MIP Presolve Techniques for a PDE-based Supply Chain Model

A. Dittel* A. Fügenschuh* S. Göttlich† M. Herty‡

April 24, 2008

Abstract

We consider a mixed-integer linear program (MIP) for supply chains which has been derived in [16] from a continuous supply chain model based on partial differential equations (PDE). We develop new presolve techniques where knowledge about the continuous framework is involved. For this purpose, several presolve levels are introduced and compared numerically. The presented methods reduce the size of the MIP in terms of number of variables and constraints, accelerate the solution process of the MIP when using numerical solvers, and finally assure that such solvers are able to find feasible solutions at all, where in some cases they would fail without.

Keywords. Supply chains, mixed integer models, preprocessing

AMS Classification. 90B10, 90C11

1 Introduction

Recently, continuous supply chain network models have gained attention from a theoretical as well as from a computational point of view [4, 5, 6, 13, 18, 19]. A continuous supply chain model has been derived from discrete event simulations in the case of many parts which is common to mass manufacturing processes [5]. Hence this model is in particular applicable

*Fachbereich Mathematik, Technische Universität Darmstadt, Germany
{dittel, fuegenschuh}@mathematik.tu-darmstadt.de

†Fachbereich Mathematik, Technische Universität Kaiserslautern, Germany
goettlich@mathematik.uni-kl.de

‡Fachgruppe Mathematik, Technische Hochschule Aachen, Germany
herty@mathc.rwth-aachen.de

in situations where the number of goods or parts in the network is huge compared to the number of machines and inventories. A continuous model typically consists of ordinary (ODE) and partial differential equations (PDE) describing the different physical reality as processing and stocking, respectively. In many applications the simulation of such a production process is only of minor importance, and a far more interesting question concerns the optimization of a given supply chain network with respect to network loads, sizes of the inventories or production costs. The numerical solution to such an optimization problem in general requires a suitable discretization of the optimality system and, in particular, a discrete approximation of the arising partial differential equations. In [16], a discretization has been proposed which finally yields a linear mixed-integer problem. However, without employing additional structure information the mixed-integer problem cannot be solved efficiently and even fails to be solvable in realtime. In this paper we are interested in efficient presolve techniques which can be applied in order to reduce the size of the mixed-integer problem and its computational solution time.

For general purpose mixed-integer linear programming (MIP), *preprocessing* or *presolving* refers to a bundle of different techniques that are known to decrease the solution time, that is, the time a numerical MIP solver needs to find an optimal solution and prove its optimality. Usually the presolving subroutine is called before the actual branch-and-cut solution process starts. Its purpose is to find and remove redundant parts of the problem formulation and thus output a somehow strengthened version of the model. In this work we focus on preprocessing techniques motivated by the underlying partial differential equation. The main idea is to use knowledge of the behavior of the underlying continuous partial differential equations in order to derive efficient presolve techniques. Finally, the most beneficial technique for solving MIPs coming from PDE discretizations turns out to be *bounds strengthening* together with a suitable *ordering* of the constraints. For a survey of other preprocessing techniques, we refer to [3, 12, 20, 29, 30].

We present our ideas on the example of an optimization problem for the continuous supply chain model. We give numerical results on the improvement of the solution time depending on the applied presolve technique. In particular, we observe that the applied preprocessing outperforms standard techniques that are implemented in most of the modern academic or commercial MIP solvers (such as Ilog Cplex).

2 Discrete Formulation of the Continuous Model

We start this section with briefly summarizing the continuous network model developed in [18, 19] and subsequently formulated as an optimal control problem, see [16, 23]. We present a valid discretization of the continuous model which can be considered as a MIP to solve the original PDE-constrained optimal control problem. A detailed derivation and analysis of the proposed MIP can be found in [16].

For convenience we motivate the basic model proposed in [18, 5]. The overall purpose is to model a production line or a supply chain that consists of processors for manufacturing goods and queues for buffering goods which have not yet been processed. Each processor is accompanied by a single queue. Goods leaving the processor move directly to the buffering queue of the next processor. To this end each processor e is characterized by a maximal processing capacity μ^e and a processing velocity v^e. The network of processors (production network) is modeled as a finite, directed graph $(\mathcal{V}, \mathcal{A})$ consisting of a set of vertices $\mathcal{V}$ and a set of arcs $\mathcal{A}$. Each arc $e \in \mathcal{A}$ corresponds to one processor and it is mapped on the interval $[a^e, b^e]$. The length of each processor is then determined by $L^e = b^e - a^e$, see Figure 1. The vertices are distribution or merging points of the production network. We denote for a fixed vertex v with multiple incoming and outgoing arcs, by δ_v^+ the set of all incoming arcs and by δ_v^- the set of all outgoing arcs, respectively. In the case of $|\delta_v^-| > 1$ there is freedom to distribute the incoming flux along the outgoing arcs. We introduce distribution rates $A_n^{v,e}$, $v \in \mathcal{V}_d$ where $\mathcal{V}_d \subset \mathcal{V}$ denotes the set of dispersing junctions, cf. Figure 2. The functions $A^{v,e}$ are required to satisfy $0 \leq A_n^{v,e} \leq 1$ and $\sum_{e \in \delta_v^+} A_n^{v,e} = 1$. Here, n denotes a timestep. For a fixed time horizon we introduce an equidistant time-grid with Δt as step size, N_T as total number of timesteps and with $t_n = n\Delta t$.

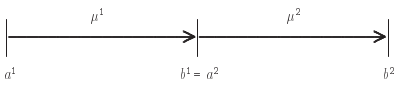

Figure 1: Example for two connected processors.

According to [19] a discrete version of the dynamic model describing the evolution of the goods *in processors* is given by (2.1).

$$\text{(PDE)}: \quad y_n^e = y_{n-1}^e + \frac{\Delta t}{L^e} v^e \left(x_{n-1}^e - y_{n-1}^e \right). \tag{2.1}$$

The previous equation is a two-point upwind discretization of a linear ad-

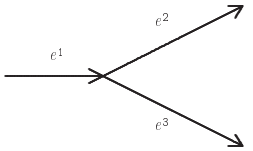

Figure 2: Example for a dispersing intersection. Here, we have only one variable control $A_n^{1,2}$ at time n, where $A_n^{1,3} = 1 - A_n^{1,2}$.

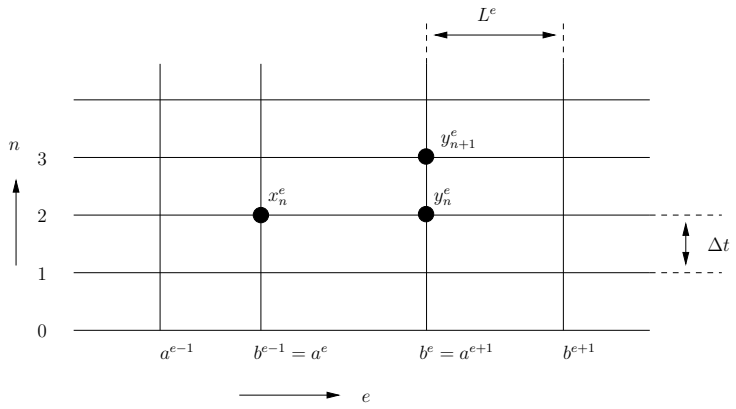

Figure 3: Two-point upwind discretization.

vection equation, see Figure 3. The variable x_n^e denotes the flux on arc e at point a^e at time n and the variable y_n^e denotes the flux on arc e at point b^e at time n. Due to this coarse space discretization, we have $\Delta x = L^e$.
Following [19, 4] a discrete version of the dynamic model describing the evolution of goods *in the buffering queue* is given by (2.2).

