Satisfiability Modulo Theories for Process Systems Engineering

Miten Mistry, Andrea Callia D'Iddio, Michael Huth, Ruth Misener*

Department of Computing; Imperial College London; South Kensington SW7 2AZ; UK

Abstract

Process systems engineers have long-recognized the importance of both logic and optimization for automated decision-making. But modern challenges in process systems engineering could strongly benefit from methodological contributions in computer science. In particular, we propose *satisfiability modulo theories* (SMT) for process systems engineering applications. We motivate SMT using a series of test beds and demonstrate the applicability of SMT algorithms and implementations on (i) planning and scheduling, (ii) applications with many numeric scales, (iii) MINLP solvers.

Keywords: Satisfiability Modulo Theories, Mixed-Integer Optimization, Generalized Disjunctive Programming, Mixed Logical-Linear Programming

1. Introduction

A major challenge in process systems engineering (PSE) is integrating: (i) long-term strategic planning decisions, (ii) medium-term tactical planning, and (iii) short-term scheduling (Maravelias and Sung 2009). One way to address these multi-scale optimization problems is by integrating logic and optimization (Hooker and Ottoson 2003). For example, a scheduling problem may have two levels: (i) assigning orders to machines and (ii) sequencing orders on each machine (Jain and Grossmann 2001). In a minimum cost model, assigning orders to machines is a mixed-integer optimization problem, sequencing orders is a constraint satisfaction problem. The entire problem may be reformulated as either an optimization or logic problem, but this misses the chance to use optimization and logic synergistically (Trespalacios and Grossmann 2014).

Process systems engineers have long-recognized the importance of both logic and optimization for automated decision-making. Early work on disjunctive programming is motivated by (i) the practical need to naturally model logical conditions such as dichotomies and implications and (ii) the

*r.misener@imperial.ac.uk; Tel: +44 (0) 20759 48315

theoretical insight gained from novel structural characterizations (Balas 1979). Contributions highlighting the importance of both logic and optimization have diverse applications, e.g. spatial layout (Sawaya and Grossmann 2005), modeling contracts in supply chain optimization (Park et al. 2006, Rodriguez and Vecchietti 2009), and manufacturing systems (Fattahi et al. 2014).

While process systems engineers have been developing methods at the interface of logic and optimization, the computer science community has also been developing hybrid logic/optimization approaches. Typical computer science applications requiring both logic and optimization are operating system scheduling and motion planning in robotics (Aminof et al. 2011, Raman et al. 2013, Beaumont et al. 2015). The difference in application domains between process systems engineering and computer science have led to a divergence in mathematical developments.

But modern challenges in process systems engineering could strongly benefit from methodological contributions in computer science. In particular, we propose *satisfiability modulo theories* (SMT) for process systems engineering applications. We motivate SMT using a series of test beds and demonstrate the applicability of SMT algorithms and implementations on (i) planning and scheduling, (ii) applications with many numeric scales, (iii) MINLP solvers.

Section 2 reviews background in both optimization and logic. Section 3 discusses existing optimization/logic hybrids. Section 4 describes 3 domains where SMT is highly applicable to PSE: (§4.1) Planning and scheduling, (§4.2) MINLP with many numeric scales, (§4.3) SMT-based MINLP solvers. Table 1 summarizes our position that SMT has complementary strengths and weaknesses with respect to mixed-integer nonlinear optimization (MINLP). We propose SMT as a methodology to address several challenges in process systems engineering. We also provide a *needs analysis* identifying the SMT development required by the process systems engineering community.

Parts of this paper have been previously published. Section 4.1 extends our recent conference paper (Mistry and Misener 2017). Lundbæk et al. (2016) solve in MINLP system for governed Proof of Work blockchain systems, which we sketch in Section 4.2. Callia D'Iddio and Huth (2017) describe the Section 4.3 `ManyOpt` tool in further detail. The purpose and novelty of this paper is to demonstrate the broad applicability of SMT to PSE.

2. Definitions & Background

Section 2.1 fixes variable notation. Sections 2.2 – 2.4 review mixed-integer nonlinear optimization (MINLP), propositional satisfiability (SAT), and sat-

Table 1: Complementary strengths in SMT/MINLP inspire applying SMT to PSE

	SMT	**MINLP**	
Traditional Community	Computer science	Engineering	Division in developments stemming from divergent applications
Deductive Reasoning	**Strong**	**Limited**	GDP less flexible than SMT
Nonlinear Functions	**Limited**	**Strong**	Transcendental functions in MINLP
Optimizing Objective	**Weak**	**Strong**	Tightly integrated in MINLP
Propositional Satisfiability	**Strong**	**Weak**	Logical propositions not typical in MINLP
Warm Starting	**Strong**	**Weak**	Would strongly benefit MINLP
Scalability	**Limited**	**Limited**	SMT : limited for nonlinear functions MINLP: limited for large problems[†]
Typical Applications	**SMT :** Software verification; Scheduling **MINLP:** Energy systems design; Biomedical engineering		

[†] For some problems, MINLP can reliably address 10^3 variables/constraints; but 10^2 variables/constraints are more typical for general problem classes

isfiability modulo theories (SMT), respectively. Figure 1 diagrams how solving MINLP problems as a series of mixed integer linear optimization problems is analogous to solving SAT problems as a series of SAT problems.

2.1. Notation for Logic & Optimization

Logic and optimization requires three variable types: (i) continuous variables $\boldsymbol{x} \in \mathbb{R}^{n_C}$, (ii) integer variables $\boldsymbol{y} \in \mathbb{Z}^{n_I}$, and (iii) Boolean (propositional) variables $\boldsymbol{Y} \in \{\mathit{True}, \mathit{False}\}^{n_B}$. Propositional connectives are: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (if...then) and $\leftrightarrow$ (if and only if). The shorthand $\underline{\vee}_{i \in I} Y_i$ represents that exactly one of the propositional variables Y_i is true.

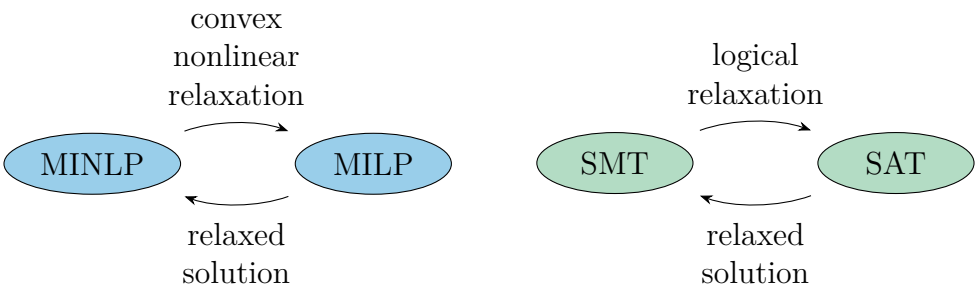

Figure 1: Mixed-integer nonlinear optimization problems (MINLP) may be solved as a series of mixed integer linear optimization problems (MILP). Satisfiability modulo theories problems (SMT) may be solved as a series of propositional satisfiability problems (SAT).

2.2. Mixed-Integer Nonlinear Optimization

Traditional mathematical optimization formulations only incorporate the continuous and integer variables. MINLP is defined:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & b_i^{\text{LO}} \leq f_i(\boldsymbol{x}) \leq b_i^{\text{UP}} && \forall\, i \in \mathcal{M} := \{1, \ldots, M\} \\ & x_j^{\text{LO}} \leq \quad x_j \leq x_j^{\text{UP}} && \forall\, j \in \mathcal{N} := \{1, \ldots, N\} \\ & x_j \in \mathbb{Z} && \forall\, j \in \mathcal{I} \subseteq \mathcal{N} \end{aligned} \tag{MINLP}$$

where $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{I}$ represent sets of constraints, variables, and discrete variables, respectively. The objective and constraints are functions $f_i : \mathbb{R}^N \mapsto \mathbb{R}\ \forall\ i \in \{0, \ldots, M\}$. Parameters $b_i^{\text{LO}} \in \mathbb{R} \cup \{-\infty\}$ and $b_i^{\text{UP}} \in \mathbb{R} \cup \{+\infty\}$ bound the set of constraints $\mathcal{M}$; parameters $x_j^{\text{LO}} \in \mathbb{R} \cup \{-\infty\}$ and $x_j^{\text{UP}} \in \mathbb{R} \cup \{+\infty\}$ bound the set of variables $\mathcal{N}$. We assume that it is possible to infer finite bounds on the variables $\boldsymbol{x}$ and that the image of f_i is finite on $\boldsymbol{x}$.

2.3. Propositional Satisfiability

Traditional propositional satisfiability only incorporates Boolean variables. SAT is defined:

Given a propositional formula φ built from the variables Y_i, is there a truth assignment that satisfies φ.

There exist efficient, satisfiability-preserving transformations from any propositional formula φ to conjunctive normal form (CNF), so SAT solvers typically assume that φ is written in CNF. In CNF: literals, i.e. propositional variables Y_i or their negation $\neg Y_i$, form clauses P_j, i.e. disjunctions ($\vee$) of literals. The final propositional formula $\varphi = \bigwedge_{j=1}^{m} P_j$ is a conjunction of clauses.

SAT applications include: planning (Kautz and Selman 1992), model checking (Biere et al. 1999) and scheduling (Zhang 2002). Although SAT is $\mathcal{NP}$-complete (Cook 1971) and the worst-case complexity is exponential, modern SAT solvers can handle problems with hundreds of thousands of variables (Malik and Zhang 2009).

Most SAT solvers use the Davis-Putnam-Logemann-Loveland (DPLL) search algorithm (Davis and Putnam 1960, Davis et al. 1962). DPLL fixes variable Y_i assignments, i.e. truth assignments, using a tree-based branching approach. DPLL propagates truth assignments to all clauses P_j. Propagating truth values may allow DPLL to assign further variables a truth value. If DPLL finds that a partial assignment is *unsatisfiable*, i.e. cannot satisfy φ, then the algorithm backtracks and assigns a different value to one of the variables. DPLL continues until it either: (i) finds a combination of truth values for Y_i satisfying φ or (ii) proves the formula φ is unsatisfiable. DPLL also has functionality supporting warm starts.

