (or their violation) is reduced. Domain propagation techniques are applied to deduce bound changes on other variables or detect an infeasibility, in which case the shift is reverted. This is repeated until all integer variables are fixed. Optimal values for the continuous variables are determined by solving a final LP.

One of the reasons why shift-and-propagate performs well on supply chain instances is the fact that feasibility can be reached for many assignments of integer variables, e.g., by paying penalty costs for non-delivery even if production setup variables are fixed to zero. In order to further improve both its success rate and the quality of the constructed solutions, we implemented an extension to the shifting value determination rule motivated by the following observation: For a continuous production variable, shifting the corresponding production setup variable to one allows the LP to run production and decrease non-delivery costs. Since shift-and-propagate regards a modified problem in the shifting phase, this does not necessarily render any additional constraint feasible in that problem. However, it may be needed to obtain feasibility in the final LP solve. Additionally, although production setup may trigger some additional cost, this is often negligible compared to the improvement obtained by reducing the non-delivery cost.

In SCIP's general setting, we were forced to implement this strategy without specific supply chain knowledge. We could achieve this by selecting binary variables that are not restricted from above by any constraints as a generalized proxy for production setup variables. Although this extension can increase the size of the final LP to be solved and the running time of the shift-and-propagate heuristic, it improves the heuristic not only for supply chain instances but also for general MIP problems such that it is now enabled by default in SCIP.

6.2. Structure-based primal heuristics

The restriction to very basic modeling components-variables, linear constraints, and bound constraints-makes it hard to pass problem-specific knowledge to a black-box MIP solver that could be exploited beneficially for the design of dedicated primal heuristics. On the other hand, not having to do so has its own benefits on the business side, as we have argued above. In order to compensate for the limited information, solvers identify and keep some common global structures themselves. We developed new heuristics which make use of these structures for constructing a feasible solution before solving the initial root LP. The latter is important since some of the given instances have very large LP relaxations that take a significant time to solve.

In the following, we will first present a very easy but nonetheless successful heuristic which is based on the bound information of variables. After that, we present two more complex structures—the clique table and the variable bound graph. We discuss their relation to supply chain models and show how they can be used in primal heuristics.

The bound heuristic fixes all binary and integer variables either to their lower or upper bound and solves the remaining LP to obtain optimal values for the continuous variables. Although quite simple, this heuristic can be very effective for supply chain instances: Often, the only integer variables in the problem are indicator variables which allow production or transportation within the network. Running the bound heuristic and fixing all those variables to zero leads to a solution which will deliver stocked products optimally under those restrictions, but not produce any new ones. These restrictions often still allow feasible solutions since non-delivery is possible, though penalized by a cost. As a basis for subsequent improvement heuristics, however, this starting solution can be very effective.

Cliques and variable bounds are two global structures that represent dependencies between variables. A clique is a set of binary variables of which at most one variable can be set to one. A variable bound is a valid bound on a variable which is not yet fixed, but depends on the value (or bound) of another variable. For example, a production variable x typically has a variable upper bound depending on a setup variable y. If y is fixed to zero, the production variable x also has an upper bound of zero; if the upper bound of the setup variable y is one, a non-zero upper bound is imposed on the production variable x. Mathematically, this corresponds to a linear constraint of the general form $x \leq ay + b$.

These structures can be given directly or indirectly by linear constraints of the model or detected by presolving techniques such as probing [8]. In modern MIP solvers, the set of all detected cliques is stored in the so-called clique table, while variable bounds can be stored in the variable bound graph. In this directed graph, each node corresponds to the lower or upper bound of a variable and each variable bound relation is represented by an arc pointing from the influencing bound to the dependent bound.

These structures cover part of the inherent implications of a supply network. For example, cliques may identify conflicting production setup variables, while the variable bound graph depicts how the possible flow in the network is influenced, e.g., by production setup variables. In general, they form relaxations of the MIP and are used by solver components, e.g., to create clique cuts [9], to deduce stronger reductions in presolving and propagation [8], or for c-MIR cut separation, where variable bounds can be used to replace non-binary variables by binary ones [10].

We present now briefly how they can be exploited by primal heuristics, and refer to [11] for more details. The algorithmic framework is the same for both heuristics:

1. In a first step they iteratively fix one or more variables, interleaved with some rounds of domain propagation. This is similar to the behavior of shift-and-propagate, but the decision of which variable to fix and to which value is based on the respective structure. This helps to predict the effects of domain propagation after a fixing.

