Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

GERALD GAMRATH, AMBROS GLEIXNER, THORSTEN KOCH, MATTHIAS MILTENBERGER, DIMITRI KNIASEW, DOMINIK SCHLÖGEL, ALEXANDER MARTIN, AND DIETER WENINGER

Tackling Industrial-Scale Supply Chain Problems by Mixed-Integer Programming

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 responsibilty for the content of this publication is assumed by the authors. Last not least, we want to thank SAP for their long-term financial and personal support.

ZIB Report 16-45 (November 2016)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Tackling Industrial-Scale Supply Chain Problems by Mixed-Integer Programming

Gerald Gamrath, Ambros Gleixner, Thorsten Koch,
and Matthias Miltenberger

Zuse Institute Berlin, Department Optimization,
{gamrath,gleixner,koch,miltenberger}@zib.de

Dimitri Kniasew and Dominik Schlögel

SAP SE, {dimitri.kniasew,dominik.schloegel}@sap.com

Alexander Martin and Dieter Weninger

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Mathematics,
{alexander.martin,dieter.weninger}@math.uni-erlangen.de

November 24, 2016

Abstract

SAP's decision support systems for optimized supply network planning rely on mixed-integer programming as the core engine to compute optimal or near-optimal solutions. The modeling flexibility and the optimality guarantees provided by mixed-integer programming greatly aid the design of a robust and future-proof decision support system for a large and diverse customer base. In this paper we describe our coordinated efforts to ensure that the performance of the underlying solution algorithms matches the complexity of the large supply chain problems and tight time limits encountered in practice.

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 (Stadtler et al. 2012, Chapter 1).

Following this trend, SAP, one of the leading vendors of business software, started to develop and sell new software products such as the SAP® Advanced Planning and Optimization component[1]. It provided various functions to solve some typical decision problems arising within supply chains of globally operating

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

companies. Some of these functions require sophisticated algorithms and use complex mathematical techniques. SAP Advanced Planning and Optimization was released to the market in 1998.

This article focuses on a specific function of SAP Advanced Planning and Optimization called Supply Network Planning Optimization (SNP Optimization), which relies on *mixed-integer linear programming* (MIP) models and solvers. Today it is part of the SAP Supply Chain Management (SAP SCM) application and was partially retrofitted into a newer solution, SAP Integrated Business Planning, and is widely used by many customers[2].

1.1 Academic-industrial synergies

Despite the many benefits of employing general MIP solver software, this approach also comes with significant challenges when applied to real-world optimization problems. These challenges motivated SAP in 2010 to start an academic-industrial research cooperation with developers of the academic MIP solver SCIP[3] from Zuse Institute Berlin (ZIB) and Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). The primary goal of this ongoing cooperation is to investigate how to improve MIP solving performance under the focus on the SNP-specific model structures, while at the same time maintaining the benefits of the generic MIP approach.

From the academic perspective, this cooperation helps tremendously to exhibit current bottlenecks of MIP solvers that may remain hidden when testing on publicly available benchmark sets. To create a common basis for investigation and performance improvement, SAP provided a benchmark set of small, medium, and large MIP instances derived from various SNP real-world scenarios.

1.2 Purpose and outline of the article

In this article we describe the results that have been achieved through this cooperation regarding the solution of large-scale real-world supply chain problems and the implementation of these algorithms in a software product delivered to SAP customers 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 needed to be developed inside a generic MIP solver to improve computational performance on MIP formulations with supply chain structure.

The article is organized as follows: First, we present the type of MIP models solved and discuss the benefits and challenges of the MIP approach. Second, 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

[2]For a detailed product description see http://help.sap.com/saphelp_scm700_ehp03/helpdata/en/81/3ec95360267614e10000000a174cb4/content.htm?frameset=/en/bb/40c95360267614e10000000a174cb4/frameset.htm and http://help.sap.com/saphelp_ibp62/helpdata/en/34/234454dafe8b24e10000000a4450e5/frameset.htm

[3]See http://scip.zib.de/.

within the running time requirements imposed by customers. Third, we present algorithmic innovations inside the MIP solver SCIP that have been developed in this cooperation. Finally, we demonstrate their performance impact over a set of supply chain benchmark problems that were used to evaluate progress within the cooperation. The appendix provides mathematical details of the underlying MIP formulations and our computational methodology.