SAT solving techniques include (Biere et al. 2009): (i) Boolean constraint propagation, where the current fixed variable set implies variable assignments, (ii) resolution, where sets of clauses derive additional clauses, (iii) and Conflict Driven Clause Learning, where an unsatisfiable result derives extra clauses pruning the search tree (Davis and Putnam 1960, Davis et al. 1962, Silva and Sakallah 1996). SAT solving methods are highly applicable to optimization (Hooker and Osorio 1999, Achterberg 2007a).

2.4. Satisfiability Modulo Theories

SMT incorporates continuous, integer, and Boolean variables to assess constraint set satisfiability by separating truth value assignment from the correctness reasoning with respect to a *theory*. SMT consists of: (i) a SAT solver and (ii) a theory solver for a theory of our choice (de Moura and Bjørner 2008a). The SMT approach to constraint satisfaction uses powerful SAT solving to derive a, potentially smaller, set of constraints to assess theory satisfiability. A background theory is a set of axioms and symbols, e.g. the theory of arithmetic. An SMT solver consists of a SAT solver and a theory solver. The idea is to leverage the strength and robustness of modern SAT solvers to search for a feasible solution. The modeling framework exposed by SMT allows for Boolean variables to be used with background theory variables, e.g $Y \rightarrow (x \geq 0)$ where x is continuous and Y is Boolean, so SMT is a natural choice when logical decisions form a part of the modeled system.

SMT research dates back to the 1970s with early work on decision proce-

dures (Nelson and Oppen 1979, 1980, Shostak 1979, 1982). Available SMT theories include: Equality with Uninterpreted functions ($\mathcal{EUF}$), linear arithmetic $\mathcal{LA}$, and arrays $\mathcal{AR}$ (Biere et al. 1999). DPLL(T) generalizes DPLL (Ganzinger et al. 2004). SMT is primarily applied in program verification and formal methods, but it also has scheduling and planning applications (Bjørner and De Moura 2011).

SMT assesses the satisfiability of a model and, if the model is satisfiable, the SMT solver returns a witness. If the model is unsatisfiable the SMT solver can return an unsatisfiable core, a mutually unsatisfiable subset of model constraints. An unsatisfiable core is a useful tool when addressing why a model does not behave how we expect or to understand why our model fails. Commonly used SMT solvers include Z3 (de Moura and Bjørner 2008a) and MathSAT (Cimatti et al. 2013).

Example 1. Suppose that we wish to satisfy Eq. (1). Equation (1) combines SAT and the theory of real arithmetic.

$$(x_1 \leq 1) \wedge (x_2 \leq 2) \wedge ((x_1 \geq 5) \vee (x_3 \leq 3)) \wedge (x_1 + x_2 + x_3 \geq 10) \tag{1}$$

For Eq. (1), SMT leverages a SAT solver by replacing each inequality with auxiliary propositional variables, e.g. $Y_1 = (x_1 \leq 1)$, and assessing propositional satisfiability of the resulting formula:

$$Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4) \wedge Y_5. \tag{2}$$

The SAT solver returns an assignment satisfying Eq. (2), e.g. $Y_i = True$, $\forall\, i$. Then, the real arithmetic theory solver checks the propositional variable meaning. Here, the theory solver deduces that the assignment is incorrect because we cannot have both $Y_1 = (x_1 \leq 1) = True$ and $Y_3 = (x_1 \geq 5) = True$. The theory solver encodes additional propositional clauses, e.g. $(\neg Y_1 \vee \neg Y_3)$, augments Eq. (2), and passes Eq. (3) to the SAT solver:

$$Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4) \wedge Y_5 \wedge (\neg Y_1 \vee \neg Y_3). \tag{3}$$

SMT iterates between the SAT and theory solvers until the algorithm terminates, in this case with the point $x_1 = 5, x_2 = 2, x_3 = 3$. □

Example 1 suggests that the SAT and theory solvers are disjoint, but the most efficient and stable SMT tools integrate the two components (Sebastiani 2007). Interaction between the theory solver and partial SAT solutions allow

the theory solver to identify unsatisfiability in a partial assignment.

The efficacy of an SMT solver depends on the theory solver quality since a propositional encoding has to be created for the SAT solver. If the encoding is weak and the theory solver cannot strengthen it effectively, the SMT solver will, in worst case, enumerate all propositional solutions.

3. Logic/Optimization Hybrids

This section reviews mathematical modeling approaches combining optimization and logic: (§3.1) Disjunctive programming, (§3.2) Generalized disjunctive programming, and (§3.3) Mixed logical-linear programming. We also discuss prior work on optimization methods using SMT (§3.4) and logic-based Benders decomposition, a commonly-used solution protocol (§3.5).

3.1. Disjunctive Programming

Balas developed Disjunctive Programming in the 1970s (Balas 1974, 1975, 1977, 1979). A disjunctive program is given by a linear objective and the disjunction of systems of linear constraints:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \bigvee_{i=1}^{k} \boldsymbol{A}_i \boldsymbol{x} \leq \boldsymbol{b}_i \\ & \boldsymbol{x} \in \mathbb{R}^n, \end{aligned} \tag{4}$$

with parameters $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A}_i \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b}_i \in \mathbb{R}^m \; \forall i = 1, \ldots, k$. The disjunctive program in Eq. (4) may be written equivalently as a mixed-integer linear program (MILP) by introducing binary variables $\boldsymbol{y}$ and auxiliary continuous variables $\boldsymbol{x}_i$ (Hooker 2002):

$$\begin{aligned} \min_{\boldsymbol{x}, \boldsymbol{x}_i, \boldsymbol{y}} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}_i \boldsymbol{x}_{\boldsymbol{i}} \leq \boldsymbol{b}_i y_i, && \forall i = 1, \ldots, k \\ & \boldsymbol{x} = \boldsymbol{x}_1 + \ldots + \boldsymbol{x}_k \\ & \boldsymbol{x} \in \mathbb{R}^n \\ & \boldsymbol{x}_i \in \mathbb{R}^n && \forall i = 1, \ldots, k \\ & \boldsymbol{y} \in \{\, 0, 1 \,\}^k . \end{aligned} \tag{5}$$

Disjunctive programming allows model developers to write certain problems more concisely and/or more meaningfully. For example, selecting one element i out of a set $i \in \{\, 1, \ldots, n \,\}$, i.e. set partitioning, is a common

constraint in process systems engineering:

$$\sum_{i=1}^{n} y_i = 1 \text{ where } y_i \in \{\, 0, 1 \,\}. \tag{6}$$

As a disjunctive constraint, Eq. (6) is easily identified as a selection constraint:

$$\bigvee_{i=1}^{n} \left(y_i = 1 \wedge \bigwedge_{i \neq j} y_j = 0 \right).$$

Most approaches for solving disjunctive programs replace propositional variables with Binary variables. In the resulting model, $y_i = 1$ implies a set of active constraints and $y_i = 0$ implies associated inactive constraints.

3.2. Generalized Disjunctive Programming

Raman and Grossmann (1994) developed Generalized Disjunctive Programming (GDP), an extension of disjunctive programming incorporating nonlinear functions. GDP offers a natural, intuitive framework to model applications with logical dependencies, e.g. (i) job shop scheduling and (ii) process network superstructure design (Türkay and Grossmann 1996, Lee and Grossmann 2000). A GDP is formulated (Grossmann and Ruiz 2012):

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{Y},\boldsymbol{c}} \quad & Z = f(\boldsymbol{x}) + \sum_{k \in K} c_k \\
\text{s.t.} \quad & g(\boldsymbol{x}) \leq \mathbf{0} \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ r_{ik}(\boldsymbol{x}) \leq \mathbf{0} \\ c_k = Y_{ik} \end{bmatrix} \qquad k \in K \\
& \Omega(\boldsymbol{Y}) = \mathit{True} \\
& \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u} \\
& \boldsymbol{x} \in \mathbb{R}^n, c_k \in \mathbb{R}, Y_{ik} \in \{\mathit{True}, \mathit{False}\}
\end{aligned} \tag{7}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, and $r_{ik} : \mathbb{R}^n \to \mathbb{R}^p$ may be nonlinear nonconvex functions. Each disjunction $k \in K$ is composed of conjunctions $i \in D_k$, i.e. the Eq. (7) square brackets. Each conjunction has a boolean variable Y_{ik}, inequality $r_{ik}(\boldsymbol{x}) \leq \mathbf{0}$, and cost variable c_k. If $Y_{ik} = \mathit{True}$ then the formulation enforces both $r_{ik}(\boldsymbol{x}) \leq \mathbf{0}$ and $c_k = Y_{ik}$. Otherwise they are ignored. Propositional formula $\Omega(\boldsymbol{Y}) = \mathit{True}$ typically contains an exclusive disjunction assumption $\underset{i \in D_k}{\veebar} Y_{ik}$ for each $k \in K$. The exclusive disjunction

assumption ensures correctness by restricting c_k to a single Y_{ik}.