$$\text{(CPL)}: \quad \sum_{e \in \delta_v^+} z_n^e = \sum_{e \in \delta_v^-} y_{n-1}^e, \quad \forall\, v, n, \tag{2.2a}$$

$$\text{(QUE)}: \quad q_n^e = q_{n-1}^e + \Delta t \left(z_n^e - x_n^e \right), \quad \forall e, n, \tag{2.2b}$$

where the variable z_n^e denotes the total inflow to arc e. The previous equations is an explicit Euler discretization of an ordinary differential equation. Further, we enforce that the inventories are nonnegative,

$$q_n^e \geq 0, \, \forall e, n.$$

Since the maximal capacity of each processor implies an upper bound on the flux variables and we prescribe the boundary data depending on the load of

the queue, we have

$$0 \leq x_n^e \leq \mu^e, \quad 0 \leq y_n^e \leq \mu^e, \quad \forall e, n. \tag{2.3}$$

Note that, if the queue is empty, i.e., $q_{n-1}^e = 0$, or almost empty (except an error of ϵ^{reg}), i.e., $q_{n-1}^e \leq \epsilon^{\text{reg}}$, the outflow x_n^e is either given by the inflow z_n^e or by the maximal capacity μ^e. In the case of $z_n^e > \mu^e$, that means the maximal capacity of processor e is exceeded, the queue starts to build up. In contrast, if the queue is full, i.e., $q_{n-1}^e > 0$, the outgoing flow is always set to the maximal capacity and the queue will be reduced, see again [4, 23] for more details.

$$(\text{FLUX}): \quad x_n^e = \min\left\{\frac{q_{n-1}^e}{\epsilon^{\text{reg}}}, \mu^e\right\}, \quad \forall e, n. \tag{2.4}$$

Remark 2.1 *To ensure that the numerical solution of (2.1) will be an approximation of the correct solution we have to impose a restriction on Δt, called CFL condition. A further restriction on the time step size occurs if the queue-outflux is chosen as (2.4). Combining these facts we claim:*

$$\Delta t := \min\left\{\epsilon^{reg}, \frac{L^e}{v^e} : \ e \in \mathcal{A}\right\}. \tag{2.5}$$

The reformulation of equation (2.4) into a linear mixed-integer framework is as follows: We transform the non-linearity in (2.4) into linear constraints by introducing binary variables $\eta_n^e \in \{0, 1\}$ and a constant parameter $M \in \mathbb{R}_0^+$ defined as

$$M := \frac{T}{\epsilon^{\text{reg}}} \cdot \max_{e \in \mathcal{A}} \mu^e. \tag{2.6}$$

Then, equation (2.4) can be reformulated as:

$$\mu^e \eta_n^e \leq x_n^e \leq \mu^e, \tag{2.7a}$$

$$\frac{q_{n-1}^e}{\epsilon^{\text{reg}}} - M\eta_n^e \leq x_n^e \leq \frac{q_{n-1}^e}{\epsilon^{\text{reg}}}. \tag{2.7b}$$

For purposes of simulation only the discretized model is rewritten as an algorithm of the form for prescribed values of $A_n^{v,e}$:

`forwardsolutionPDE`

(1) **For** n **From** 1 **To** T **Do**
(2) **For All** $e \in A$ **Do**
(3) $y_n^e := y_{n-1}^e + \frac{\Delta t}{L^e} v^e \left(x_{n-1}^e - y_{n-1}^e\right)$; (PDE)
(4) $z_n^e := A_{n-1}^{v,e} \sum_{e \in \delta_v^-} y_{n-1}^e$; (CPL)
(5) $x_n^e := \min\left\{\frac{q_{n-1}^e}{\epsilon^{\text{reg}}}, \mu^e\right\}$; (FLUX)
(6) $q_n^e := q_{n-1}^e + \Delta t \left(z_n^e - x_n^e\right)$; (QUE)
(7) **End Do**
(8) **End Do**

Finally, we need to define initial values, i.e., at $n = 0$ we assume an empty network:

$$x_0^e = 0,\ y_0^e = 0,\ q_0^e = 0, \quad \forall e \in \mathcal{A}.$$

We prescribe inflow profiles $f_i^{in}(n)$ for all discrete time steps n on designated arcs i incoming to the network ($a^i = -\infty$).

For optimization purposes we define a cost functional. To this end, we introduce a functional measuring the sizes of the queues:

$$J = \sum_{e \in A, n \in T} q_n^e.$$

Other cost functionals can be treated similarly. Having the previous discussion in mind, the mixed-integer problem for minimizing queue lengths using rerouting in a supply chain is then stated as follows:

$$\begin{aligned}
&\min && \sum_{e \in A, n \in T} q_n^e, && (2.8a)\\
&\text{subject to} && && (2.8b)\\
&\text{(PDE)} && \forall\, e, n: \quad y_n^e = y_{n-1}^e + \frac{\Delta t}{L^e} v^e \left(x_{n-1}^e - y_{n-1}^e\right), && (2.8c)\\
&\text{(CPL)} && \forall\, v, n: \quad \sum_{e \in \delta_v^+} z_n^e = \sum_{e \in \delta_v^-} y_{n-1}^e, && (2.8d)\\
&\text{(FLUX1)} && \forall\, e, n: \quad \mu^e \eta_n^e \le x_n^e, && (2.8e)\\
&\text{(FLUX2)} && \forall\, e, n: \quad x_n^e \le \frac{q_{n-1}^e}{\epsilon^{\text{reg}}}, && (2.8f)\\
&\text{(FLUX3)} && \forall\, e, n: \quad \frac{q_{n-1}^e}{\epsilon^{\text{reg}}} - M\eta_n^e \le x_n^e, && (2.8g)\\
&\text{(QUE)} && \forall\, e, n: \quad q_n^e = q_{n-1}^e + \Delta t \left(z_n^e - x_n^e\right), && (2.8h)\\
&\text{(BND)} && \forall e, n: \quad 0 \le x_n^e \le \mu^e,\ 0 \le y_n^e \le \mu^e,\ 0 \le z_n^e,\ 0 \le q_n^e, && (2.8i)\\
& && \quad 0 \le \eta_n^e \le 1, && (2.8j)\\
&\text{(VAR)} && \forall e, n: \quad x_n^e, y_n^e, z_n^e, q_n^e \in \mathbb{R}, && (2.8k)\\
& && \quad \eta_n^e \in \{0, 1\}. && (2.8l)
\end{aligned}$$

3 Bounds Strengthening

A description of the bounds strengthening technique for general MIPs can be found in introductory textbooks, e.g. [22]. For convenience we recall the necessary steps in Appendix A. We use the terms `lowerBoundStrengthening`$(x; (\text{CONSTR}))$

and `upperBoundStrengthening`$(x;$ (CONSTR)), respectively, to indicate which bound of variable x will be considered for improvement using an equality or inequality constraint (CONSTR) out of (2.8c) – (2.8h) from the MIP model.