2. When all variables covered by the structure were fixed, the LP relaxation of the remaining problem is solved and a simple rounding heuristic is applied to compute values for the remaining, unfixed integer variables.

3. If the rounding step fails, the remaining problem is solved as a sub-MIP.

Therefore, the heuristics ultimately implement the large neighborhood search (LNS) paradigm. That means that the problem is restricted to the neighborhood of a given reference point, in this case by fixing of variables. This neighborhood is then solved as a sub-MIP, typically up to some working limits. In contrast to most existing LNS heuristics for MIP, they do not rely on an optimal LP solution or a primal feasible MIP solution as reference point to define the neighborhood. Instead they iteratively define the neighborhood based on the global structures.

In the clique heuristic, the fixing is done based on a clique partition computed in advance, i.e., a set of cliques such that each binary variable is part of exactly one of these cliques. For each clique in the partition, the clique heuristic selects an unfixed variable with smallest objective coefficient and fixes it to one. The subsequent domain propagation step then fixes all other variables in the clique to zero and possibly identifies deductions on variables outside of this clique. The rationale is to fix one variable to one since this causes many domain reductions in propagation and thus reduces the size of the problem to be solved as a sub-MIP. We fix the cheapest variable to one in order to increase the objective value as little as possible.

The variable bound heuristic first computes a topological order for the nodes in the variable bound graph, i.e., an order such that for all arcs in the graph, the origin precedes the destination in that order. The heuristic then regards all nodes of the graph in this order and decides on whether and how to fix the corresponding variable. We developed several fixing schemes that make different trade-offs between feasibility, solution quality, and size of the final sub-MIP.

6.3. Further heuristics

Besides the previously described methods, many general MIP heuristics were implemented during the course of the research project which also helped to improve the performance on the supply chain instances. This includes

- a random rounding heuristic,

- the fast rounding heuristic ZI rounding [12],

- a two-opt heuristic, which tries to improve the best known solution by changing the value of two variables at a time, and

- two more LNS heuristics: zeroobj, which disregards the objective function, allowing for more presolving reductions; and proximity search [13], which searches for an improving solution close to the incumbent.

7. Engineering the MIP Solver III: LP Solving

The fast solution of linear programs is a key ingredient for the performance of SNP Optimization. On the one hand, some practitioners build large supply chain problems with continuous decision variables only. On the other hand, the solution of linear programs is required at many points during the MIP solving process and may easily take up the largest share of the overall solution time. The tight time restrictions demand a highly efficient implementation. This is particularly crucial for pure LPs, where no intermediate solutions are available, and for large instances where most of the available time may be spent in the initial root LP of the branch-and-bound tree.

Particular challenges are the large dimensions of the models as described earlier and the numerically difficult input data as will be discussed in the next section. A typical feature of large supply chain instances is the extreme sparsity of their data, i.e., the abundance of zero elements in the constraint matrix. On average over the regarded instances, the number of nonzeros per constraint is about 7.7 while this number is on average almost 60 for instances in the MIPLIB 2010 [14] benchmark set. Exploiting this characteristic is crucial for any efficient solver [15].

Despite its missing capability to exploit parallel computer hardware, our benchmarks showed the revised simplex method to outperform other LP algorithms such as the barrier method. It is the method of choice in the supply chain context for two further reasons. First, it computes so-called basic solutions, i.e., solutions defined by a set of active constraints, which are often numerically “cleaner” than an interior point solution returned by the barrier without a crossover step. Second, the simplex can hot-start from these basic solutions after small modifications to the LP, the standard scenario inside the branch-and-bound tree of a MIP solver.

The above challenges were addressed in our simplex code SoPlex [16]. We have implemented a technique to combine multiple dual simplex pivots into one iteration by performing so-called bound flips in the step length computation, the so-called ratio test. This technique is known as the long step rule in the simplex literature [17]. Our tests showed, however, that the supply chain instances from our benchmark set exhibit only little potential for performing such long steps. Hence, although this enhancement slightly reduces the number of pivots until optimality, the improvement of the overall performance was small, also because of the additional overhead introduced by computing long steps.