To leverage existing MINLP solvers and relaxation techniques, GDP models are often reformulated as MINLPs (Ruiz et al. 2012). A GDP model may be reformulated as using either the big-M (Nemhauser and Wolsey 1988) or Hull relaxation (Lee and Grossmann 2000) reformulation. Both reformulations replace propositional variables Y_{ik} with binary variables y_{ik} where $Y_{ik} = \mathit{True} \leftrightarrow y_{ik} = 1$. The reformulations also substitute:

$$\sum_{i \in D_k} y_{ik} = 1, \qquad \forall k \in K \tag{8}$$

$$\boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{a} \tag{9}$$

for $\Omega(\boldsymbol{Y}) = \mathit{True}$, where Eq. (8) corresponds to $\veebar_{i \in D_k} Y_{ik}$ for each $k \in K$. The big-M reformulation replaces the Eq. (7) disjunction with:

$$r_{ik}(\boldsymbol{x}) \leq \boldsymbol{M_{ik}}(1 - y_{ik}), \qquad \forall k \in K,\, i \in D_k,$$

whereas the hull reformulation uses the constraints:

$$\begin{aligned} y_{ik} r_{ik}(\boldsymbol{\nu_{ik}}/y_{ik}) &\leq \boldsymbol{0}, && \forall k \in K, i \in D_k \\ \boldsymbol{0} \leq \boldsymbol{\nu_{ik}} &\leq y_{ik}\boldsymbol{x_u}, && \forall k \in K, i \in D_k \\ \boldsymbol{\nu_{ik}} &\in \mathbb{R}^n. \end{aligned}$$

The hull relaxation is at least as tight as big-M but incorporates additional continuous variables ($\boldsymbol{\nu_{ik}}$). There has been significant research into appropriate algorithms for interesting classes of GDP's (Türkay and Grossmann 1996, Lee and Grossmann 2000, Vecchietti et al. 2003, Ruiz et al. 2012, Trespalacios and Grossmann 2016).

3.3. Mixed Logical-Linear Programming

Mixed Logical/Linear Programming (MLLP) is formulated (Hooker and Osorio 1999):

$$\begin{aligned} &\min \;\; \boldsymbol{c}^\top \boldsymbol{x} \\ &\text{s.t.} \quad p_j(\boldsymbol{Y}, \boldsymbol{y}) \rightarrow (\boldsymbol{A_j}\boldsymbol{x} \geq \boldsymbol{a_j}),\; j \in \mathcal{J} \mid q_i(\boldsymbol{Y}, \boldsymbol{y}),\; i \in \mathcal{I}\,. \end{aligned} \tag{10}$$

Eq. (10) splits the constraints into continuous and logical parts, on the left and right of the bar, respectively. The logical part consists of formulas $q_i(\boldsymbol{Y}, \boldsymbol{y})$ where $\boldsymbol{Y} \in \{\mathit{True}, \mathit{False}\}^{n_B}$ and $\boldsymbol{y} \in \mathbb{Z}^{n_I}$. The continuous part

is formulated as logical implications such that if $p_j(\boldsymbol{Y}, \boldsymbol{y})$ is true then the constraint $\boldsymbol{A_j x} \leq \boldsymbol{a_j}$ is imposed.

MLLP models are solved by branching on the propositional variables $\boldsymbol{Y}$ and discrete variables $\boldsymbol{y}$. As branching takes place, MLLP progressively strengthens the relaxation by enforcing constraints $\boldsymbol{A_j x} \geq \boldsymbol{a_j}$ if the corresponding antecedent p_j is true. Since the logical part is separated from the continuous part, MLLP enables propositional satisfiability algorithms to derive further logical constraints and prune the search space.

An MLLP model may look different from the equivalent MILP. For many cases, e.g. where MILP binary variables model existence or assignment, MLLP may result in an easier-to-comprehend model with fewer variables. MLLP may be extended to models with nonlinear constraints, i.e. to Mixed Logical/Nonlinear Programming (MLNLP) (Türkay and Grossmann 1996, Bollapragada et al. 2001, Bemporad and Giorgetti 2004, 2006, Carbonneau et al. 2011, 2012).

3.4. Optimization Methods based on Satisfiability Modulo Theories

Even advances such as GDP (Grossmann and Ruiz 2012) cannot compete with the expressiveness of constraints written in SMT solvers. SMT solvers support logical theories and dependencies, do precise arithmetic, and enable incremental solving. But SMT solvers may have performance issues with division, reasoning over integers, and only limited support for transcendental functions (de Moura and Passmore 2013). MINLP tools support the transcendental functions well and scale will for mixed integer reasoning (Carvajal et al. 2014), but MINLP solvers cannot solve incrementally and have limited support for logical constraints and are sensitive to rounding errors. Table 1 summarizes distinctions between SMT and MINLP.

Two of the most prominent SMT-based optimization methods are optimization modulo theories and integer linear programming modulo theories.

Optimization Modulo Theories integrates optimization and SMT with respect to the theory of linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$) (Sebastiani and Tomasi 2015). The models are equivalent to MILP problems. Sebastiani and Tomasi (2015) consider different approaches to solve the MILP problems, e.g. offline and inline schemas with linear, binary or adaptive search. Sebastiani and Tomasi (2015) compare optimization modulo theories versus linear GDP using both a convex hull and a big-M relaxation. The comparisons, based on strip packing and job shop scheduling case studies (Sawaya and Grossmann 2005), demonstrate that an SMT solver may

be used for optimization.

Integer Linear Programming Modulo Theories is an optimization framework where MILP, rather than SAT, is leveraged as the efficient solver (Manolios and Papavasileiou 2013). Integer linear programming modulo theories is an optimization framework in which difference logic is used to communicate with the solver. Manolios and Papavasileiou (2013) implement their framework as a constraint handler for the MILP solver SCIP (Achterberg 2007b, 2009). A weakness of a MILP-based approach is that floating point calculations may lead to wrong answers. Errors based on floating point do not happen in SMT because all formulae evaluate to true or false only.

3.5. Logic-Based Benders Decomposition

Hybrid optimization/logic approaches have been developed combining mixed-integer linear programming (MILP) and constraint programming (CP), e.g. Jain and Grossmann (2001), Maravelias and Grossmann (2004), Li and Womer (2008), Sitek (2014), or multiple levels of MILP, e.g. Maravelias (2006). The hybrid formulations usually use logic-based Benders decomposition (LBBD) (Hooker and Ottoson 2003), a generalization of Benders decomposition (Benders 1962). The principles of Benders decomposition remain: we have a master problem and a subproblem which generates cuts if the solution from the master problem is infeasible. The difference is that LBBD requires a logic proof deriving an objective bound. Other hybrid algorithms use branch-and-check (Thorsteinsson 2001) or Lagrangian decomposition (Papageorgiou and Trespalacios 2016).

Hybrid MILP/CP methods are typically applied to scheduling and its variants (Sitek 2014). This is reasonable: CP is very good at assessing scheduling feasibility. The problem with hybrid MILP/CP is that, if the application does not have a suitable CP constraint, a hybrid method may be poor since bespoke CP constraints take full advantage of very specific mathematical structures. This manuscript evaluates satisfiability modulo theories as an alternative to CP in the hybrid scheme.

4. Satisfiability Modulo Theories for Process Systems Engineering

This section demonstrates the applicability of SMT algorithms and implementations with respect to: (i) planning and scheduling, (ii) applications with many numeric scales, (iii) MINLP solvers.

4.1. Planning and Scheduling

See Table 2 for descriptions of the sets, parameters and variables mentioned in the formulations.

4.1.1. MILP Models

The MILP formulation follows the Hooker (2007) discrete time model; a continuous time formulation (Türkay and Grossmann 1996) is harder to solve for the models considered in this manuscript. The minimum cost model is:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{t\in\mathcal{T}} F_{ij}y_{ijt} \tag{11}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} y_{ijt} = 1, \qquad \forall j \in \mathcal{J} \tag{12}$$

$$\sum_{j\in\mathcal{J}}\sum_{t'\in\mathcal{T}_{ijt}} c_{ij}y_{ijt'} \leq C_i, \qquad \forall i \in \mathcal{I}, t \in \mathcal{T} \tag{13}$$

$$y_{ijt} = 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}, \quad t < r_j \text{ or } t > d_j - p_{ij} \tag{14}$$

$$y_{ijt} \in \{0,1\}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}. \tag{15}$$

Equation (12) requires that a task is assigned to a single machine and only starts once. Equation (13) characterises the resource constraints using set $\mathcal{T}_{ijt}$. Equation (14) limits the time windows for a given task with r_j and d_j.

The makespan is the total schedule length. The minimum makespan model is similar to minimum cost, but minimum makespan requires an additional variable $M \geq 0$, a new constraint bounding the makespan M from below by the local makespan of each task:

$$M \geq \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}(t + p_{ij})y_{ijt}, \quad \forall j \in \mathcal{J}, \tag{16}$$

and a different objective:

$$\min \; M. \tag{17}$$

4.1.2. Logical Models

As in the MILP case, a discrete logical formulation can be very large when set $\mathcal{T}$ is large. But, for an SMT solver, a discrete formulation loses information and is therefore less favorable. Some inherent task properties are the release, due and processing times which are not directly present in the model. They would have to be inferred from the constraints given by

Table 2: Model symbols (Hooker 2007).