In the sequel we describe a hierarchy consisting of three successive levels of more and more evolved presolving techniques, all based on the bounds strengthening procedure and a careful reordering of the constraints. The first level, in the sequel called L1, refers to the direct application of the bounds strengthening procedure to a given instance of model (2.8) together with a suitable ordering of the constraints and variables. As the constraints come from a coupled PDE/ODE system which has been discretized by an upwind scheme, there is an inherent order of the constraints implied. Hence, we sort the constraints and apply bounds strengthening only in upwind direction of the flow, that is, we apply the following algorithm:

`presolveLevel1`

(1) **For** n **From** 1 **To** T **Do**
(2) **For All** $e \in A$ **Do**
(3) `boundsStrengthening`$(y_n^e;$ (PDE))
(4) `boundsStrengthening`$(z_n^e;$ (CPL))
(5) `lowerBoundStrengthening`$(x_n^e;$ (FLUX1))
(6) `upperBoundStrengthening`$(x_n^e;$ (FLUX2))
(7) `lowerBoundStrengthening`$(x_n^e;$ (FLUX3))
(8) `upperBoundStrengthening`$(\eta_n^e;$ (FLUX1))
(9) `lowerBoundStrengthening`$(\eta_n^e;$ (FLUX3))
(10) `lowerBoundStrengthening`$(x_n^e;$ (FLUX1))
(11) `lowerBoundStrengthening`$(x_n^e;$ (FLUX3))
(12) `boundsStrengthening`$(q_n^e;$ (QUE))
(13) **End Do**
(14) **End Do**

Note that steps (10) and (11) are a repetition of steps (5) and (7), respectively. This is necessary since the update of the bounds from other steps, namely the bounds on the binaries η_n^e in step (8) and (9), can have a further influence on the bounds of these variables.

To the best of our knowledge there exists only few theoretical results on the number of iterations a bounds strengthening procedure will require. Only if one puts additional assumptions on the structure of the constraint system some results in this direction are known. For example in the case of bounded integer programming problems with at most two non-zero coefficients per inequality (IP2, for short), Bar-Yehuda and Rawitz [7] proved that the bounds strengthening routine can be implemented such that it terminates on those systems in pseudopolynomial time $O(mU)$, where m is the number of inequalities and $U := \max\{u_j - l_j : j\}$ is the largest differ-

ence between lower and upper bound among all variables. In this spirit we formulate the result dealing with the behavior of the bounds strengthening routine on the model. In particular, we implemented it in such way that only one round is necessary to strengthen all bounds. It is also sufficient and no further updates of bounds are found in a second round.

Theorem 3.1 *Let an instance of model (2.8) be given. We assume that the bounds on all variables are strengthened by calling* `presolveLevel1`*. Then no further update on any bound is found when calling* `presolveLevel1` *again.*

Proof. As abbreviations we write $\underline{x}^e_n, \overline{x}^e_n, \underline{y}^e_n, \overline{y}^e_n, \underline{z}^e_n, \overline{z}^e_n, \underline{q}^e_n, \overline{q}^e_n, \underline{\eta}^e_n, \overline{\eta}^e_n$ for the lower and upper bounds on the variables $x^e_n, y^e_n, z^e_n, q^e_n, \eta^e_n$, respectively.

We prove the claim by induction over the timesteps n. For $n = 1$ there is nothing to show since all lower and upper bounds are initialized with the initial values of the system (i.e., they are all set to 0). Suppose now the statement is true for all timesteps up to $n-1$. Then we deduce that it also has to be true for the next timestep n.

Suppose there is an update on the lower bound $\underline{y}^e_n$ of variable y^e_n. The only step that touches this bound in algorithm `presolveLevel1` is step (3). From there we must have that, if $1 - \frac{\Delta t}{L^e} v^e > 0$,

$$\underline{y}^e_n < \left(1 - \frac{\Delta t}{L^e} v^e\right) \underline{y}^e_{n-1} + \frac{\Delta t}{L^e} v^e \underline{x}^e_{n-1}. \tag{3.1}$$

By assumption the bounds $\underline{y}^e_{n-1}$ and $\underline{x}^e_{n-1}$ did not change in this round (i.e., during the application of `presolveLevel1` for the second time), hence we also have

$$\underline{y}^e_n \geq \left(1 - \frac{\Delta t}{L^e} v^e\right) \underline{y}^e_{n-1} + \frac{\Delta t}{L^e} v^e \underline{x}^e_{n-1}, \tag{3.2}$$

which is the desired contradiction. With an analogue argument we derive a contradiction if $1 - \frac{\Delta t}{L^e} v^e < 0$.

Suppose there is an update on the upper bound $\overline{y}^e_n$, then we also obtain a contradiction following the arguments from above. The same can be done for the lower or upper bounds on z^e_n using (CPL), and the bounds on q^e_n using (QUE). For $\overline{z}^e_n$ and constraint (CPL) we remark that $\underline{z}^e_n = 0$ for all arcs e and timesteps n, hence there is no contribution when updating the upper bound on z^e_n.

The more complicated cases are the lower bound on the variables x^e_n and the lower and upper bound on η^e_n. We start with the proof that $\overline{\eta}^e_n$ has not changed.

Suppose that there is an update on $\overline{\eta}_n^e$ due to constraint (FLUX1) in step (8) of `presolveLevel1`. That means,

$$1 = \overline{\eta}_n^e > \lfloor \frac{1}{\mu^e} \overline{x}_n^e \rfloor = 0. \tag{3.3}$$

This update can only occur if the upper bound $\overline{x}_n^e$ was changed before, which can only come from step (6) and constraint (FLUX2). From this we deduce

$$\overline{x}_n^e > \frac{\overline{q}_{n-1}^e}{\epsilon^{\text{reg}}}, \tag{3.4}$$

hence the upper bound $\overline{q}_{n-1}^e$ also has changed before, which is in contradiction to the induction's assumption.

Suppose that $\underline{\eta}_n^e$ will be changed due to constraint (FLUX3) in step (9). Then

$$0 = \underline{\eta}_n^e < \lceil -\frac{\overline{x}_n^e}{M} + \frac{\underline{q}_{n-1}^e}{M\epsilon^{\text{reg}}} \rceil = 1. \tag{3.5}$$

Hence $\underline{q}_{n-1}^e$ or $\overline{x}_n^e$ has changed before. Since there cannot be a change of $\underline{q}_{n-1}^e$ by induction assumption, it is $\overline{x}_n^e$ that has changed. This is, as above, only possible in step (6), from which we already know that a contradiction follows.

Suppose that $\underline{x}_n^e$ will be changed due to step (10). That means from (FLUX1) we have

$$\underline{x}_n^e < \mu^e \underline{\eta}_n^e. \tag{3.6}$$

Hence $\underline{\eta}_n^e$ was updated earlier in step (9). Above we already argued that this leads to a contradiction.

And finally, suppose that $\underline{x}_n^e$ will be changed due to step (11). Then

$$\underline{x}_n^e < \frac{\underline{q}_{n-1}^e}{\epsilon^{\text{reg}}} - M \cdot \overline{\eta}_n^e. \tag{3.7}$$

Since $\overline{q}_{n-1}^e$ did not change by assumption, it is $\overline{\eta}_n^e$ that must have been changed. This however is only possible in step (8), from which we already know that this leads also to a contradiction. □

This theorem implies that for model (2.8) bounds strengthening can be implemented in such way that it needs $T \cdot A$ many steps (each step consists of a constant number of arithmetic operations), where T is the number of timesteps and A is the number of arcs (processors).

3.1 Presolving for Constraints with Binary Variables

Note that the binary variables η_n^e in the MIP model are in fact artificial auxiliaries. Their only purpose is to transform the nonlinear constraint (FLUX) into linear ones. The nonlinearity here is due to the min operator. This, however, comes at the price of introducing the infamous big-M formulation, where a suitable large value for M serves as an upper bound on the values in min operator, i.e., a bound on $\frac{q_{n-1}^e}{\epsilon^{\mathrm{reg}}}$. Since q_{n-1}^e can be quite large, and ϵ^{reg} is typically small, one will expect large values for M. This can cause numerical trouble for the bounds strengthening procedure as well as for the subsequent linear programming algorithm. For the bounds strengthening, however, we do not need to resort to the linearized formulation (FLUX1), (FLUX2), and (FLUX3). We can directly make use of the nonlinear constraint (FLUX) instead.