2 The SNP Optimization model

The goal of SNP Optimization is to provide quantitative decision support for planners by suggesting medium- or long-term plans for typical supply chain processes such as procurement, production, transportation, and customer-demand fulfillment. 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
- Transportation facilities

see Stadtler et al. (2012) for details. SNP Optimization tries 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. In other words, SNP Optimization decides how many of the products to produce, store, transport between the locations, and so on. Some quantities may be produced and transported only in discrete lots.

The granularity of these quantities is determined according to the coarse temporal structure of the planning interval (*horizon*), which is subdivided into periods, called *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 customers 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.

Besides network flow type constraints in order to model the transportation of material through the supply network, a feasible supply chain plan must satisfy a variety of business-specific constraints. A detailed description of the basic MIP model employed in the SNP Optimization package is given in the appendix.

3 Benefits and challenges of the MIP approach

To find a feasible or even optimal solution for the supply chain problem, SNP Optimization maps the business-related data from the SAP system into a mathematical model consisting of variables and linear equations and inequalities. Here, the variables represent the planning decisions per bucket, while the constraints represent business-specific rules or restrictions such as stock-level balance or resource capacity limits. Moreover, a part of the variables may be integral, since supply chains often require some production or transport to be planned in discrete lots. In addition to the constraint set, the model also contains an objective function representing the overall costs that have to be minimized. Some customers create remarkably large scenarios with SNP Optimization, resulting in MIP models with up to 30 million variables and constraints, where up to 500,000 of the variables can be integer.

After the SNP Optimization has built the mathematical model, it invokes an MIP solver to minimize it. The result is then converted into a supply chain plan proposed to the customer.

3.1 MIP vs. heuristics

There were three main reasons why SAP decided, from the very beginning of the SNP Optimization development, to embrace the MIP approach instead of heuristic algorithms:

- Feature complexity of real-world supply chain problems
- Extensibility of MIP models
- Possibility to evaluate feasibility and solution quality

In the face of the feature complexity of the supply chain problems to be solved by SNP Optimization, one strong argument for MIP was the expressiveness of general MIP models. Besides multiple stages in production, capacity constraints, and discrete lot sizes, many SNP-specific hard or soft restrictions may occur in an SNP Optimization problem. 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 SNP-specific aspects is neither available on the software market nor described in literature. However, for the SAP customers that use SNP Optimization, all planning features may be relevant. It was SAP's objective to provide a single software solution for all these business requirements.

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 customer 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 (Stadtler et al. 2012). 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. Since the SNP Optimization model is directly derived from the customer'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 Klotz (2014)).

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 customers 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 the customers is typically limited to approximately four hours when performing the SNP Optimization in a batch processing mode. When running the tool manually on smaller scenarios, customers usually expect a solving time of few minutes.

4 Decomposition techniques

One 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 cooperation, decomposition is applied both inside the MIP solver as well as on the SNP Optimization side 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., disjointed groups of variables and the constraints linking them together. 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 customers often prefer decomposition, in particular when planning large supply chains. By offering these decomposition approaches in conjunction with an exact model, SNP Optimization enables the customer 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. 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 often breaks 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 of the customer problems. A more complex way is to assign time slots to the subproblems that are proportional to their estimated complexity. 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 the 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 (Bixby and Rothberg 2007, Achterberg and Wunderling 2013).

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 briefly describe three presolving techniques suitable for improving the MIP formulation of real-world supply chain instances. Further mathematical details can be found in Gamrath et al. (2015b)

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, or 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 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.

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. 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 the 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 material 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 which describe the transport of "suboptimal" materials along an arc of the supply chain network.

Disconnected components

Although a well-modeled problem should not contain disconnected components, we observe that they occur frequently in our supply chain instances. This hap-

pens 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.

6 Engineering the MIP solver II: primal heuristics

The supply network planning models investigated in our cooperation tend to be hard to solve to optimality within the given time limit. If optimality is not reached, customers 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.