Name	Description
Sets	
$\mathcal{I} = \{1, \ldots, m\}$	Facilities
$\mathcal{J} = \{1, \ldots, n\}$	Tasks
$\mathcal{T} = \{1, \ldots, p\}$	Discrete time points
$\mathcal{T}_{ijt} = \{t' \mid t - p_{ij} < t' \leq t\}$	Start times: j is in progress at t on i
$\mathcal{H} = \{1, \ldots, H-1\}$	Iteration indices; assume that we are at iteration H in the hybrid model
$\mathcal{J}_{hi} \subseteq \mathcal{J}$	Local assignment of tasks to i in iteration h in hybrid model
$\bar{\mathcal{J}}_{hi} \subseteq \mathcal{J}_{hi}$	Tasks mutually responsible for infeasibility/local optimality in hybrid model
Parameters	
p_{ij}	Processing time of j on i
c_{ij}	Resource consumption of j on i
C_i	Resource capacity on i
r_j	Release time of j
d_j	Due time of j
F_{ij}	Cost of assigning j to i (min cost only)
Variables	
y_{ijt}	Assign j to i starting at t (MILP, binary)
y_{ij}	Assign j to i (hybrid, binary)
Y_{ij}	Assign j to i (SMT, Boolean)
s_j	Start time of j (SMT & hybrid, continuous)
c'_j	Resource 'position' of j (SMT, continuous)
M	Makespan (min makespan only, continuous)

Eqs. (13) and (14), and the set $\mathcal{T}_{ijt}$ and may weaken the SMT theory solver. A continuous time minimum cost model is:

$$\min \sum_{j \in \mathcal{J}} f_j \tag{18}$$

$$\text{s.t. } s_j \geq r_j, \qquad \forall j \in \mathcal{J} \tag{19}$$

$$Y_{ij} \rightarrow (s_j \leq d_j - p_{ij}), \qquad \forall i \in \mathcal{I}, j \in \mathcal{J} \tag{20}$$

$$\bigvee_{i \in \mathcal{I}} \left(Y_{ij} \wedge \bigwedge_{i' \neq i} \neg Y_{i'j} \right), \qquad \forall j \in \mathcal{J} \tag{21}$$

$$\bigwedge_{j \in \mathcal{J}'} Y_{ij} \rightarrow \left(\left(\sum_{j \in \mathcal{J}'} c_{ij} \leq C_i \right) \vee \bigvee_{\substack{j, j' \in \mathcal{J}' \\ j' \neq j}} (s_j + p_{ij} \leq s_{j'}) \right), \qquad \forall i \in \mathcal{I}, \mathcal{J}' \in \mathbb{P}(\mathcal{J}) \setminus \emptyset \tag{22}$$

$$Y_{ij} \rightarrow (f_j = F_{ij}), \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{23}$$

The minimum makespan model is similar to the minimum cost model, but does not have variables f_j or parameters F_{ij}. Additional variable $M \geq 0$ represents the makespan and the minimum cost objective Eq. (18) is replaced with the makespan objective:

$$\min \ M. \tag{24}$$

We also introduce the constraint which implies that if task j is assigned to facility i then the makespan must be greater than or equal to the task's completion time:

$$Y_{ij} \rightarrow (M \geq s_j + p_{ij}), \quad \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{25}$$

This logical formulation is flawed because the total number of Eq. (22) constraints is exponential in the number of tasks and model building time may be a bottleneck. Also, some of these constraints may provide redundant information, e.g. if a set of tasks cannot be scheduled on the same machine, any superset of these tasks cannot be scheduled.

We improve the logical formulation by interpreting time and resource consumption as separate dimensions and thereby convert the planning and scheduling problem to a generalized two dimensional bin packing problem (Garey et al. 1976). The difference between this conversion and 2BP is that the items (tasks) may not have the same height and width when they are placed in different bins (machines); the items are further constrained on one dimension (release and due times). We convert Eq. (22) by modeling task j as an item with height c_{ij} and width p_{ij} when placed on machine i. New variable $c'_j \geq 0$ with upper bounding constraint:

$$Y_{ij} \rightarrow \left(c'_j \leq C_i - c_{ij}\right), \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \tag{26}$$

accounts for the new dimension and Eq. (22) is replaced with:

$$(Y_{ij} \wedge Y_{ij'}) \rightarrow \begin{cases} (s_j + p_{ij} \leq s_{j'}) \vee (s_{j'} + p_{ij'} \leq s_j) \\ \vee (c'_j + c_{ij} \leq c'_{j'}) \vee (c'_{j'} + c_{ij'} \leq c_j) \end{cases} \quad \forall i \in \mathcal{I}, j, j' \in \mathcal{J}, j < j'. \tag{27}$$

With this replacement, there are a quadratic rather than exponential number of constraints. This formulation is more meaningful when considering the

SMT unsatisfiable core since the tasks are related pairwise, e.g. an unsatisfiable core consisting of pairs $\{(1,2),(1,3),(2,3)\}$ would mean that these three tasks are unsatisfiable when assigned to the same machine whereas the exponential can return any constraint that corresponds to a superset of $\{1,2,3\}$. The algorithm we implement for pure SMT bounds the objective seeking feasible solutions hence we do not get an unsatisfiable core, but for the hybrid models we can get unsatisfiable cores from which we derive cuts.

4.1.3. Novel Hybrid Model combining SMT and MILP

The hybrid MILP/SMT strategy splits the problem into: the master problem and the subproblem. The master problem optimizes a less constrained problem; this optimization solution is checked in the subproblem for correctness. If feasible then optimality is achieved, otherwise a cut is derived to reject the current incumbent (and possibly others) from the master problem and the process is repeated.

For planning and scheduling, we adapt the Hooker (2007) LBBD method developed for hybrid MILP/CP. The formulations, cuts and relaxations listed below were all formulated by Hooker (2007), we describe the differences in adapting the approach to hybrid MILP/SMT. The problems are solved with a master problem (MILP) that assigns tasks to facilities and a subproblem (SMT) that assesses the feasibility (for minimum cost) or local optimality (for minimum makespan). The solution process iterates by solving the master problem for an assignment and then, having fixed these assignments, solving the subproblem and repeating until the termination criteria is satisfied. On each iteration, the subproblem derives Benders cuts to be added to the master problem, these cuts prevent the current assignment from being reassessed and are typically strong enough to prune large amounts of the search space. Hooker (2007) states that 'experience shows that it is important to include a relaxation of the subproblem'; we use the same relaxations when assessing the hybrid MILP/SMT approach.

Only the master problems are formulated here, the subproblems consist of constraints associated with the start times and resource limits where the Boolean variables are fixed according to the master problem assignment. For minimizing cost, the master problem is:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} F_{ij}y_{ij} \tag{28}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}} y_{ij} = 1, \qquad \forall j \in \mathcal{J} \tag{29}$$

$$\sum_{j \in \bar{\mathcal{J}}_{hi}} (1 - y_{ij}) \geq 1, \qquad \forall i \in \mathcal{I}_h, h \in \mathcal{H}, \tag{30}$$

$$\sum_{j \in J(t_1,t_2)} p_{ij} c_{ij} y_{ij} \leq C_i(t_2 - t_1), \tag{31}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{32}$$

The binary variables y_{ij} represent task j being assigned to machine i. The master problem only contains the assignment variables and costs; task start times are in the subproblem. The y_{ij} values in a master problem solution have a one-to-one correspondence with subproblem variables Y_{ij}, i.e. $y_{ij} = 1 \iff Y_{ij} = \text{True}$. Equation (30) are the Benders cuts added on each iteration. Note that the task set is $\bar{\mathcal{J}}_{hi}$ which is derived by greedily filtering tasks from $\mathcal{J}_{hi}$ while keeping the local assignment infeasible. The SMT unsatisfiable core may not necessarily return all unsatisfiable facilities under the local assignment (it is more likely that the core will only be associated with a single machine), but the core is mutually unsatisfiable therefore it will satisfy the properties we want in $\bar{\mathcal{J}}_{hi}$. Equation (31) is the subproblem relaxation where $J(t_1, t_2)$ contains all tasks that are released at or after t_1 and are due at or before t_2, there are a finite number of (t_1, t_2) pairs which can be derived from the combinations of release and due times across all tasks. Hooker (2007) gives an intuition for this relaxation by referring to $p_{ij}c_{ij}$ as the 'energy' of task j on facility i and $C_i(t_2 - t_1)$ as the total energy available in time window $[t_1, t_2]$ on facility i. Therefore the total energy of the tasks $j' \in J(t_1, t_2)$, $\sum_{j'} p_{ij'} c_{ij'}$, must not exceed the total available energy. Given the bin packing formulation, an alternative interpretation is that $p_{ij}c_{ij}$ is the area of task (item) j in facility (bin) i and $C_i(t_1 - t_2)$ is the total available area of that window.

The master problem for minimizing makespan differs on the objective, Benders cuts and subproblem relaxation. As for the MILP and logical models, we introduce the variable $M \geq 0$ and objective:

$$\min \ M. \tag{33}$$

The relaxation added to the problem is:

$$C_i M \geq \sum_{j \in \mathcal{J}} c_{ij} p_{ij} y_{ij}, \quad \forall i \in \mathcal{I}. \tag{34}$$

The relaxation is similar to the minimum cost relaxation however here the 'total area' is defined by the makespan. If the assignment is feasible, the

Benders cuts take the form:

$$M \geq M^*_{hi} - \sum_{j \in \bar{\mathcal{J}}_{hi}} (1 - y_{ij}) p_{ij}, \forall i \in \mathcal{I}_h, h \in \mathcal{H}, \tag{35}$$

where M^*_{hi} is the optimal makespan achieved on facility i with the assignment in iteration h. Similarly to minimum cost, the set $\bar{\mathcal{J}}_{hi}$ represents a subset of all tasks assigned to i however here it is a set that results in M^*_{hi} and removal of a task would cause M^*_{hi} to change. If the assignment is infeasible, we add an Eq. (30) cut to reject it. In SMT applying an algorithm to find such a set is unnecessary since the unsatisfiable core results in one. If the deadlines of the tasks in the set $\bar{\mathcal{J}}_{hi}$ are not all the same, we can add the stronger Benders cut (note w_{hi} is new):

$$M \geq M^*_{hi} - \sum_{j \in J_{hi}} (1 - y_{ij}) p_{ij} - w_{hi}, \tag{36}$$

$$w_{hi} \leq \left(\max_{j \in J_{hi}} \{d_j\} - \min_{j \in J_{hi}} \{d_j\} \right) \sum_{j \in J_{hi}} (1 - y_{ij}), \tag{37}$$

$$w_{hi} \leq \max_{j \in J_{hi}} \{d_j\} - \min_{j \in J_{hi}} \{d_j\}, \tag{38}$$

$$w_{hi} \geq 0, \tag{39}$$

for all $i \in \mathcal{I}, h \in \{1, \ldots, H-1\}$.