We introduce a new procedure `nonlinearBoundsStrengthening`$(x_n^e;$ (FLUX)), which performs the following update steps:

$$\overline{x}_n^e := \min\left\{\overline{x}_n^e, \min\left\{\frac{\overline{q}_{n-1}^e}{\epsilon^{\mathrm{reg}}}, \mu^e\right\}\right\}, \tag{3.8}$$

$$\underline{x}_n^e := \max\left\{\underline{x}_n^e, \min\left\{\frac{\underline{q}_{n-1}^e}{\epsilon^{\mathrm{reg}}}, \mu^e\right\}\right\}. \tag{3.9}$$

Hence the steps (5) to (13) in `presolveLevel1` now shrink to a single step, and the overall level 2 presolve algorithm is the following:

`presolveLevel2`

(1) **For** n **From** 1 **To** T **Do**
(2) **For All** $e \in A$ **Do**
(3) `boundsStrengthening`$(y_n^e;$ (PDE))
(4) `boundsStrengthening`$(z_n^e;$ (CPL))
(5) `nonlinearBoundsStrengthening`$(x_n^e;$ (FLUX))
(6) `boundsStrengthening`$(q_n^e;$ (QUE))
(7) **End Do**
(8) **End Do**

We remark that Theorem 3.1 can be carried over to `presolveLevel2` to show, as before, that there are no further updates when applying the same routine a second time.

3.2 Aggregation of Constraints

Consider constraint (QUE) for timestep n and arc e, and insert constraint (FLUX) for x_n^e, then we arrive at

$$\text{(AGG)} \quad q_n^e = q_{n-1}^e + \Delta t \left(z_n^e - \min \left\{ \frac{q_{n-1}^e}{\epsilon^{\text{reg}}}, \mu^e \right\} \right). \tag{3.10}$$

Note that by this aggregation we removed the dependency of (QUE) from the variable x_n^e. Thus, when applying (nonlinear) bounds strengthening to this new constraint, one can expect better bounds in some cases. Namely,

- if $\frac{\overline{q}_{n-1}^e}{\epsilon^{\text{reg}}} \leq \mu^e$ we additionally have the following update on the upper bound of q_n^e:

$$\overline{q}_n^e := \min \left\{ \overline{q}_n^e, \overline{q}_{n-1}^e \left(1 - \frac{\Delta t}{\epsilon^{\text{reg}}} \right) + \Delta t \cdot \overline{z}_n^e \right\}, \tag{3.11}$$

- and if $\frac{\underline{q}_{n-1}^e}{\epsilon^{\text{reg}}} \geq \mu^e$ we can update the lower bound of q_n^e as follows:

$$\underline{q}_n^e := \max \left\{ \underline{q}_n^e, \underline{q}_{n-1}^e + \Delta t (\underline{z}_n^e - \mu^e) \right\}. \tag{3.12}$$

Hence the highest level of presolving L3 consists of the following steps:

`presolveLevel3`

(1) **For** n **From** 1 **To** T **Do**
(2) **For All** $e \in A$ **Do**
(3) `boundsStrengthening`(y_n^e; (PDE))
(4) `boundsStrengthening`(z_n^e; (CPL))
(5) `nonlinearBoundsStrengthening`(x_n^e; (FLUX))
(6) `boundsStrengthening`(q_n^e; (QUE))
(7) `nonlinearBoundsStrengthening`(q_n^e; (AGG))
(8) **End Do**
(9) **End Do**

4 Computational Results

We test our three presolve algorithms, which we call *scmip-presolve* in the sequel, on selected test instances defined by a network (with capacities, velocities and lengths given for each processor), an inflow profile and the duration of simulation. The algorithms are implemented in C++ and compiled with GNU gcc 3.2.3. All computations are performed on a 2.4 GHz

AMD64X2 processor with 2 GB RAM on a Linux platform. The MIPs are solved by Ilog Cplex 10.0 [21] with default settings.

For each problem instance, we generate MIPs containing the mixed-integer formulated problem after scmip-presolving with presolve levels L1, L2, and L3. These MIPs contain only the variables that have not been fixed by scmip-presolve, and the constraints they appear in. For comparison, we also generate the "pure" MIP without scmip-presolving. We measure the computing time for the solution of the MIPs by considering the runtime for scmip-presolve (L1, L2, L3), and the solution time consumed by Cplex (divided into Cplex presolve time and solution time for the presolved MIP).

4.1 Chain of Processors

For instances of networks without branches, the flux is uniquely determined by the PDE/ODE constraints.

We choose a line of five processors with processor capacities, velocities and lengths as shown in Table 1.

Processor	μ^e	v^e	L^e
0	100	100	1
1	20	10	1
2	6	5	1
3	2	7	1
4	5	1	1

Table 1: Processing rates μ^e, processing velocities v^e and lengths L^e.

The total time horizon for the optimization is given by $T = 60$, and the timestep size is $\Delta t = 0.01$. We prescribe an inflow profile on the first arc $e = 0$ by the function $f_0^{in}(t) = 25$ for $t \leq 4$ and $f_0^{in}(t) = 0$ otherwise. Although the model is deterministic for this problem instance, it provides an example in which the gap between lower and upper bound on each variable cannot be closed by application of our simplest form of bounds strengthening (L1). Obtaining the solution by presolve, requires both the application of the special procedure for the minimum and the aggregation of the queue equations, which refers to our highest level of bounds strengthening (L3). In Table 2, we show the percentages of variables whose value is fixed after application of scmip-presolve in the levels L1, L2, and L3.
We interprete the results in Table 2 as follows. Obviously, there is no optimization involved in solving the problem on the chain of processors. Taking

Level	Processor					Average
	0	1	2	3	4	
L1	100	26.70	20.06	61.16	22.27	46.04
L2	100	63.36	34.55	64.59	22.31	56.96
L3	100	100	100	100	100	100

Table 2: Percentage of number of variables where upper and lower bound coincide.

a closer look at the equations (PDE)–(QUE), we observe the following: Assume at timestep $n-1$ all upper and lower bounds on the variables y^e_{n-1}, x^e_{n-1} and q^e_{n-1} coincide (this is for example the case for $n = 1$.) For simplicity assume additionally $q^e_{n-1}/\epsilon < \mu^e$. Then, y^e_n and z^e_n are uniquely defined by (PDE, CPL) or equivalently step 3 and 4 in `presolveLevel1`. Using preprocessing algorithm L1, we then update the bounds on x^e_n by considering equation the steps 5–7. At last we update the bounds on η^e_n using step 8 and 9, respectively. Due to this ordering of the equations we only obtain $x^e_n \in [0, \frac{q^e_{n-1}}{\epsilon}]$ and $\eta^e_n = 0$. This in turn implies bounds on q^e_n in (QUE), i.e., by step 12 of the algorithm. Hence, using L1 after the first processor we cannot expect coinciding lower and upper bounds on all variables. Note that a simple change in the order of the presolve procedure does not help for improving the bounds, since the variable η^e_n and x^e_n are coupled. However, there is a remedy for this problem in some cases: We explicitly solve equation (FLUX) instead of using equations (5) - (12). As seen in Section 3.3 this cannot be done for any values of q^e_{n-1} and x^e_n and therefore, there are still variables left where the upper and the lower bound differ (row L2 in Table 2). A further improvement can be obtained by aggregation, i.e., the update using equation (QUE) will be independent on the update of equation (FLUX). Of course, this yields better results for the variables q^e_{n-1} as seen in row L3 of Table 2. Finally, we observe that in this example combining all preprocessing strategies, i.e., the 'correct' ordering of the equations, the direct computation of the minimum in (FLUX) and the aggregation in (QUE) strongly improves the result.