In contrast, we could achieve significant performance improvements by focusing on the pricing step, which selects a variable or constraint that determines the step direction for the next iteration. Different selection criteria are described in the literature and it is known that the steepest edge pricing rule often yields the smallest number of iterations [18]. However, our performance tests showed that on the supply chain instances the computationally more expensive steepest edge pricing was outperformed by the cheaper devex pricing rule because of higher iteration speed.

Independent of the pricing criteria, however, the extreme sparsity of the supply chain instances can be exploited more fundamentally. In the dual simplex algorithm, pricing determines the step direction by selecting a variable or constraint that currently violates one of its bounds and shifting it to its closest bound in order to reduce primal infeasibility. This variable or constraint is selected from a subset called the basis, which contains as many elements as number of constraints in the problem. While a naive implementation might scan the entire simplex basis in every iteration for the largest violations, this is prohibitively expensive for our large

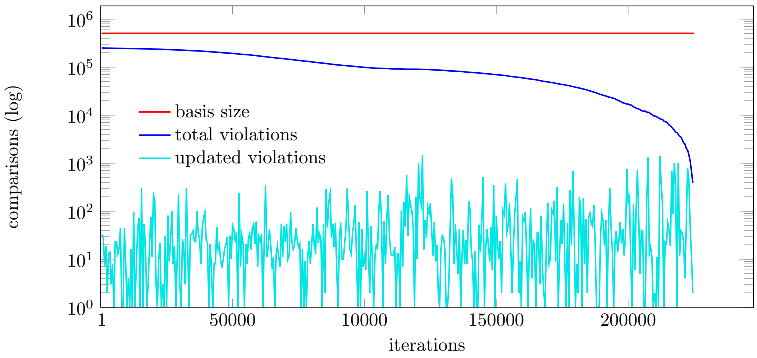

Fig. 7.1. Pricing statistics: The total number of current violations is several orders of magnitude larger than the number of updated violations per iteration.

supply chain models. As a first remedy, we implemented a procedure that keeps track of these violations by updating instead of recomputing them from scratch.

Still, the pricing step can strongly impact the performance because the number of violated variables and constraints is typically very high throughout most of the iterations. On the contrary, we noticed that the majority of violations remain untouched after a single simplex iteration. This is visualized in Figure 7.1, which shows the number of pricing comparisons in all three cases for one typical instance. While the number of violations is only slowly getting smaller, the number of actual changes introduced in a single iteration is several orders of magnitude smaller.

We used this to design an efficient update scheme for keeping a short list of most promising pricing candidates. For the correctness of the simplex algorithm it is sufficient to select one violated variable or constraint. This allowed us to reduce the number of comparisons in pricing significantly during most of the dual iterations. Only in the final iteration we need to ensure that all basic variables and constraints are within their bounds.

8. Engineering the MIP Solver IV: Numerics

Many of the investigated models contain coefficients of very large magnitude both in the constraints as well as in the objective function. Optimal solution values can be in the range of 10^{18}. This imposes difficulties for any software implementation using double-precision arithmetic, which can only express floating-point numbers with up to 16 digits on current architectures. Moreover, at most 12 digits are typically reliable because the last digits are often corrupted due to floating-point round-off errors. During the course of the research project, we robustified SCIP with respect to such extreme numerical scenarios, which required rather technical changes in details of the MIP solving process. In the following, we try to give only a few selected examples.

Per default, SCIP treats all values above 10^{20} as infinity. We changed this limit to 10^{30} in order to prevent cutting off very large numbers. Additionally, we introduced a safety threshold of 10^{15}; larger values are handled with special care, e.g., by deactivating bound tightening on constraints with such large numbers.

Furthermore, activity-based bound tightening relies heavily on so-called minimal and maximal activities, which represent lower and upper bounds on the constraint function. For performance reasons, these activities are usually updated whenever the bound of a variable changes instead of being recomputed from scratch when accessed. However, these updates are prone to numerical errors, in particular if an update reduces their absolute value by several orders of magnitude. Therefore, we added a method checking the reliability of an update; if necessary, the value is marked as unreliable and recomputed from scratch next time it is used. This helped to avoid several cases of incorrect bound tightenings on supply chain instances with very large coefficients.

Our last example is related to presolving. Although the preprocessing phase transforms the model into a mathematically equivalent formulation, it may happen that a solution to this transformed problem is not feasible after mapping it back to the original formulation. This is due to the use of tolerances, usually in the range of 10^{-6} to 10^{-9}, that help to cope with floating-point round-off errors.