Shift-and-propagate

Shift-and-propagate (Berthold and Hendel 2014) 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 startup 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 startup 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 startup 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 startup 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.

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 nondelivery 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 startup 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 startup 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 (Savelsbergh 1994). 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 startup variables, while the variable bound graph depicts how the possible flow in the network is influenced, e.g., by production startup variables. In general, they form relaxations of the MIP and are used by solver components, e.g., to create clique cuts (Johnson and Padberg 1982), to deduce stronger reductions in presolving and propagation (Savelsbergh 1994), or for c-MIR cut separation, where variable bounds can be used to replace non-binary variables by binary ones (Marchand and Wolsey 2001).

We present now briefly how they can be exploited by primal heuristics, and refer to Gamrath et al. (2015a) 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 propa-

gation 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.

Other heuristics

Besides the previously described methods, many general MIP heuristics were implemented in the course of the 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* (Wallace 2010),
- 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* Fischetti and Monaci (2014), 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 customers 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 non-zeros per constraint is about 7.7 while this number is on average almost 60 for instances in the MIPLIB 2010 (Koch et al. 2011) benchmark set. Exploiting this characteristic is crucial for any efficient solver (Hall and McKinnon 2005).

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.

Within our cooperation, we addressed the above challenges in our simplex code SoPlex (Wunderling 1996). 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 (Kostina 2002). 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 (Forrest and Goldfarb 1992). 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 algo-

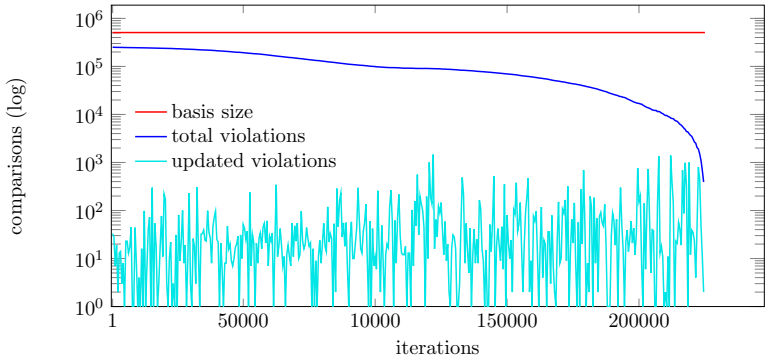

Figure 1: Pricing statistics: The total number of current violations is several orders of magnitude larger than the number of updated violations per iteration.

rithm, 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 naïve implementation might scan the entire simplex basis in every iteration for the largest violations, this is prohibitively expensive for our large 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 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 cooperation, 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:

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

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 constraints. 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 Quantifying progress

In the literature, a very popular measure for comparing MIP solver performance is the (average) running time to proven optimality. In our project, however, we apply

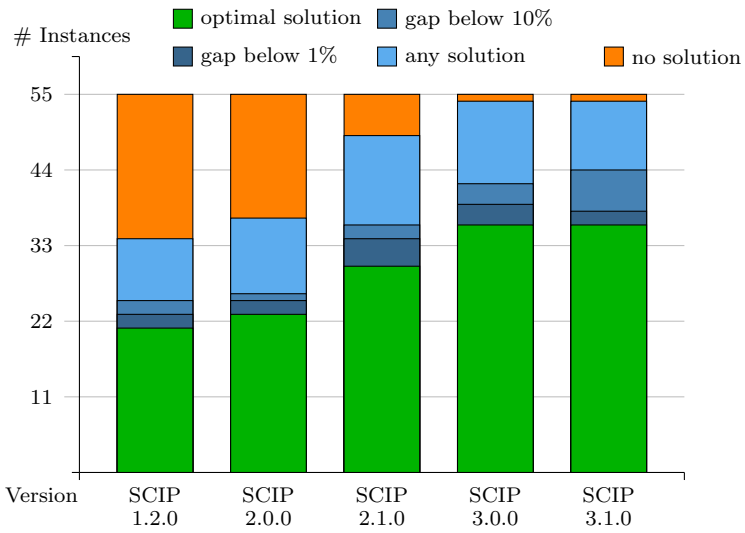

Figure 2: For each SCIP version the solution quality for the instances in the test set is illustrated. The number of found solutions is continuously increasing from version to version. The same holds for the solution quality measured by the integrality gap within the time limit.

a more customer-oriented measure. For each benchmark instance, there is a time limit within which the customer expects 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.