The plots in Figure 4 show the gaps between lower and upper bound remaining after scmip-presolve in the levels L1, L2, and L3 for the outflux y and queue q for processor 4.
We give a survey on the computing times for preprocessing in Table 3 for the original MIP and for the MIP remaining after applying scmip-presolve with

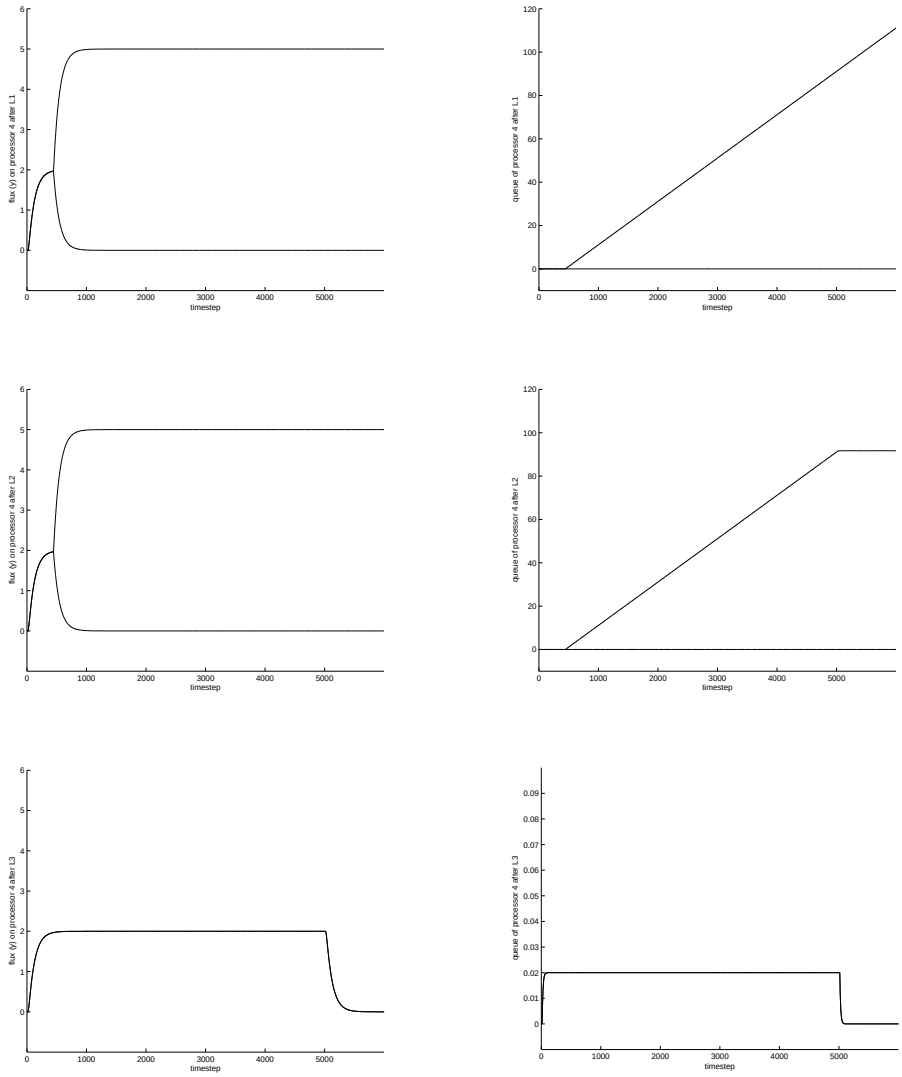

Figure 4: Bounds on flux y and queue q after L1, L2, and L3

L1, L2, and L3, respectively. The first column contains the presolve level, where "Cplex" stands for the application of the default routines of the Cplex solver, and no preprocessing by scmip-presolve. The second column shows the computing time consumed by the particular level in seconds. Column three shows the computing time in seconds needed by Cplex for the solution of the presolved MIP via its branch-and-cut method (b&c).

presolve	preprocessing	solution (Cplex b&c)
Cplex	1.62 sec	infeasible
L1	0.18 sec	17.94 sec
L2	0.17 sec	1.81 sec
L3	0.27 sec	0.01 sec

Table 3: Performance of the presolve algorithm

The most striking fact is the infeasibility of the remaining MIP after preprocessing by Cplex only. Since (commercial) MIP solvers are black-boxes from the end users' point of view, we can only speculate that due to round-off errors induced by Cplex' own preprocessing routines the problem is considered infeasible. This is clearly wrong and our newly introduced presolve routines, in particular L3, give the numerically correct solution. Further, we observe that the new presolve steps take only roughly 10 % of the computing time of the presolve used by Cplex. As expected the more sophisticated presolve techniques yield lower solution times of the remaining MIP. This trend is also observed in the following examples.

4.2 4×4 Block Network

The test instance is a block network with 4×4 interior network vertices, four inflow arcs $(i_1, \ldots, i_4)$, and four outflow arcs $(o_1, \ldots, o_4)$, see Figure 5. For the inflow arcs $i_1, \ldots, i_4$, we choose maximal processing rates of $\mu^i = 100$ and velocities $v^i = 100$ $(i \in \{i_1, \ldots, i_4\})$. For the remaining arcs, we chose $\mu^e = 1$ and $v^e = 1$. The processor lengths are $L^e = 1$ for all network arcs. We choose a time horizon for the optimization of $T = 10$ time units and a timestep of $\Delta t = 0.01$. The inflow profile on each network inflow arc $i \in \{i_1, \ldots, i_4\}$ is given by $f_i^{in}(t) = 6$ for $t \leq 3$ and $f_i^{in}(t) = 0$ otherwise.

In this network there is the possibility to distribute flux at each vertex. Consequently, the constraints of the MIP do not define a unique feasible solution. This implies that the problem cannot be solved in the presolve step. As in the example above, we solve the MIP by Cplex, both without

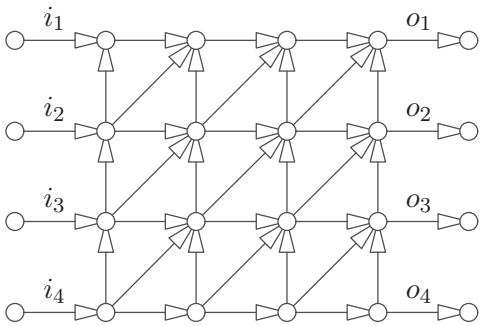

Figure 5: 4×4 block network

scmip-presolve and in the versions presolved by L1, L2, L3. The results are listed in Table 4. Since the problem is not solvable in presolve in this case, there is also computational effort for the solution of the presolved MIP via an LP based branch-and-cut approach. Columns 1 contains the presolve level and column 2 the presolve times in seconds for scmip-presolve. In columns 3 and 4, we list the computing times consumed by Cplex, divided into time for presolve (column 3) and solution of the root relaxation (column 4). Column 5 shows the total solution time on the part of Cplex.

preprocessing			solution (Cplex b&c)	
level	scmip pre	Cplex pre	root LP	total
Cplex	–	450.00 sec	334.21 sec	785.85 sec
L1	0.33 sec	453.54 sec	383.93 sec	838.18 sec
L2	0.34 sec	460.38 sec	384.32 sec	845.42 sec
L3	0.34 sec	1.46 sec	276.43 sec	278.25 sec

Table 4: Computation times in seconds for presolved MIPs

We observe that the more sophisticated presolve techniques strongly outperform Cplex. Further, the computing time needed for the proposed presolve is small compared with the presolve time of Cplex. This behavior is theoretically explained by the assertion of Theorem 1, which states that a single round of each respective bounds strengthening procedure L1, L2, L3 is sufficient, whereas a general purpose MIP solver (such as Cplex) uses more heuristically motivated rules to determine the number of rounds in the preprocessing.