Like most MIP solvers, SCIP uses a mixture of relative and absolute tolerances for checking feasibility. While relative tolerances are stable with respect to scaling of a constraint, this does not hold for absolute checks as illustrated by the following example:

$$10^5 x - y = 0 \xrightarrow[\text{by } 10^{-5}]{\text{scaling}} x - 10^{-5} y = 0.$$

The solution $(x^\star, y^\star) = (1, 100000.05)$ is feasible for the scaled equation, since the constraint is only violated by $5 \cdot 10^{-7}$, which is below the tolerance of 10^{-6}. On the other hand, the original equation is violated by 0.05, which is beyond the accepted tolerance.

In order to improve the consistency of our tolerances under scaling, we added the option to consider the violation relative to the largest contribution of a variable to the left-hand side, i.e., solution value of the variable multiplied by its coefficient in the constraint. In the previous example, this check finds the solution $(x^\star, y^\star) = (1, 100000.05)$ feasible also for the original constraint since the check is done relative to 100000.05 which is the largest contribution of a variable.

In the supply chain instances discussed in this paper, this modified check is particularly important for stock level balance equations like (2.2). Their right-hand side is typically zero while the stock levels as well as in- and out-flows can have very high values in millions or billions. An absolute tolerance of 10^{-6} is impractical in those cases and cannot be achieved given the use of floating-point arithmetic.

9. Computational Results

In the literature, a very popular measure for comparing MIP solver performance is the (average) running time to proven optimality. This, however, does not always reflect well the suitability of algorithms in an application setting. Hence, in the following we describe a more practice-oriented benchmarking methodology and evaluate the impact of the algorithmic advances described in Sections 5 to 8 computationally.

9.1. Experimental setup

The basis of our evaluation is a representative selection of 55 benchmark instances supplied by SAP. The instances model diverse real-world supply chain scenarios, each in multiple levels of detail, leading to a variety of problem types and sizes. Basic statistics about the instance sizes are presented in Figure 9.1. The majority of instances are mixed-integer problems (illustrated as blue triangles in Figure 9.1), but there are also 13 mixed-binary problems (green circles)

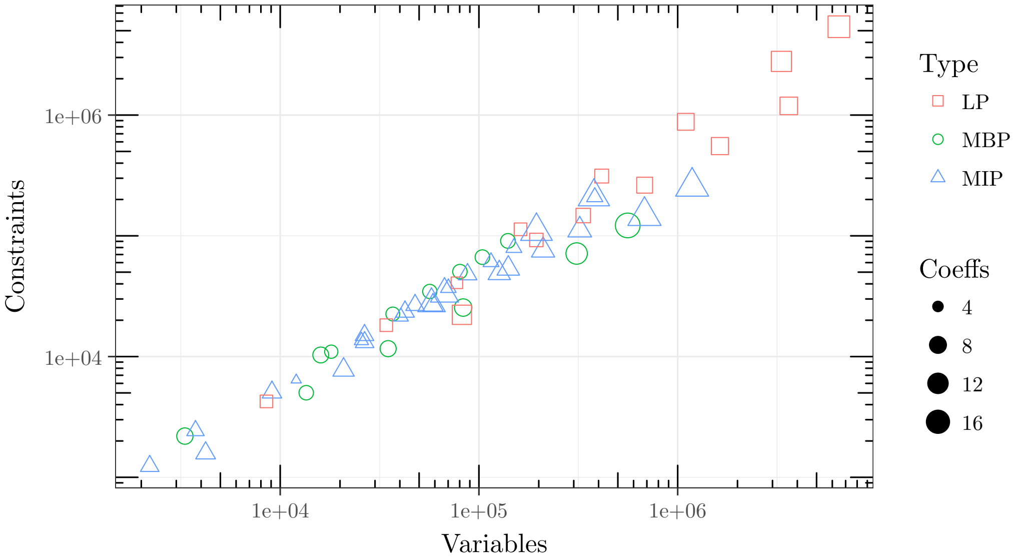

Fig. 9.1. Problem types and sizes of the 55 benchmark instances used for the computational evaluation.

and 14 pure LPs (red squares) without integer or binary variables. The problem sizes in terms of variables (x-axis) and constraints (y-axis) ranges from a few thousand variables and constraints (lower left) to several million variables and constraints (upper right). Thereby, the largest instances are pure LPs, but also the largest MIP has 1.2 million variables and 256 000 constraints. All instances have slightly more variables than constraints; the ratio between the two ranges from 1.2 to 4.7. The average number of non-zero coefficients per constraint is between 3.6 and 18.5, illustrated by the size of the instance symbol in Figure 9.1. The amount of integrality is usually below 10% of the variables but can be as high as 30% for one problem class.