Figure 2 classifies the instances in our test set 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 this instances at all, only that the customer needs to wait for the result longer than the desired time limit. In our cooperation, 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 Berthold (2013). In Figure 3, we can see a continuous improvement from one version to the next concerning this measure. Over the course of the project, 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 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 described 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 before. Moreover, LP pricing was accelerated by adding hyper sparse functionality.

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 that can be deployed to a large and diverse customer base such as SAP's *SNP Optimizer*. Clearly, the performance of the underlying MIP solver is key to the success of this approach and can become critical when applied to real-world problems of industrial scale. The high degree of problem-specific knowledge in supply network modeling and the complexity of today's state-of-the-art MIP solvers require a diverse skill-set to tackle these challenges. This motivated the practitioners from SAP and the developers of the academic MIP solver *SCIP* to join forces.

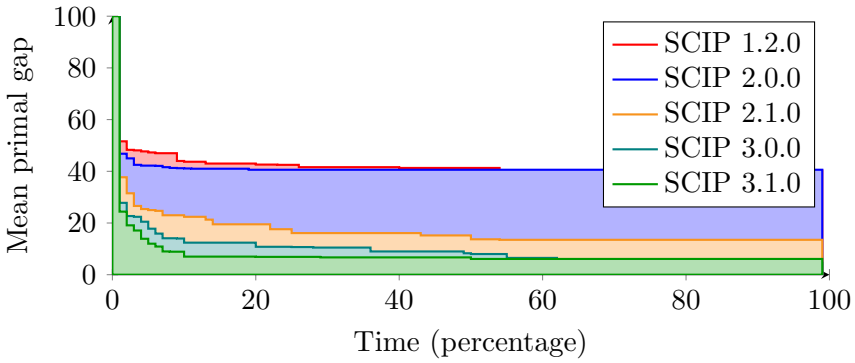

Figure 3: The comparison of the primal integrals using normalized time limits demonstrates that from version to version better solutions are found earlier during the solution process.

In this article, we tried to show that the successes of our cooperation could only be achieved by algorithmic improvements on many different levels, both external and internal to the MIP solver. We managed to exploit supply-chain-specific structures without tying ourselves to specific models. And we learned that some features are even better exploited inside the general MIP solver, which may seem counter-intuitive 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 improvements of the SCIP performance 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.

Acknowledgments. We want to 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 responsibilty for the content of this publication is assumed by the authors.

References

Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-38189-8_18.

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611 – 614, 2013. doi: 10.1016/j.orl.2013.08.007.

Timo Berthold and Gregor Hendel. Shift-and-propagate. *Journal of Heuristics*, 21(1):73 – 106, 2014. doi: 10.1007/s10732-014-9271-0.

Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming–a look back from the other side of the tipping point. *Annals of Operations Research*, 149:37–41, 2007. doi: 10.1007/s10479-006-0091-y. URL http://dx.doi.org/10.1007/s10479-006-0091-y.

Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014. doi: 10.1007/s10732-014-9266-x.

John J. Forrest and Donald Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57(1):341–374, 1992. doi: 10.1007/BF01581089. URL http://dx.doi.org/10.1007/BF01581089.

Gerald Gamrath, Timo Berthold, Stefan Heinz, and Michael Winkler. Structure-based primal heuristics for mixed integer programming. In *Optimization in the Real World*, volume 13, pages 37 – 53. 2015a. ISBN 978-4-431-55419-6. doi: 10.1007/978-4-431-55420-2_3.

Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367 – 398, 2015b. doi: 10.1007/s12532-015-0083-5.

J. A. Julian Hall and Ken I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Comp. Opt. and Appl.*, 32(3):259–283, 2005. doi: 10.1007/s10589-005-4802-0. URL http://dx.doi.org/10.1007/s10589-005-4802-0.