4.3 A real-world example

Initially our interest in supply chain models was stimulated by the following real world problem. During the set up of a new production factury the engineers wanted to design a network consisting of 12 processors (i.e., arcs in our model), where unfinished products are circling around from one processor to the next on small pallets. For further details of this application we refer to [16]. The parameter setting is such that queues are built up in front of processor $2, \ldots, 8$. Again, the aim of optimization is to minimize the sizes of the queues.

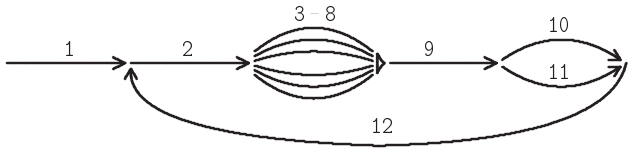

Figure 6: Real-world network

We are basically interested in two special situations: $\epsilon^{\text{reg}} = 1$ and $\epsilon^{\text{reg}} = 0.5$. Due to the CFL condition (2.5), the parameter ϵ^{reg} determines the time step size Δt and consequently the total amount of variables of the optimization problem.

	preprocessing			solution (b&c)	
ϵ^{reg}	level	scmip pre	Cplex pre	root LP	total
1	Cplex	–	3.64 sec	0.72 sec	224.6 sec
1	L3	0.02 sec	0.56 sec	0.81 sec	11.05 sec
0.5	Cplex	–	14.6 sec	4.84 sec	no sol. found
0.5	L3	0.05 sec	0.89 sec	3.96 sec	10231.34 sec

Table 5: Computation times in seconds for presolved MIPs

In test case $\epsilon^{\text{reg}} = 1$ the total solution time of presolve technique L3 is 20 times faster than Cplex. But the most astonishing result is the second one. For the choice of $\epsilon^{\text{reg}} = 0.5$ Cplex was not able to find to any feasible solution within 24 hours. However, together with our approach the problem could be solved in nearly 3 hours. This example underlines the necessity of problem-based presolve techniques to even guarantee solvability.

5 Further Applications of the Presolve Techniques

Even so the presented techniques where devised for the particular system (2.8), some preprocessing techniques to further applications discussed below. Of particular importance is the `presolveLevel1` which mimics the PDE–dynamics. The presolve Levels `presolveLevel2` and `presolveLevel3` are related to the structure of our problem and are thus restricted to models having a similar constraint structure.

The technique `presolveLevel1` is applicable to all transport processes on networks. Its basic idea is the finite speed of propagation of information across an arc of the network. Such problems arise for example in water contamination problems [25, 26, 27], in some traffic assignment problems [8, 10, 17, 24, 28], in data communication problems [11] or supply chain problems [1, 16]. All these models can be described by the following general transport process on an arc e of a network

$$y_t^e + f_x(y^e) = g(y^e), \qquad \forall\, x \in [a^e, b^e], t \geq 0. \tag{5.1}$$

Here, $f_y(y^e)$ is the transport velocity and g an external source term. Our technique applies whenever f and g are linear in y. For example, in the case of water contamination detection we have $f(y^e) = v^e\, y^e$ and $g(y^e) = -c^e y^e$. Therein, c^e is the velocity for the transport of the containment and $-y^e$ describes a decay process for the containment of rate c^e. In the example of a simple traffic assignment without jams the source term g vanishes and we can approximate $f(y^e)$ again by $f(y^e) = v^e y^e$. Similar, assertions are made for supply chain and data networks, see [10] and [16]. In the case of linear transport and decay a possible two–point discretization of (5.1) is given by

$$y_{n+1}^e = y_n^e + \frac{\Delta t}{\Delta x} v^e\, (y_n^e - x_n^e) - c^e \Delta t y_{n+1}^e. \tag{5.2}$$

Here, $y(a^e, t) \approx y_n^e$ and $y(b^e, t) \approx x_n^e$. We also set $\Delta x = b^e - a^e$. Clearly, (5.2) is similar to (2.1).

The different applications yield different coupling conditions for connected arcs. Those coupling conditions have also to be discretized and further non–linearities arise. In general, we obtain a discretized system of the following type

$$\Phi\left((y_n^{e_i})_{e_i \in \delta_v^-}, (x_n^{e_j})_{e_j \in \delta_v^+} \right) = 0 \tag{5.3}$$

where e_i and e_j are connected arcs at the junction v. Here, Φ contains the detailed modeling of the physical process at the vertex. Typical coupling

conditions in the case of traffic assignment are

$$\Phi\left((y_n^{e_i})_{e_i\in\delta_v^-},(x_n^{e_j})_{e_j\in\delta_v^+}\right)=\sum_{e_i\in\delta_v^+}x_n^{e_i}-\sum_{e_j\in\delta_v^-}y_n^{e_j}. \tag{5.4}$$

Other examples can be found in [26, 16, 17]. Given the general discretization for a linear(!) function Φ the obvious modification of `presolveLevel1` is as follows.

`presolveLevel1`

(1) **For** n **From** 1 **To** T **Do**
(2) **For All** $e \in A$ **Do**
(3) `boundsStrengthening`(y_n^e; (5.2))
(4) `boundsStrengthening`(x_n^e; (5.3))
(5) **End Do**
(6) **End Do**

Note that depending on the details of the coupling condition or further nonlinearities steps might be necessary. This is exemplified in Section 3 in the case of the supply chain model. Therein, additional $\min-$ condition are discretized and further `boundStrengthening` steps are necessary. Additionally, we might add bound constraints on x_n^e and y_n^e to describe maximal part flows, maximal containments, or other restrictions. Those constraints can be included into the presolve in the same way as describe in this article.

6 Summary

We used the order of constraints induced by the ODE/PDE system and transferred this to a preprocessing algorithm by the means of bounds strengthening. Furthermore, we applied some special presolving techniques for the resolution of the nonlinear constraints avoiding the direct preprocessing by bounds strengthening upon the linearized constraints containing binary variables. The results outperform the black-box Cplex preprocessing techniques. In particular, some instances have been considered infeasible by Cplex – most likely – due to roundoff errors in the preprocessing engine. These instances could be solved by the proposed techniques. Further, the presolve method is efficient even for large-scale problems. The techniques have been demonstrated by an example of optimizing a production network, but apply similarly to other discretized PDE-constrained optimization problems.

Acknowledgements

This work was supported by the University of Kaiserslautern Excellence Cluster 'Dependable Adaptive Systems and Mathematical Modeling' and the Procope project D/0628176 of the Deutscher Akademischer Austauschdienst DAAD, as well as by the DFG Collaborative Research Center SFB-666 at the Darmstadt University of Technology. Parts of this work were carried out at the Hausdorff Institute for Mathematics, Bonn. We are grateful for their kind hospitality.

References

[1] D'Apice, C., and Manzo, R. A fluid dynamic model for supply chains. *Netw. Heterog. Media*, Vol. 1, pp. 379–398, 2006.

[2] D'Apice, C., and Manzo, R. Packet flow on telecommunication networks. *SIAM J. Math. Anal.*, Vol. 38, pp. 717–740, 2006.

[3] Andersen, E.D., and Andersen, K.D. Presolving in linear programming. *Mathematical Programming*, Vol. 71, pp. 221 – 245, 1995.

[4] Armbruster, D., de Beer, C., Freitag, M., Jagalski, T., and Ringhofer, C. Autonomous Control of Production Networks using a Pheromone Approach. *PHYSICA A*, Vol. 363(1), 2006.

[5] Armbruster, D., Degond, P., and Ringhofer, C. A model for the dynamics of large queuing networks and supply chains. *SIAM J. on Applied Mathematics*, Vol. 66, pp. 896–920, 2006.