4.1.4. Numerical Results

To solve the MILP, CP and SMT models we used CPLEX 12.7, CP Optimizer 12.7 and Z3 (de Moura and Bjørner 2008a), respectively. The MILP models are in Pyomo (Hart et al. 2011, 2012), any implementations involving CP are in C++ and all further implementations are in Python (Z3 has a Python API). SMT assesses satisfiability rather than optimality, so we model the objective with a constraint that is iteratively tightened via the last objective found. All test cases were run on a HP EliteDesk 800 G1 TWR with 16GB RAM and an Intel® Core™ i7-4770 @ 3.40Ghz running Ubuntu 16.04.1 LTS. The test set, originally generated by Hooker (2007), can be found online[1]. There are 335 total instances from 4 classes: 195 from 'c', 50 from 'de', 40 from 'df' and 50 from 'e'. The subsequent analysis consists of performance profiles (Dolan and Moré 2002) and average runtimes. We say that an instance *terminates* if it proves infeasibility or converges within the time limit to the optimal solution. The performance profiles discard in-

[1] `http://web.tepper.cmu.edu/jnh/instances.htm`

stances with fewer than 18 tasks as toy problems. Results for these toy test cases may be biased towards implementation opposed to algorithm performance. The total number of problems after discarding toy instances is 230 among which there are 135 'c', 30 'de', 30 'df' and 35 'e' instances. The average run times are given for a subset of the 'c' instances. These instances were chosen as they cover the boundary at which some solvers begin to time out and are indicative of how much of an improvement can be achieved by using an alternative. Hooker (2007) found that a hybrid MILP/CP method outperforms both MILP and CP independently with respect to both in speed and in tractability of problems. Next we see if a similar result can be found with the use of hybrid MILP/SMT.

We analyze the results of minimizing costs first. Recall that Figure 2 discards problems with fewer than 18 tasks as toy instances. The hybrid implementations outperform all of the individual methods. The hybrid-CP implementation is fastest for more than 70% of the test set, hybrid-SMT becomes competitive given one order of magnitude. The hybrid implementations can solve close to 90% of the problems (nearly all tractable problems).

The average running time for some of the 'c' instances with 2, 3 and 4 facilities is presented in Table 3, we exclude the larger instances as there are a fair amount of timeouts beyond the instances shown. The hybrid method is clearly superior for larger instances. SMT appears to be slightly better than MILP and CP, but the hybrid formulation with the relaxation outperforms all three. The hybrid algorithm performs well for this objective because in each iteration we seek the next best assignment. This is handled by the MILP solver which is able to efficiently find it since it is not constrained by resource consumption or release and due times. The subproblem just has to check a series of independent resource constrained scheduling problems. Therefore we delegate parts of the problem in the correct locations.

We now analyze the makespan minimization results. The Figure 3 performance profile, similarly to the minimum cost tests, discards instances with fewer than 18 tasks as toy problems. CP solves almost all problems in the fastest time. SMT, MILP and Hybrid-CP are comparable in terms of tractable problems, CP solves at least 9% more problems. Hybrid-SMT performs the worst. The hybrid methods perform less well than using independent solvers. Table 4 records the average running time for some of the 'c' instances with 2, 3 and 4 facilities. Here CP outperforms all of the other solvers. SMT is about an order of magnitude slower and, as problems scale does not see as much of a loss of performance as MILP. The hybrid

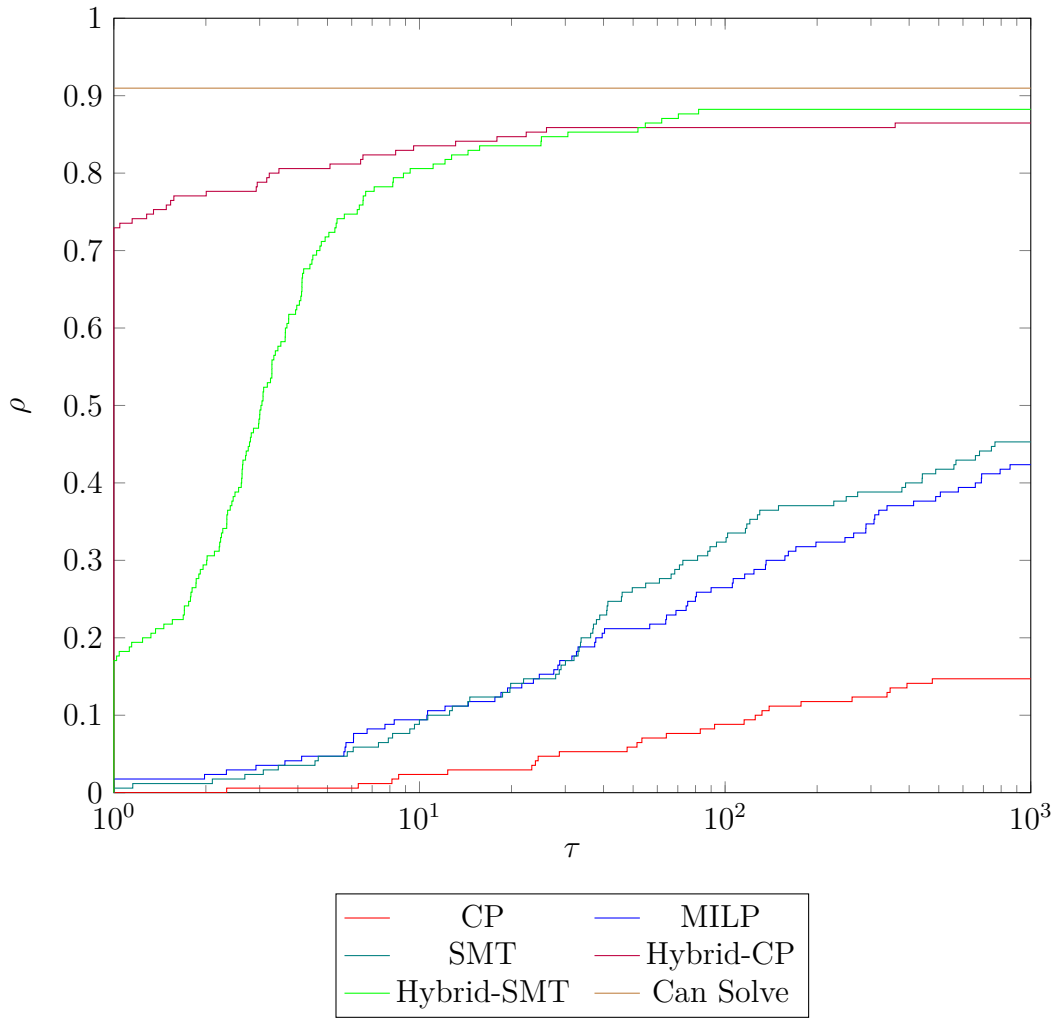

Figure 2: Minimum cost performance profile for 'c' and 'e' Hooker (2007) problems with at least 18 tasks (170/335 in the test set: 135 'c', 35 'e').

approaches have similar runtimes to that of MILP with Hybrid-CP generally performing the better as problems scale.

The CP and SMT algorithms perform well here because there is a tight coupling between constraints and objective, i.e. past search knowledge can be used. The hybrid algorithm does not perform as well because the MILP solver only learns about the feasible space through the Benders cuts hence the idea of a next best assignment may not have as much of an effect in the early iterations. Also the independent analysis of each assignment prevents the subproblem solver from learning about relations across facilities.

4.2. Dealing with Many Numeric Scales

Many PSE models may have to incorporate numerical values across very different scales. Existing optimization algorithms or tools may not be able

Table 3: Average minimum cost running times for the 5 'c' instances with $\#\mathcal{I}$ facilities and $\#\mathcal{J}$ tasks. A '+' indicates that at least one instance timed out (limit 3600s).

$\#\mathcal{I}$	$\#\mathcal{J}$	MILP	CP	SMT	Hybrid-CP	Hybrid-SMT
2	16	18.37	25.90	2.61	0.24	2.85
	18	41.05+	391.78	8.57	0.61	1.18
	20	1130.61	491.67+	28.93	1.85	1.22
	22	239.32+		437.83+	355.04	634.21
	24	14.04+		3004.79+	623.65+	1091.66
	26	2129.65+			180.59+	931.34+
	28	777.10+			973.85	6.94+
3	18	24.19+	7.27	3.68	0.95	2.39
	20	84.34+	35.29	11.53	0.39	0.89
	22	380.78+	495.98+	31.37	1.23	2.19
	24	685.10+		695.86	4.49	10.56
	26			2242.94+	53.74	100.11
	28	260.02+			70.61	714.43
	30	1027.52+			366.64+	190.24+
4	20	451.66	140.95	13.14	0.33	1.00
	22	362.50	1487.88+	69.95	0.83	2.46
	24	130.50+		1004.36+	6.72	35.23
	26			1614.68+	6.14	28.41
	28	455.26+		3355.82+	9.86	43.39
	30	2793.17+			45.44	209.43
	32				527.40	1079.15

to accommodate such needs nor gracefully degrade when dealing with such models. Let us discuss one such example, where the system process is that of trusted state replication in information systems, and where the engineering is concerned with designing such a system with appropriate tradeoffs: between the resilience of such a distributed system, its throughput, and overall cost of deployment and operation.