Ellis L Johnson and Manfred W Padberg. Degree-two inequalities, clique facets, and biperfect graphs. *North-Holland Mathematics Studies*, 66:169–187, 1982.

Ed Klotz. Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs. In Alexandra Newman and Janny Leung, editors, *Bridging Data and Decisions*, TutORials in Operations Research, pages 54–108. 2014. doi: 10.1287/educ.2014.0130.

Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi: 10.1007/s12532-011-0025-9. URL http://mpc.zib.de/index.php/MPC/article/view/56/28.

Ekaterina Kostina. The long step rule in the bounded-variable dual simplex method: Numerical experiments. *Mathematical Methods of Operations Research*, 55(3):413–429, 2002. doi: 10.1007/s001860200188. URL http://dx.doi.org/10.1007/s001860200188.

Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001. doi: 10.1287/opre.49.3.363.11211.

Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

Hartmut Stadtler, Bernhard Fleischmann, Martin Grunow, Herbert Meyr, and Christopher Sürie. *Advanced Planning in Supply Chains*. Management for Professionals. Springer-Verlag Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-24215-1.

Chris Wallace. ZI round, a MIP rounding heuristic. *Journal of Heuristics*, 16(5):715–722, 2010. doi: 10.1007/s10732-009-9114-6. URL http://dx.doi.org/10.1007/s10732-009-9114-6.

Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.

Appendix: Mathematical Details

This appendix presents details about the mathematical model used in SNP Optimization and about our benchmarking methodology.

Mixed-integer linear programming models

To give a better insight on the structure and the connection between business and mathematical programming a basic supply chain model is elaborated here. Every linear programming model consists of the two fundamental parts: the objective function, subject of minimization or maximization, and the constraint matrix, consisting of different inequalities and restricting the solution space. In the case of cost based supply chain models, all arising costs—like transport costs, production costs, and most importantly non-delivery costs—constitute the objective function, which then needs to be minimized. Costs usually are connected to variables that need to be calculated by the solver. These are exactly the variables the customer is interested in: how many products need to be transported on which transportation lane or how many products need to be produced in this special factory.

The supply chain structure itself yields numerous constraints. These emerge from restrictions entered by the customer like limited resource capacities but also from essential requirements like stock balance consistency. Step by step we construct all relevant constraints needed to formulate the stock balance equation on the one hand and an opposed type of constraint resulting form the capacity resource restriction on the other. Keep in mind that this is only a rather simple exemplary model, without any claims of completeness.

Definition of sets

T	set of time buckets
L	set of locations
D	set of demands
D_t	set of demands in time bucket t
D_l	set of demands at location l
A	set of arcs
A^l	set of arcs with location l as destination
A_l	set of arcs with location l as origin
P	set of products
PL_l	set of products that are handled at location l
PA_a	set of products that can be transported on arc a
O_l	set of production Models at location l
R	set of resources

Demands

The main goal of supply chain management is to satisfy requested demands. However, there are no cost-relevant benefits or rewards created by satisfying a demand. Instead, in order to initiate activity within the supply chain despite the costs of

production and transportation, the model incorporates artificial non-delivery penalties.

Definition of parameters:

d_t	demand d in bucket t
DQ_{d_t}	quantity of demand d in bucket t
δ_{d_t}	maximum allowed lateness for demand d in bucket t
CN_{d_t}	non-delivery costs for demand d in bucket t
$CL_{d_t}(t')$	late delivery costs delivering $(t' - t)$ buckets late

Definition of variables:

VN_{d_t}	quantity not delivered for demand d in bucket t
$VL_{d_t}(t')$	quantity delivered in bucket t' for demand d in bucket t

A demand may allow a late delivery by some time buckets. To obtain the non-delivered amount regarding one demand in a bucket all deliveries for that demand, even the delayed ones, are subtracted from the original demand quantity in that bucket:

$$VN_{d_t} = DQ_{d_t} - \sum_{t'=t}^{t+\delta_t} VL_{d_t}(t')$$

In addition to the constraints there are costs that need to be respected. All cost equations will be displayed for one single time bucket only. To obtain total costs all relevant time buckets need to be summed up. Non-delivery costs are calculated by:

$$\sum_{d_t \in D_t} CN_{d_t} \cdot VN_{d_t}$$

Where late delivery is allowed, additional costs need to be incurred to motivate on-time delivery. These late delivery costs are calculated by:

$$\sum_{d_t \in D_t} \sum_{t'=t+1}^{t+\delta_{d_t}} CL_{d_t}(t') \cdot VL_{d_t}(t')$$

Transport

To make transfer of products possible, transport lines between locations, here called arcs, need to be modeled.