[6] Armbruster, D., Marthaler, D., and Ringhofer, C. Kinetic and Fluid Model Hierarchies for Supply Chains. *SIAM J. Multiscale Modeling and Simulation*, Vol.2 (1), pp. 43-61, 2004.

[7] Bar-Yehuda, R., and Rawitz, D. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, Vol. 29 (4), pp. 595 – 609, 2001.

[8] Blinkin, M. Problem of optimal control of traffic flow on highways. *Automation and Remote Control*, Vol. 37, pp. 662–667, 1976.

[9] Brearley, A.L., Mitra, G., and Williams, H.P. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, Vol. 8, pp. 54 – 83, 1975.

[10] Bretti, G, Natalini, R. and Piccoli, B. Fast algorithms for the approximation of a traffic flow model on networks. *Discrete Contin. Dyn. Syst.*, Ser. B(6), pp. 427–448, 2006.

[11] Bretti, G., D'Apice, C., Manzo, R. and Piccoli, B. A continuum-discrete model for supply chains dynamics. to appear in *Networks and Heterogeneous Media*, (2008).

[12] Crowder, H., Johnson, E., and Padberg, M.W. Solving large-scale zero-one linear programming problems. *Operations Research*, Vol. 31, 803 – 834, 1983.

[13] Daganzo, C.F. *A Theory of Supply Chains.* Springer Verlag, New York, Berlin, Heidelberg, 2003.

[14] Fügenschuh, A., Göttlich, S., and Herty, M. A New Modeling Approach for an Integrated Simulation and Optimization of Production Networks. H.-O. Günther, D. Mattfeld, L. Suhl (Eds.), *Management logistischer Netzwerke.* Physica-Verlag Heidelberg, 45 – 60, 2007.

[15] Fügenschuh, A., Göttlich S., and Herty, M. Water Contamination Detection. In: A. Oberweis, C. Weinhardt, H. Gimpel, A. Koschmider, V. Pankratius, B. Schnizler (Eds.), *eOrganisation: Service-, Prozess-, Market-Engineering.* 8. Internationale Tagung Wirtschaftsinformatik, Universitätsverlag Karlsruhe, Karlsruhe, 501 – 518, 2007.

[16] Fügenschuh, A., Göttlich, S., Herty, M., Klar, A., and Martin, A. A Discrete Optimization Approach to Large Scale Supply Networks based on Partial Differential Equations. SIAM J. on Scientific Computing, Vol. 30(3), pp. 1490-1507, 2008.

[17] Fügenschuh, A., Herty, M., Klar, A., and Martin, A. Combinatorial and continuous models for the optimization of traffic flow networks. *SIAM J. Opt.*, 2005.

[18] Göttlich, S., Herty, M., and Klar, A. Network models for supply chains. *Comm. Math. Sci.*, Vol. 3(4), pp. 545–559, 2005.

[19] Göttlich, S., Herty M., and Klar, A. Modelling and optimization of supply chains on complex networks. *Comm. Math. Sci.*, Vol. 4(2), pp. 315–330, 2006.

[20] Hoffman, K., and Padberg, M. Improving Representations of Zero-one Linear Programs for Branch-and-Cut. *ORSA Journal of Computing*, Vol. 3, pp. 121–134, 1991.

[21] ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. Information available at URL http://www.cplex.com.

[22] Kallrath, J. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis.* Vieweg Verlag, Wiesbaden, 2002.

[23] Kirchner, C., Herty, M., Göttlich, S., and Klar, A. Optimal Control for Continuous Supply Network Models. *Networks and Heterogenous Media*, Vol. 1(4), pp. 675–688, 2006.

[24] Kotsialos, A, Papageorgiou, M., Mangeas, M. and Haj-Salem, H. Coordinated and integrated control of motorway networks via non-linear optimal control. *Transport Research Part C*, Vol. 10, pp. 65–84, 2002.

[25] Laird, C., Biegler, L., van Bloemen Waanders, B. and Bartlett, R. Contamination source determination for water networks. *A.S.C.E. J. of Water Resources Planning and Management*, Vol. 131(2), pp. 125 – 134, 2005.

[26] Laird, C., Biegler, L. and van Bloemen Waanders, B. Real-time, large scale optimization of water network systems using a subdomain approach. To appear in proceedings of the *Second CSRI Conference on PDE-Constrained Optimization*.

[27] Laird, C., Biegler, L. and van Bloemen Waanders, B. A mixed integer approach for obtaining unique solutions in source inversion of drinking water networks. *J. of Water Resources Planning and Management*, Vol. 132(4), pp. 242 – 251, 2006.

[28] Lebacque, J and Khoshyaran, M. First order macroscopic traffic flow models for networks in the contex of dynamic assignment. *Transportation Planning-State of the Art*, 2002.

[29] Savelsbergh, M.W.P. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, Vol. 6(4), pp. 445 – 454, 1994.

[30] Suhl, U.H., and Szymanski, R. Supernode processing of mixed-integer models *Computational Optimization and Applications*, Vol. 3, pp. 317 – 331, 1994.

A Bounds Strengthening in General

For convenience we give a brief description of bounds strengthening for general mixed-integer programming problems.

Let an arbitrary MIP

$$\begin{aligned} \min \quad & c^T x && \text{(A.1)} \\ \text{s.t.} \quad & Ax \left\{ \begin{matrix} \leq \\ = \end{matrix} \right\} b && \text{(A.2)} \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} && \text{(A.3)} \end{aligned}$$

be given. We consider the i-th inequality of the constraint system $Ax \left\{ \begin{matrix} \leq \\ = \end{matrix} \right\} b$, which is of the following form:

$$\text{(IEQ)} \quad a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i. \tag{A.4}$$

Each variable x_j has lower and upper bounds l_j, u_j with $l_j \leq x_j \leq u_j$ and $l_j \in \mathbb{R} \cup \{-\infty\}, u_j \in \mathbb{R} \cup \{+\infty\}$. If for some $j \in \{1, ..., n\}$ we have $\overline{l}_j > l_j$ or $\overline{u}_j < u_j$, and the set of feasible solution does not change, i.e.,

$$\begin{aligned} & \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \leq b, l \leq x \leq u\} \\ = \quad & \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax \leq b, l \leq x \leq u, \overline{l}_j \leq x_j \leq \overline{u}_j\}, \end{aligned}$$

we say that $\overline{l}_j, \overline{u}_j$ are *improved lower and upper bounds*, respectively. In principle, best possible bounds can be obtained by taking each variable x_j as objective function and solve a minimization (for the lower bound) and a maximization problem (for the upper bound). However, such a procedure would be by far too time consuming in practice. *Bounds strengthening* now is a much simpler technique to obtain such improved bounds by using informations solely from the constraint systems $Ax \leq b$ and the given bounds $l \leq x \leq u$.

Select an inequality i and a variable with index j. Then (A.4) is equivalent to

$$a_j x_j \leq b - \sum_{\substack{k=1,\ldots,n \\ k \neq j}} a_k x_k. \tag{A.5}$$

As abbreviations we define

$$l_j^+ := \sum_{\substack{k=1,\ldots,n \\ k \neq j}} \max\{a_k, 0\} \cdot l_k, \qquad l_j^- := \sum_{\substack{k=1,\ldots,n \\ k \neq j}} \min\{a_k, 0\} \cdot l_k,$$
$$u_j^+ := \sum_{\substack{k=1,\ldots,n \\ k \neq j}} \max\{a_k, 0\} \cdot u_k, \qquad u_j^- := \sum_{\substack{k=1,\ldots,n \\ k \neq j}} \min\{a_k, 0\} \cdot u_k.$$

Here we use the intuitive rules for evalutating expressions containing $\pm\infty$, namely, $l_j^+ = -\infty$ if for one $k \neq j$ with $a_k > 0$ it is $l_k = -\infty$, for instance.