As a case study, consider a blockchain solution for replicated state (Lundbæk et al. 2016). The intuition is that each network node contains a copy of a chain of blocks, where each block contains a number of transactions that occurred in the system. This blockchain represents the state of the information system, which may for example record sensor readings, log entries and certification data. This blockchain is replicated across all network nodes by a consensus protocol that adds new blocks in a peer-to-peer fashion, where only valid blocks are forwarded in the network. Traditional approaches such as the PBFT protocol (Castro and Liskov 1999), which forms the foundation for IBM's Hyperledger platform, rely on a "leader" node that proposes a new

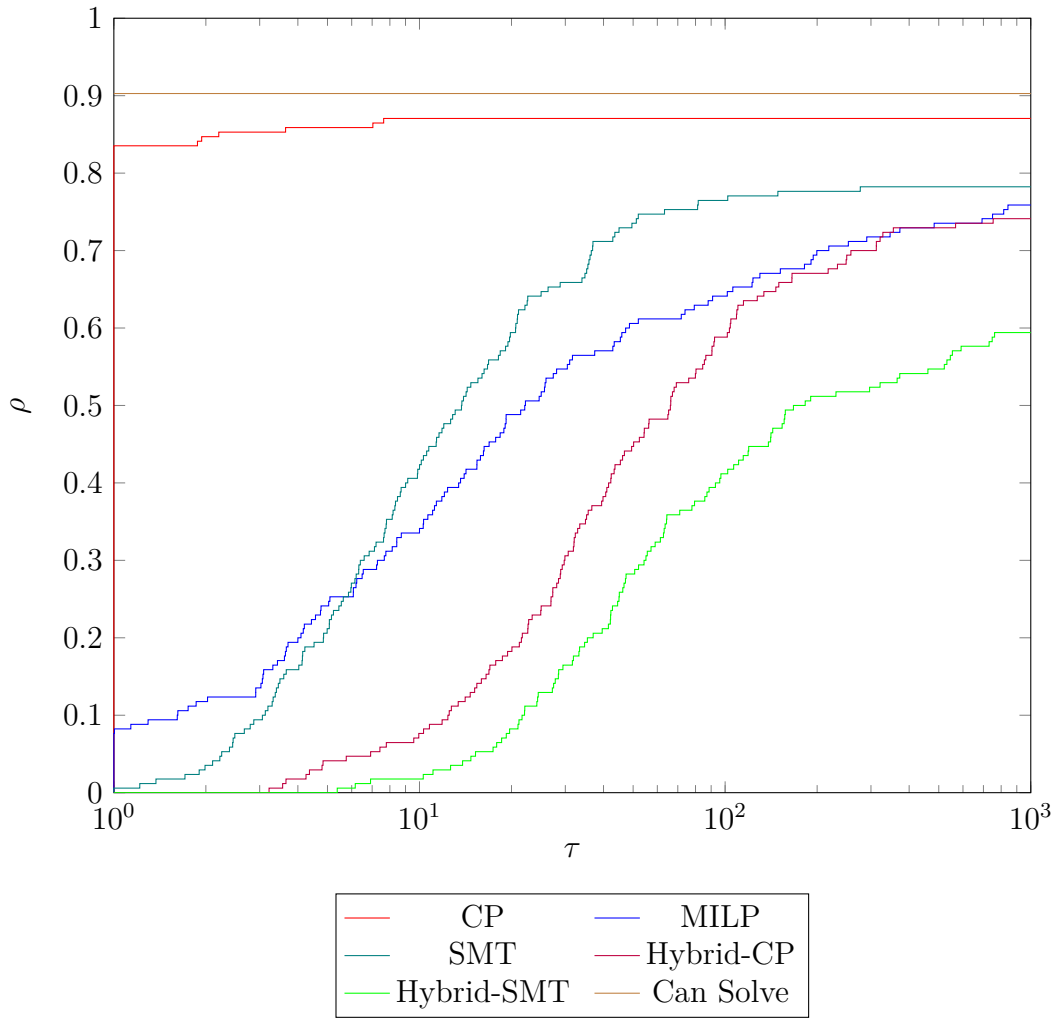

Figure 3: Minimum makespan performance profile for 'c' and 'e' Hooker (2007) problems with at least 18 tasks (170/335 in the test set: 135 'c', 35 'e').

block to be added to the chain.

Lundbæk et al. (2016) model the Proof of Work mechanism using a MINLP. Proof of Work elects a leader in a manner that is resilient to manipulation, but having miners compute in a race of solving a cryptographic puzzle, the the hash of some input has a certain level of difficulty – the number of leading zero bits. For s deployed miners, Lundbæk et al. (2016) derive the Figure 4 model. The Figure 4 constraints refer to random variables:

- $E^s(noR)$ is the expected number of (concurrent) solving attempts of all miners before a puzzle solution for the next block is found
- $prob^s(PoWTime > th)$ the probability that the *actual* time to find such a solution is larger than th
- $prob^s(PowTime < th')$ the probability that the actual time to find a

Table 4: Average minimum makespan running times for 5 'c' instances with $\#\mathcal{I}$ facilities & $\#\mathcal{J}$ tasks. A '+' indicates that at least one instance timed out (limit 3600s).

$\#\mathcal{I}$	$\#\mathcal{J}$	MILP	CP	SMT	Hybrid-CP	Hybrid-SMT
2	16	2.05	0.36	3.05	3.09	39.94
	18	101.02	0.79	2.69	14.79	186.91
	20	684.29	0.93	2.58	24.59	172.25
	22	208.94+	20.41	102.07	110.11	226.22+
	24	501.97+	15.97	369.77+	188.01	75.41+
	26	257.35+	42.69	2227.89+	921.40	
	28	2312.61+	589.03		2150.96+	
3	18	5.49	0.58	1.71	24.16	19.66
	20	7.37	0.11	0.81	8.08	5.87
	22	320.10	2.06	50.47	80.92	464.21
	24	443.19+	4.46	70.05	697.48	13.98+
	26	521.88+	1.99	15.44	87.85	793.55
	28	1177.62+	16.16	755.21+	174.58+	131.44+
	30	1743.12+	343.78	164.97+	769.82+	128.23+
4	20	0.95	0.10	0.55	4.89	5.25
	22	14.41	0.31	3.59	25.75	21.58
	24	69.33	0.96	8.36	46.24	260.94
	26	181.31+	1.50	7.66	193.16	135.14
	28	167.35	1.12	13.51	67.19	18.75+
	30	201.01+	10.97	114.51	664.59	436.32+
	32	517.89+	5.27	97.14	191.85	626.97

puzzle solution is smaller than th'

- $prob^s(\mathsf{failure})$ the probability that all s miners fail to find a puzzle solution for the next block with the given resources of randomness, and
- $prob^s(\textit{disputes within } \mu)$ the probability that more than one miner finds a puzzle solution within μ seconds.

The constant T is the time in seconds that miners need to complete one attempt of solving the puzzle. This model also captures that puzzle solutions occur, on average, within a specified time window $[\tau_l, \tau_u]$ and that worst-case solution time is within $[th', th]$ with large, specified probability. We also want to understand how likely mining disputes may occur within μ seconds, that more than one miner finds a puzzle solution within that period of time. Such insights may inform consensus protocol design, for example by realizing that so-called *blockchain forks* occur with sufficiently small probability.

The parameter r represents the number of bits of shared randomness that miners may use to attempt puzzle solutions. The cost function we consider

$$
\begin{array}{rcl}
0 < s_l \leq s \leq s_u \qquad 0 < d_l \leq d \leq d_u \qquad 0 < r_l \leq r \leq r_u \qquad \epsilon \geq prob^s(\mathsf{failure}) \\
\tau_u \geq T \cdot E^s(noR) \geq \tau_l \qquad\qquad \delta_2 \geq prob^s(disputes\ within\ \mu) \\
\delta \geq prob^s(PoWTime > th) \qquad \delta_1 \geq prob^s(PoWTime < th')
\end{array}
$$

Figure 4: Constraint set $\mathcal{C}$ for two optimization problems: (a) *minimize* $\mathrm{Cost}(s, r, d)$ as in (40) subject to constraints in $\mathcal{C}$; and (b) *maximize* d subject to $\mathcal{C} \cup \{\mathrm{Cost}(s, r, d) \leq budget\}$ for cost bound *budget*. This is parameterized by constants $0 \leq \delta, \delta_1, \delta_2, \epsilon, th, th', \tau_l, \mathrm{TVC}, \mathrm{TFC}$ and $0 < T, s_l, r_l, d_l$. Variables or constants $s_l, s_u, s, d_l, d_u, d, r_l, r_u, r$ are integral

$$
\begin{array}{rcl}
s_l \leq s \leq s_u \qquad d_l \leq d \leq d_u \qquad r_l \leq r \leq r_u \qquad \lambda = \lfloor 2^r/s \rfloor \\
y = (1 - 2^{-d})^s \qquad w = (1 - 2^{-d})^{\lfloor \mu/T \rfloor + 1} \qquad 0 \leq \lfloor \mu/T \rfloor \\
\epsilon \geq y^{\lambda+1} \qquad \lceil (th/T) - 1 \rceil < \lambda \qquad 0 < \lfloor (th'/T) - 1 \rfloor \\
E^s(noR) = \dfrac{1 - y^{\lambda+1} - (\lambda + 1) \cdot (1 - y) \cdot y^{\lambda+1}}{1 - y} \\
\tau_u \geq T \cdot E^s(noR) \geq \tau_l \qquad \delta_1 \geq 1 - y^{\lfloor (th'/T) - 1 \rfloor + 1} \\
\delta \geq y^{\lceil (th/T) - 1 \rceil + 1} - y^{\lambda+1} \\
\delta_2 \geq 1 + (s - 1) \cdot w^s - s \cdot w^{s-1}
\end{array}
$$

Figure 5: Arithmetic version of set of constraints $\mathcal{C}$ from Figure 4, with additional soundness constraints for this representation.

for these constraints is

$$
\mathrm{Cost}(s, r, d) = \mathrm{TVC} \cdot E^s(noR) \cdot s + \mathrm{TFC} \cdot s \tag{40}
$$

where d is the level of difficulty (the number of leftmost bits that need to be 0 in a hash to solve the puzzle), s is the number of deployed miners, TVC models the cost of one solution attempt by one miner (dependent on prize of energy), TFC models fixed costs such as depreciation of a mining unit, and this cost function implicitly depends on r since the random variable $E^s(noR)$ does so. A mathematical analysis of this model turns these random variables into explicit arithmetic expressions depicted in Figure 5, reproduced from Lundbæk et al. (2016).

To get a sense of the numerical scales here, we may have $[r_l, r_u] = [64, 128]$, $[s_l, s_u] = [4, 64]$, $T = 0.002 \cdot 10^{-9}$, and so forth. This means that many expressions have a base in $(0, 1)$ with exponents that may be as large as 2^{128}, and that there are also subtractions of such numbers within the context of inequalities. This means that tools that rely on normal floating-point arithmetic, such as the tools ANTIGONE (Misener and Floudas 2014), BARON

(Sahinidis 1996), and SCIP (Achterberg 2009), judge the models to be infeasible in their preprocessing stages already, although the models are indeed feasible. In fairness, these global MINLP solvers were not built for dealing with numerical problems such as those encountered in those models.