Definition of parameters:

$CT_{ap}(t)$	cost of transporting one unit of product p via arc a in bucket t

Definition of variables:

$VT_{ap}(t)$	quantity of product p transported on arc a in bucket t

For simplicity, we will not consider any restrictions on these arcs. But there may still arise transport costs when shipping products form one location to another:

$$\sum_{a \in A} \sum_{p \in PA_a} CT_{ap}(t) \cdot VT_{ap}(t)$$

Production

Another basic activity of the supply chain is the production of a product by consuming other products. This is modeled by production models, containing all information on input and output products.

Definition of parameters:
$CO_o(t)$ cost of applying production model $o \in O_l$ once at location l in bucket t

Definition of variables:
$VO_o(t)$ continuous number of applications of production model $o \in O_l$ at location l in bucket t

Similarly to transports, we will neither consider limitations nor possible coefficients for input or output. Production costs are computed by:

$$\sum_{l \in L} \sum_{o \in O_l} CO_o(t) \cdot VO_o(t)$$

Procurement

Procurement models external acquisitions of products and makes them available at specific locations.

Definition of variables:
$CP_{lp}(t)$ cost of procuring product p at location l in bucket t

Definition of variables:
$VP_{lp}(t)$ quantity procured of product p at location l in bucket t

Procurement costs may be incurred, especially when weighting external sourcing against internal production. The costs are calculated by:

$$\sum_{l \in L} \sum_{p \in PL_l} CP_{lp}(t) \cdot VP_{lp}(t)$$

Stock level

Products stored at locations and existing stock from previous time buckets need to be accounted. But there can be further restrictions, for instance on stock quantity.

Definition of parameters:
$PR^+_{op}(t)$ output quantity of product p from production model o in bucket t
$PR^-_{op}(t)$ input quantity of product p to production model o in bucket t

Definition of variables:
$VS_{lp}(t)$ stock level at location l of product p in bucket t
$CS_{lp}(t)$ cost of storing product p at location l in bucket t

For products kept in stock at a location so called inventory holding costs can be incurred:

$$\sum_{l \in L} \sum_{p \in PL_l} CS_{lp}(t) \cdot VS_{lp}(t)$$

We finally have introduced all relevant variables and equations to define stock balance equations, which contain 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:

$$VS_{lp}(t) = VS_{lp}(t-1) + VP_{lp}(t) + \sum_{o \in O_l} PR^{+}_{op}(t) \cdot VO_o(t) + \sum_{a \in A^l} VT_{ap}(t)$$
$$- \sum_{o \in O_l} PR^{-}_{op}(t) \cdot VO_o(t) - \sum_{a \in A_l} VT_{ap}(t) - \sum_{d_\tau \in D_l} VL_{d_\tau p}(t)$$

Resource capacities

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.

Definition of parameters:

$RR_{or}(t)$	requirement of resource r for production model o in bucket t
$RC_r(t)$	capacity of resource r in bucket t

There are no costs for resource consumption. Production resource capacity restrictions are mapped to the constraints:

$$\sum_{o \in O_l} RR_{or}(t) \cdot VO_o(t) \leq RC_r(t)$$

Production with minimum lot sizes

Assuming all variables to this point were continuous, the above model describes a linear program. Only when modeling minimum lot sizes or fixed lot sizes, we obtain a mixed-integer program.
The minimum lot size for production describes a minimum quantity of a product that needs to be produced in case production takes place. To model this restriction, a binary decision variable needs to be introduced, defining if production occurs in a specific time bucket. Further a new variable representing the minimum lot size itself is defined.