For each benchmark instance, a time limit has been specified within which practitioners expect to obtain a high-quality solution. Because of the challenging nature of the instances and the often tight limits, many benchmark instances cannot be solved to optimality within the given time. Therefore, our foremost measure is the best primal bound computed within this time limit, i.e., the objective function value of the best solution found by SCIP. The given time limits vary between 30 and 7200 seconds, depending mostly on the problem size.

While pure LPs need to be solved to optimality, MIPs have a so-called optimality gap, signifying how far the solution value of an integer feasible solution is away from that of the optimal one. Whenever a new incumbent, i.e., an improved solution is found, the optimality gap is reduced and this development can be represented as a monotonously decreasing function. To measure how quickly the quality of the best known primal solution approaches the optimum, we additionally employ the primal integral measure [19].

For practical applications, this measure is richer than the traditional way of comparing solution times or solution qualities at the limit limit, especially since our main goal is to provide good solutions quickly. To obtain the version-to-version progress displayed in Figure 9.3, we averaged over the primal integrals for all instances in the test set. In order to account for each instance equally, the primal bound function was normalized by the instance-dependent time limits.

All computations were performed on Intel Xeon X5672 CPUs with 3.2 GHz and 64 GB of main memory, running only one problem instance at a time in order to measure solution times accurately. We compared the performance of five different versions of SCIP, version 1.2.0 to version 3.1.0, that span the duration of our research project. SCIP 1.2.0 was released just before the work on the research project started and hence serves as a baseline.

9.2. Evaluation

Figure 9.2 displays the performance progress over the SCIP versions released during the project. To this end, the instances in our test set are classified according to the quality of the primal bound at the end of the instance-specific time limit into the following categories: a provably optimal solution is returned, the final optimality gap is below 1% (near-optimal), below 10% (high-quality), a feasible solution with worse optimality gap is returned, and no solution could be found at all, which is considered a failure.

As can be seen, we managed to improve the solution quality with every new release of SCIP and eventually were able to compute solutions to all but one problem instance. Note that the failing instance is an LP with more than 5 million variables and constraints. This does not mean that the SNP Optimizer is not able to solve these instances at all, only that the user needs to wait for the result longer than the desired time limit. In our evaluation, we defined tight time limits in order to set ambitious goals.

In order to also compare the speed at which improving solutions are found we depict the development of the mean primal gap over all instances, similar as suggested by [19]. In Figure 9.3, we can see a continuous improvement from one SCIP version to the next concerning this measure. Improving solutions were found earlier in the solving process while also the overall primal gap could be decreased significantly. In the following, we shortly present how this progress was achieved.

Both SCIP 1.2.0 and SCIP 2.0.0 were run with default settings. SCIP 1.2.0 was released

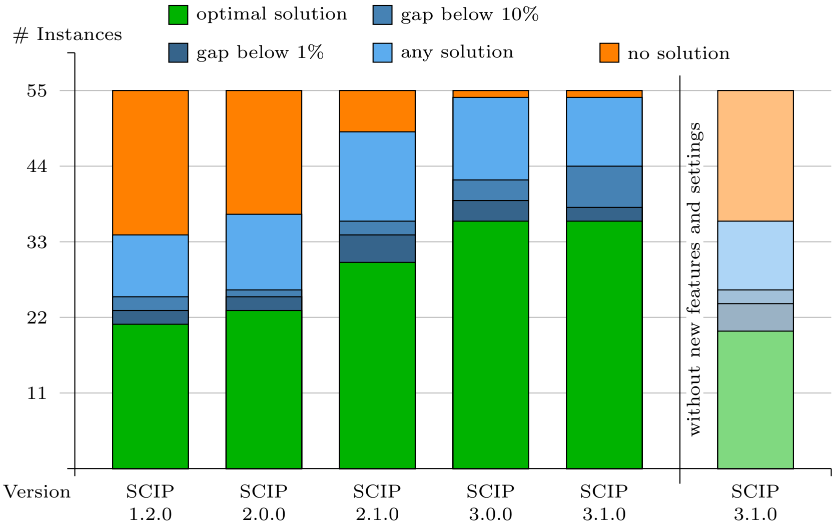

Fig. 9.2. For each SCIP version the solution quality for the instances in the test set is illustrated. The number of found solutions is continuously increasing from version to version. The same holds for the solution quality measured by the integrality gap within the time limit.