With that understanding, we may turn to the arbitrary-precision arithmetic package `Data.BigFloat`, written in the functional programming language Haskell and used in the Banking sector in some applications to ensure compliance. This package uses bespoke algorithms that guarantee that returned results are correct up to and excluding the last bit and where the bit precision is specifiable. These algorithms may not terminate since arbitrary precision real-arithmetic is undecidable in general.

Lundbæk et al. (2016) generate a list of all triples (s, r, d) for which the non-robust constraints are feasible. Then we remove all such triples that are not robustly feasible from that list (Robustness under varying levels of puzzle difficulty, number of corrupted miners, and so forth), and finally only list what amounts to the k triples with the maximal k values of d for which cost is within *budget*, where the latter is some user-specified function the optimal cost of all robustly feasible triples.

This computation is efficient since the combinatorial space for the integral values of (s, r, d) is moderately large. For sake of efficiency, all computations are performed with a medium level of precision set in `Data.BigFloat`. But then we verify the feasibility of those k top triples by validating their robust feasibility with much higher precision reset in `Data.BigFloat`. These verification computations take much longer but they are only done a few times, we often use $k = 5$. This approach *underapproximates* optimal solutions: some feasible triples may not be added to the initial list and so the k triples that are reported may not be the best possible solutions. This trades of efficiency of arithmetical computation with precision of approximation of optima.

This approach was used in Lundbæk et al. (2016) to understand, for example, what parameter choices will support sufficient resiliency against failure and corrupted miners, yet guarantee that the number of transactions added to the blockchain within blocks is similar to that of Visa's global system workload. For example, this may require 18 miners, a level of difficulty of 35, and at least 48 bits of randomness r shared across the 18 miners at a cost of slightly over $54,000$ USD; we refer to Section 5 in Lundbæk et al. (2016) for more details on such analyses.

Based on this discussion, it would be of interest to build MINLP solvers, or refactor existing ones so that they could follow a similar approach: resource

intense computations done with lower precision, validation of results done with much higher precision. Also, it would be very useful to support this approach for the various techniques that reduce MINLP to other problem types such as NLP – branch-and-bound being one example.

Such tool enhancements would allow us to explore much richer versions of the Proof of Work model above, for example by faithfully modeling consortia partners and their system resources. And it would provide opportunities for SMT solving to interact with such a tool and its components, for example within the `ManyOpt` approach discussed in Section 4.3.4 below.

4.3. Optimization Solvers based on Satisfiability Modulo Theories

SMT-based optimization solvers typically extend SMT solvers, e.g. Barcelogic (Bofill et al. 2008), Yices (Dutertre and de Moura 2006) and Z3 (de Moura and Bjørner 2008a). This section discusses solvers: νZ (§4.3.1), OptiMathSAT (§4.3.2) Symba (§4.3.3), and ManyOpt (§4.3.4). We focus on ManyOpt, the first SMT-based MINLP solver.

4.3.1. νZ

The νZ extension to the SMT solver Z3 adds optimization functionality (Bjørner and Phan 2014, Bjørner et al. 2015). νZ addresses weighted MaxSAT/SMT and linear arithmetic problems. νZ also supports multi-objective optimization via: lexicographic, Pareto fronts and box objectives. The νZ algorithms resemble the Nieuwenhuis and Oliveras (2006) approach, but, instead of implementing additional rules, νZ updates a variable value as the search progresses. The solver also has resolution-based approach that reasons based on the unsatisfiable core (Narodytska and Bacchus 2014).

Bjørner and Phan (2014) solve the MILP subproblems by using SMT to find a feasible solution and invoking the simplex algorithm to minimize under the given integral assignment, the integral assignment is then rejected and a bounding constraint is placed on the objective.

4.3.2. OptiMathSAT

OptiMathSAT (Sebastiani and Trentin 2015) extends the SMT solver MATSAT5 (Cimatti et al. 2013). The optimization capabilities are similar to νZ, but the optimization constraints are formulated differently. Sebastiani and Tomasi (2015) compare OptiMathSAT to a GDP framework and show that the solver is competitive with the state-of-the-art.

4.3.3. SYMBA

SYMBA (Li et al. 2014) is built on top of Z3, but its implementation differs from νZ. SYMBA optimizes for linear *real* arithmetic (not limited to rational) however it does not support strict inequalities. The optimization process maintains a triple $\langle M, U, O \rangle$ where M is a set of models, U is an underapproximation and O is an overapproximation. The inference rules correspond to initialization, checking for unboundedness and tightening the under and over approximations. Symba only uses Z3 as a black box.

4.3.4. MANYOPT

`ManyOpt` (Callia D'Iddio and Huth 2017) is an MINLP solver extending the SMT solving. The `ManyOpt` design is inspired by ManySAT (Hamadi et al. 2009). The hypothesis is that no single method wins on a varied class of optimization problems. Therefore, `ManyOpt` runs different combinations in parallel on a problem input and sees which "wins" on that input. This approach thus supports solving time speedup, and also allows for the empirical comparison of such combination methods on input data sets.

4.3.5. MANYOPT *Tool Architecture*

Figure 6 diagrams the main structure `ManyOpt`. As input, `ManyOpt` takes a desired *accuracy* for the computed global optimum and, either an OSiL model (Fourer et al. 2010) for a MINLP problem. Then `ManyOpt` executes *in parallel* a set of dataflows on that OSiL input, where each dataflow has its own feature vector of how to realize configurable tool layers. A feature is a particular method or heuristic and a feature vector represents a combination of all features to create an SMT-based optimizer.

The tool layers are Preprocessing, Integrality Management, Continuous Relaxation Optimization, and Feasibility Checking. Each of these layers contains within it an (extensible) list of features, as seen in Fig. 6.

For each feature vector, the data flow of its process is as follows. First, an SMTLIB 2.0 representation of the model is constructed, based on the input and selected features. After that, a *Preprocessing* phase may transform the SMTLIB 2.0 representation using either *Binarization* or a novel *Binarized Flattening* techniques to convert the MINLP problem into a MBNLP problem, where all integer variables can only have values 0 or 1.

Second, the Main Optimization Process uses the lower three layers, organized as a *stack*, to find an optimal solution for the given model within the specified accuracy. The *Integrality Management* layer may apply *branch-and-bound* techniques – interpreted *logically* through SMT – such as One-By-One

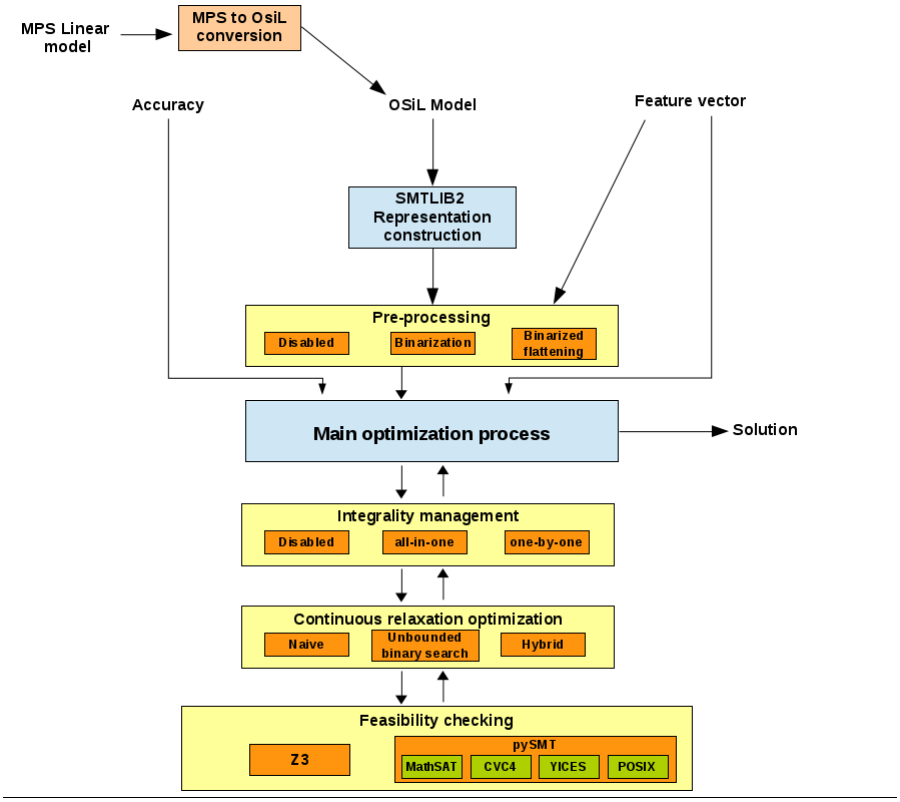

Figure 6: The architecture and approach of `ManyOpt` (Callia D'Iddio and Huth 2017)

or our novel All-In-One to handle integrality constraints. This layer produces NLP problems as continuous relaxations of the original MINLP problem, and then relies on the *Continuous Relaxation Optimization* layer to solve them.

The Continuous Relaxation Optimization layer takes an NLP problem as input and reduces optimization to the repeated use of the *Feasibility Checking* layer. A number of such reduction algorithms are supported: the *naive* method (the term "naive" is used without prejudice and inspired by work on pseudo-boolean optimization (Eén and Sörensson 2006)), the *unbounded binary search* method See, e.g. Beaumont et al. (2015), Callia D'Iddio and Huth (2017) and a *hybrid* combination of these two methods (Callia D'Iddio and Huth 2017).

The feasibility of an NLP problem, represented in SMTLIB 2.0, is verified using an SMT solver. `ManyOpt` supports in the Figure 6 feasibility layer, the solver `Z3` (de Moura and Bjørner 2008b) directly, the solvers `MathSAT` (Bruttomesso et al. 2008), `CVC4` (Barrett et al. 2011) and `YICES` (Dutertre 2014) through the `PySMT` library (Gario and Micheli 2015); other SMT solvers compatible with SMTLIB 2.0 may be used through a *POSIX piping*.