Definition of parameters:

$VMIN\_VO_o(t)$	quantity of minimum lot size
M	large constant used in big-M method

Definition of variables:

$BMIN\_VO_o(t)$	binary indicator variable for minimum lot sizes

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.

$$VO_o(t) - M \cdot BMIN\_VO_o(t) \leq 0$$
$$VMIN\_VO_o(t) \cdot BMIN\_VO_o(t) - VO_o(t) \leq 0$$

Production with fixed lot sizes

Fixed lot sizes for production represent the requirement of producing only multiples of a fixed quantity. To model fixed lot sizes for production the according variable for the number of application of the production model needs to be restricted to be integral. Furthermore we introduce a new variable to determine the lot size itself.

Definition of parameters:

$VLOT\_VO_o(t)$	lot size for production quantities of product p on production model $o \in O_l$ at location l in bucket t

Definition of variables:

$VO_o(t)$	integral number of applications of production model $o \in O_l$ at location l in bucket t

When the production variable VO is continuous, the lot size factor will have no influence on the solution, it only needs to be considered when reconverting the mathematical solution into the business model. Therefore we can combine the equations of production with and without lot sizes, as they differ only in the integrity of the production variable VO. So instead of a new cost function, the previous production cost function will change to:

$$\sum_{l \in L} \sum_{o \in O_l} CO_o(t) \cdot VLOT\_VO_o(t) \cdot VO_o(t)$$

Also the constraints for the stock level equations have to be adapted accordingly:

$$VS_{lp}(t) = VS_{lp}(t-1) + VP_{lp}(t) + \sum_{o \in O_l} PR^{+}_{op}(t) \cdot VLOT\_VO_o(t) \cdot VO_o(t) + \sum_{a \in A^l} VT_{ap}(t)$$
$$- \sum_{o \in O_l} PR^{-}_{op}(t) \cdot VLOT\_VO_o(t) \cdot VO_o(t) - \sum_{a \in A_l} VT_{ap}(t) - \sum_{d_\tau \in D_l} VL_{d_\tau p}(t)$$

In the same way minimum and fixed lot sizes for transport activities can be introduced.

Objective function

While the constraint matrix of this exemplary model consists of the combination of all constraints, we obtain the objective function by summing all cost functions we introduced so far:

$$
\begin{aligned}
\text{ObjFunct} = \sum_{d_\tau \in D} \left(\sum_{t' > \tau}^{\tau + \delta_{d_\tau}} CL_{d_\tau}(t') \cdot VL_{d_\tau}(t') + CN_{d_\tau} \cdot VN_{d_\tau} \right) \\
+ \sum_{t \in T} \Bigg(\sum_{a \in A} \sum_{p \in PA_a} CT_{ap}(t) \cdot VT_{ap}(t) \\
+ \sum_{l \in L} \sum_{o \in O_l} CO_o(t) \cdot VLOT\_VO_o(t) \cdot VO_o(t) \\
+ \sum_{l \in L} \sum_{p \in PL_l} \Big(CP_{lp}(t) \cdot VP_{lp}(t) + CS_{lp}(t) \cdot VS_{lp}(t) \Big) \Bigg)
\end{aligned}
$$

This objective function will then be subject of minimization.

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:

- 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
- 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
- fix material flows: some production may incur fix material flows independent of the number of produced lots
- fix resource consumption: some production may incur fix resource capacity consumption, independent of the number of produced lots

Benchmarking methodology

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.

The used test set consists of a representative subset of all training instances supplied by SAP within our cooperation. They model diverse real-world supply

chain scenarios, each in multiple levels of detail, leading to a variety of problem sizes. The given time limits vary between 30 and 7200 seconds, depending mostly on the problem size. Some instances are pure LPs, i.e., there are no integer or binary variables, whereas the majority of instances are either mixed-integer or mixed-binary problems. The amount of integrality is usually below 10% of the variables but can be as high as 30% for one problem class.

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 solutions approaches the optimum, one can employ the primal integral (Berthold 2013).

For practical applications, this measure is richer than the traditional way of comparing solution times or solution qualities at the limit limit, especially in the described setting of this project, where our main goal is to provide good solutions quickly. To obtain the version-to-version progress displayed in Figure 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.