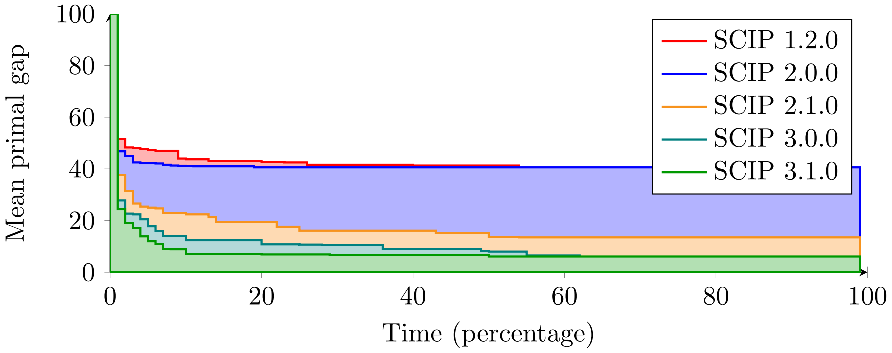

Fig. 9.3. The comparison of the primal integrals using normalized time limits demonstrates that from version to version better solutions are found earlier during the solution process.

before work on the project started, whereas SCIP 2.0 was released shortly after. The most important new feature in SCIP 2.0.0 is the shift-and-propagate heuristic, which computes feasible solutions very early in the solving process. It helped to solve more instances to optimality and reduce the number of failed instances.

SCIP 2.1.0 led to a considerable performance improvement, partly due to using customized SCIP parameters. On the one hand, algorithmic features which proved to be inefficient for the regarded supply chain instances were disabled or reduced in their intensity, most prominently the probing presolver, which was able to find only few reductions while consuming a significant amount of time due to the large size of many of the problems. More importantly, however, the aggressiveness of the available heuristics was increased in order to focus on finding solutions rather than proving optimality. Additionally, first versions of the structure-based heuristics were added, which saw further enhancements in the following releases. This was complemented by many more performance improvements in SCIP, among them improved and extended algorithms for presolving and node preprocessing. The corresponding SoPlex release contained for the first time the long step ratio test and modifications for numerical stability: LP scaling prior to running the simplex algorithm and an increased Markowitz threshold, that is used for the internal LU factorizations.

The three presolvers described above were added in release 3.0.0 of SCIP. Among them, the detection of independent components in the problem instance during presolving had the highest impact, often leading to significantly reduced problem sizes. Numerical problems were reduced by the addition of the three modifications presented in the last section. Furthermore, some parameters were modified to decrease the number of useless cutting planes and to switch the default LP pricing rule from steepest edge to the faster devex pricing. Additionally, the overall pricing implementation in SoPlex was improved to exploit sparsity of the data.

The work on SCIP 3.1.0 mainly succeeded in improving performance on the primal side. The random rounding heuristic was added to SCIP and other existing heuristics were improved, most importantly the shift-and-propagate heuristic as described above. Moreover, LP pricing was accelerated by adding hyper sparse functionality.

Finally, in order to distinguish the impact of the work presented in Sections 5–8 from the general speedup achieved by the four releases of SCIP during the duration of our reseach project, we also performed a run with SCIP 3.1 where we disabled all new algorithms and reset all customized parameters to their default value. The solution quality on the instances, illustrated by the rightmost bar in Figure 9.2, lies between the quality obtained for SCIP 1.2 and SCIP 2.0. This demonstrates that the majority of the improvement in SCIP for supply chain instances stems from the developments described in this paper.