We distinguish two cases. For $a_j > 0$ it follows from (A.5) that

$$x_j \leq \frac{1}{a_j}(b - (l_j^+ + u_j^-)) =: \overline{u}_j. \tag{A.6}$$

If x_j is a continuous variable (i.e., for $j > p$) we have an improved upper bound on x_j if $\overline{u}_j < u_j$. That is, we set $u_j := \min\{u_j, \overline{u}_j\}$. For an integer variable x_j (for $j \leq p$) we can improve this bound further by rounding it down to the next integer value, thus $u_j := \min\{u_j, \lfloor \overline{u}_j \rfloor\}$. We denote by `upperBoundStrengthening`$(x_j;$ (IEQ)) the algorithm that tries to update the upper bound on variable x_j using the information in inequality (IEQ).

In the other case, if $a_j < 0$, we analogously conclude that

$$x_j \geq \frac{1}{a_j}(b - (l_j^- + u_j^+)) =: \overline{l}_j, \tag{A.7}$$

and obtain an improved lower bound $l_j := \max\{l_j, \overline{l}_j\}$ on a continuous variable x_j, and $l_j := \max\{l_j, \lceil \overline{l}_j \rceil\}$ on an integer variable x_j. We call this algorithm `lowerBoundStrengthening`$(x_j;$ (IEQ)).

In general mixed-integer preprocessing, the above steps are carried out iteratively for all constraints $i = 1, \ldots, m$ and for all variables $j = 1, \ldots, n$. Then the next round of presolve starts again from the beginning. This is performed until either no improved bounds are found anymore, or an infeasibility is detected. The latter is the case if after an update of the bounds we obtain $l_j > u_j$, and hence the MIP has no feasible solution at all due to a contradiction in the constraint system. In practice, a termination of the bounds strengthening procedure might also be forced if a given number of iteration rounds is reached, or if the improvements in the bounds fall below a certain level.

Another results of bounds strengthening is given by the case $l_j = u_j$ for some variable x_j. In this case, the variable is determined to have the value l_j in every feasible solution to the problem. Hence this value can be used in all constraints, and the variable can be removed from the problem.

Note that from some inequality i we obtain either a new lower bound (candidate) on variable x_j or a new upper bound (candidate) on the same variable, whereas from equality constraints we get upper and lower bounds (candidates) at the same time. In practice, one would not split an equality constraint into two seperate inequalities and sequentially perform presolving on them, but in one single step. For an equality constraint

$$\text{(EQ)} \quad a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i \tag{A.8}$$

we write `boundsStrengthening`(x_j; (EQ)) for the algorithm that tries to obtain better lower and upper bounds on x_j from (EQ). If $a_{ij} > 0$ then this is carried out by applying `upperBoundStrengthening`(x_j; (IEQ1)) on

$$\text{(IEQ1)} \quad a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i, \tag{A.9}$$

and `lowerBoundStrengthening`(x_j; (IEQ2)) on

$$\text{(IEQ2)} \quad a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i. \tag{A.10}$$

Otherwise, if $a_{ij} < 0$, then we strengthen the lower bound with (IEQ1) and the upper bounds with (IEQ2).

When implementing the bounds strengthening algorithm in computer code, one is faced with the limitations of a finite precision floating point arithmetic. Hence one has to be careful with effects due to numerical instabilities, which in particular occur when presolving equality constraints. As we will see below, an instance of our model can be falsely declared as infeasible by the bounds strengthening procedure. This can be avoided by

updating the bounds only if they improve the existing ones by at least a value $\varepsilon := 10^{-12}$. In a double precision floating point arithmetic the ULP (unit in the last place) for numbers with an exponent of 0 is about 10^{-16} (or precisely 2^{-23}), hence our choice of ε leaves room of 4 decimal digits for computations.

B Further Computational Results

In order to demonstrate that we are able to solve also large scale networks with our methods, we consider networks of $k \times 6$ interior vertices, where k runs from 1 to 101, see Figure 7. We measure the presolve time consumed by Cplex for the pure MIP, the L1-presolved MIP, and the L3-presolved MIP.

For the inflow arcs $i_1, \ldots, i_3$, we choose maximal processing rates of $\mu^i = 100$ ($i \in \{i_1, \ldots, i_3\}$) and for the remaining arcs $\mu^e = 1$. In contrast to the 4×4 block network considered above, we choose velocities $v^e = 1$ for all arcs. The processor lengths are set to $L^e = 1$. We choose a time horizon of $T = 8$ time units and a timestep size of $\Delta t = 0.25$. The inflow profile on each network inflow arc $i \in \{i_1, \ldots, i_3\}$ is given by $f_i^{in}(t) = 5$ for $t \leq 2$ and $f_i^{in}(t) = 0$ otherwise.

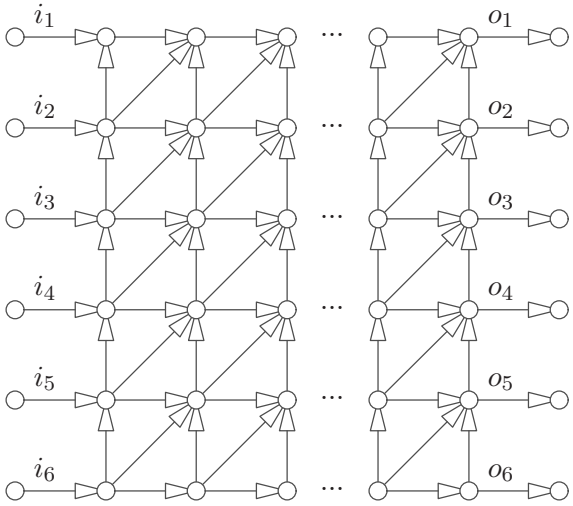

Figure 7: $k \times 6$ block network

In Table 6, we compare the overall computing times for preprocessing in dependence of the network size k. Columns 1, 5, and 8 show the total preprocessing times in seconds for each run. These are the sum of the computing time of scmip-presolve in the particular levels and Cplex presolve time for

the remaining MIP (shown in columns 3 and 4 for L1, and in columns 6 and 7 for L3).

k	Cplex pre	L1	Cplex pre	L1 total	L3	Cplex pre	L3 total
1	0.01	0.00	0.01	0.01	0.00	0.01	0.01
11	0.91	0.05	0.60	0.65	0.04	0.09	0.13
21	1.34	0.10	0.63	0.73	0.08	0.10	0.18
31	1.64	0.14	0.64	0.78	0.11	0.10	0.21
41	1.88	0.19	0.64	0.83	0.15	0.10	0.25
51	2.35	0.24	0.66	0.90	0.19	0.09	0.28
61	2.83	0.29	0.66	0.95	0.22	0.10	0.32
71	3.02	0.35	0.70	1.05	0.25	0.11	0.36
81	3.48	0.39	0.70	1.09	0.29	0.11	0.40
91	3.76	0.41	0.73	1.14	0.33	0.11	0.44
101	4.05	0.47	0.70	1.17	0.36	0.10	0.46

Table 6: Total preprocessing times for MIPs corresponding to $k \times 6$ block networks

Similarly to the previous example we observe that scmip-presolve is an efficient technique to shorten the time spent for preprocessing the MIP and strongly improves the solution time of the remaining MIP. It clearly outperforms Cplex black-box preprocessing. Further, the more sophisticated presolve levels are nearly as expensive as the simpler levels but reduce the time necessary for the solution of the remaining MIP.