10. Conclusion

Modeling flexibility and optimality guarantees are only two benefits of mixed-integer programming that make it an ideal basis for building a robust and feature-rich decision support system. In this paper, we discussed the algorithmic challenges of the MIP approach in the context of supply chain planning and showed how it can be applied successfully even to a large and diverse user base such as that of SAP's SNP Optimizer.

We tried to emphasize that this result could only be achieved by algorithmic improvements on many different levels, both external and internal to the MIP solver. These techniques exploit supply-chain-specific structures without tying themselves to specific models. Notably, some of these features are even better exploited inside the general MIP solver, which may seem counterintuitive at first. To give only one simple example, we observed that different components of the model may become disconnected only during the preprocessing phase of the solver after it has identified fixed linking variables or redundant linking constraints through a large collection of mathematical reduction techniques.

As a result, we achieved drastic performance improvements within the academic MIP solver SCIP for supply chain planning problems. We demonstrated this in particular with respect to the quality of the primal solutions obtained within ambitious time limits specified according to instance size and user requirements.

Acknowledgements. We wish to sincerely thank SAP for their long-term financial and personal support. In addition, this work has been supported by the Research Campus Modal Mathematical Optimization and Data Analysis Laboratories funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). Furthermore, we acknowledge funding through the DFG SFB/Transregio 154. All responsibility for the content of this publication is assumed by the authors.

References

[1] H. Stadtler, B. Fleischmann, M. Grunow, H. Meyr and C. Sürie, Advanced Planning in Supply Chains, Management for Professionals, Springer-Verlag Berlin Heidelberg, 2012.

[2] T. Achterberg, SCIP: Solving Constraint Integer Programs Mathematical Programming Computation, **1**:1 (2009), 1–41.

[3] E. Klotz, Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs, A. Newman and J. Leung, editors, Bridging Data and Decisions, TutORials in Operations Research, pages 54–108, 2014.

[4] R. Bixby and E. Rothberg, Progress in computational mixed integer programming–a look back from the other side of the tipping point, *Annals of Operations Research*, **149** (2007), 37–41.

[5] T. Achterberg and R. Wunderling, Mixed integer programming: Analyzing 12 years of progress, M. Jünger and G. Reinelt, editors, Facets of Combinatorial Optimization, pages 449–481, Springer Berlin Heidelberg, 2013.

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

[7] T. Berthold and G. Hendel, Shift-and-propagate, *Journal of Heuristics*, **21**:1 (2014), 73–106.

[8] M.W.P. Savelsbergh, Preprocessing and probing techniques for mixed integer programming problems, *ORSA Journal on Computing*, **6** (1994), 445–454.

[9] E.L. Johnson and M.W. Padberg, Degree-two inequalities, clique facets, and biperfect graphs, *North-Holland Mathematics Studies*, **66** (1982), 169–187.

[10] H. Marchand and L.A. Wolsey, Aggregation and mixed integer rounding to solve MIPs, *Operations Research*, **49**:3 (2001), 363–371.

[11] G. Gamrath, T. Berthold, S. Heinz and M. Winkler, Structure-based primal heuristics for mixed integer programming, Optimization in the Real World, volume 13, pages 37–53, 2015.

[12] C. Wallace, ZI round, a MIP rounding heuristic, *Journal of Heuristics*, **16**:5 (2010), 715–722.

[13] M. Fischetti and M. Monaci, Proximity search for 0-1 mixed-integer convex programming, *Journal of Heuristics*, **20**:6 (2014), 709–731.

[14] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy and K. Wolter, MIPLIB 2010, *Mathematical Programming Computation*, **3**:2 (2011), 103–163.

[15] J.A.J. Hall and K.I.M. McKinnon, Hyper-sparsity in the revised simplex method and how to exploit it, *Comp. Opt. and Appl.*, **32**:3 (2005), 259–283.

[16] R. Wunderling, Paralleler und objektorientierter Simplex-Algorithmus, PhD thesis, Technische Universität Berlin, 1996.

[17] E. Kostina, The long step rule in the bounded-variable dual simplex method: Numerical experiments, *Mathematical Methods of Operations Research*, **55**:3 (2002), 413–429.

[18] J.J. Forrest and D. Goldfarb, Steepest-edge simplex algorithms for linear programming, *Mathematical Programming*, **57**:1 (1992), 341–374.

[19] T. Berthold, Measuring the impact of primal heuristics, *Operations Research Letters*, **41**:6 (2013), 611–614.