4.3.6. ManyOpt *Supported Features*

Many of the features that `ManyOpt` supports mimick techniques familiar from MINLP optimization, but they tend to *delegate algorithmic aspects to the SMT solver and focus on declarative aspects.*

Binarization. Binarization mechanisms are widely used for mixed optimization, in order to handle integrality constraints where variables can only have values 0 or 1 (see Floudas (1995) Subsection 6.2.1), thus transforming the MINLP problem into an MBNLP problem. Formally, an integrality constraint $x \in \mathbb{Z}$ for variable x bounded by $l \leq x \leq u$ is represented by q many variables $b_1, \ldots, b_q$ with $q = 1 + \lceil \log_2(u - l) \rceil$, equality constraint $x = l + b_1 + 2b_2 + 4b_3 + \cdots + 2^{q-1}b_q$ and inequalities $0 \leq b_i \leq 1$ for all i such that $1 \leq i \leq q$ added to the optimization problem. Now, integrality constraints $b_i \in \mathbb{Z}$ can replace the original $x \in \mathbb{Z}$ integrality constraint.

Binarized Flattening. An alternative to (algorithmic) branch and bound consists in *flattening* declaratively the integrality constraints during the preprocessing stage, thus and moving algorithmic complexity to the SMT solvers via the OR operator. This method can be imagined as an "extreme" version of branch and bound, which tries to prevent all the integrality violations by adding specific constraints before starting the actual solving of the optimization problem. Formally, for each integrality constraint $x \in \mathbb{Z}$ where x is bounded by $l \leq x \leq u$ we add the assertion

$$(x \geq l) \wedge \bigwedge_{l \leq i \leq u} (x \leq i \vee x \geq i + 1) \wedge (x \leq u) \tag{41}$$

If this approach is applied as it is, then it would be dramatically inefficient. But, if this approach is combined with binarization, each variable involved in integrality constraints will be bounded just by 0 and 1. Then only one disjunction is needed to exclude all the non-integer values between 0 and 1. For $b \in \mathbb{Z}$, we "flatten" this by simply adding the assertion

$$(b = 0) \vee (b = 1) \tag{42}$$

The one-by-one approach. This approach is similar to a standard branch and bound approach. If a feasible solution is found that violates at least one integrality constraint, choose one variable x whose value v violates an integrality constraints and add the assertion $x \leq \lfloor v \rfloor \vee x \geq \lceil v \rceil$ to the optimization problem. Since disjunctions are supported in SMT as assertions, this is well

defined and avoids a split into two optimization problems by delegating such combinatorial complexity to the SAT engine of the SMT solver.

The all-in-one approach. We call the above approach one-by-one, as each iteration only adds a sole assertion about a sole variable. But SMT solvers allow us to add more than one constraint at a time, leading to the *all-in-one* approach; it collects *all* variables whose values violate integrality constraints in a feasible solution and adds the above disjunction as an assertion but for *all* such variables simultaneously.

Delegating to lower layers. We may also delegate the management of integrality constraints to the Feasibility Checking layer, where SMT solvers can attempt to manage this – for example with *integer* type variables.

4.3.7. MANYOPT *Continuous relaxation optimization*

This layer provides algorithms to solve NLP problems.

The naive approach. This method is simple but familiar in optimization (see e.g. Eén and Sörensson (2006)). It consists of a loop in which a value is found for the objective function using the feasibility checker, and an attempt to find a lower value is made by adding an assertion saying that the objective function must be smaller than that value. When the problem becomes infeasible, the last found objective value, if there is one, is the optimal solution.

Unbounded binary search. This method has two main phases (see e.g. Beaumont et al. (2015)):

1. The *bounds search*, in which initial lower and upper bounds are found such that the optimal value of the objective is between such bounds.
2. The *bisection phase*, in which the interval between the lower and upper bound is restricted by splitting it in two equal parts, until the optimal value of the objective is found, relative to the specified accuracy.

Hybrid method. This modifies unbounded binary search so that in each phase "naive steps" are used to determine whether the current value is already optimal – in which case the method stops.

4.3.8. MANYOPT *experimental results*

Appendix A gives some experimental results for MINLPLib (Bussieck et al. 2003) benchmarks solved by `ManyOpt`.

5. Conclusion

This manuscript proposes applying satisfiability modulo theories to process systems engineering. We motivate our position using three test beds: (i) planning and scheduling, (ii) applications with many numeric scales, (iii) MINLP solvers. For planning and scheduling, we consider a logic-based Benders decomposition technique combining MILP and SMT. Prior LBBD approaches use constraint programming rather than SMT. We find that hybrid MILP/SMT techniques are significantly stronger than either technique individually on minimum cost models of scheduling, but that the hybrid is weaker for minimum makespan. For applications with many numerical scales, we find that SMT can handle the difficult calculations in the blockchain example. Finally, we develop an MINLP solver using SMT technology and that this solver `ManyOpt` can handle relevant process systems engineering applications.

Acknowledgments

The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, EP/L016796/1), EPSRC EP/P008739/1, and a Royal Academy of Engineering Research Fellowship to R.M. is gratefully acknowledged.

References

Achterberg, T. 2007a. Conflict analysis in mixed integer programming. *Discrete Optim.* **4** 4–20.

Achterberg, T. 2007b. Constraint integer programming. Ph.D. thesis, Technische Universität Berlin.

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Math. Program. Comput.* **1** 1–41.

Aminof, B., O. Kupferman, R. Lampert. 2011. Formal analysis of online algorithms. *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*. 213–227.

Balas, E. 1974. Intersection cuts from disjunctive constraints. Tech. Rep. Management Sciences Research Report No. 330, Graduate School of Industrial Administration, Carnegie Mellon University.

Balas, E. 1975. Disjunctive programming: Cutting planes from logical conditions. *Nonlinear Programming 2*. Elsevier, 279–312.

Balas, E. 1977. A note on duality in disjunctive programming. *J. Optim. Theory Appl.* **21** 523–528.

Balas, E. 1979. Disjunctive programming. *Ann. Discrete Math.* **5** 3–51.

Barrett, C., C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, C. Tinelli. 2011. CVC4. *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA*. 171–177.

Beaumont, P., N. Evans, M. Huth, T. Plant. 2015. Confidence analysis for nuclear arms control: SMT abstractions of Bayesian belief networks. *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria*. 521–540.

Bemporad, A., N. Giorgetti. 2006. Logic-based solution methods for optimal control of hybrid systems. *IEEE Transactions on Automatic Control* **51** 963–976.

Bemporad, Alberto, Nicolò Giorgetti. 2004. *SAT-Based Branch & Bound and Optimal Control of Hybrid Dynamical Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 96–111.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4** 238–252.

Biere, A., A. Cimatti, E. Clarke, Y. Zhu. 1999. *Tools and Algorithms for the Construction and Analysis of Systems: 5th International Conference, TACAS'99*, chap. Symbolic Model Checking without BDDs. Springer Berlin Heidelberg, Berlin, Heidelberg, 193–207.

Biere, A., M. Heule, H. van Maaren, T. Walsh. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands.

Bjørner, N., L. De Moura. 2011. Satisfiability modulo theories: Introduction and applications. *Commun. ACM* 69–77.

Bjørner, N., A. D. Phan. 2014. νZ - maximal satisfaction with Z3. *Proceedings of the 6th International Symposium on Symbolic Computation in Software Science*, *SCSS*, vol. 30. 1–9.

Bjørner, N., A.-D. Phan, L. Fleckenstein. 2015. *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015*, chap. νZ - An Optimizing SMT Solver. Springer Berlin Heidelberg, 194–199.

Bofill, M., R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio. 2008. *Computer Aided Verification: 20th International Conference, CAV 2008*, chap. The Barcelogic SMT Solver. Springer Berlin Heidelberg, 294–298.

Bollapragada, S., O. Ghattas, J. N. Hooker. 2001. Optimal design of truss structures by logic-based branch and cut. *Oper. Res.* **49** 42–51.

Bruttomesso, R., A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani. 2008. The MathSAT 4SMT solver. *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA*. 299–303.

Bussieck, M. R., A. S. Drud, A. Meeraus. 2003. MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming. *INFORMS J. Comput.* **15** 114–119.

Callia D'Iddio, A., M. Huth. 2017. Manyopt: An extensible tool for mixed, nonlinear optimization through SMT solving. *CoRR* **abs/1702.01332**.

Carbonneau, R. A., G. Caporossi, P. Hansen. 2011. Globally optimal clusterwise regression by mixed logical-quadratic programming. *Eur. J. Oper. Res.* **212** 213 – 222.

Carbonneau, R. A., G. Caporossi, P. Hansen. 2012. Extensions to the repetitive branch and bound algorithm for globally optimal clusterwise regression. *Comput. Oper. Res.* **39** 2748 – 2762.

Carvajal, R., S. Ahmed, G. Nemhauser, K. Furman, V. Goel, Y. Shao. 2014. Using diversification, communication and parallelism to solve mixed-integer linear programs. *Oper. Res. Lett.* **42** 186–189.

Castro, M., B. Liskov. 1999. Practical byzantine fault tolerance. *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA*. 173–186.

Cimatti, A., A. Griggio, B. J. Schaafsma, R. Sebastiani. 2013. *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013*, chap. The MathSAT5 SMT Solver. Springer Berlin Heidelberg, 93–107.

Cook, Stephen A. 1971. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, New York, New York, USA, 151–158.

Davis, M., G. Logemann, D. Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* **5** 394–397.

Davis, M., H. Putnam. 1960. A computing procedure for quantification theory. *J. ACM* **7** 201–215.

de Moura, L., N. Bjørner. 2008a. *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008*, chap. Z3: An Efficient SMT Solver. Springer Berlin Heidelberg, 337–340.

de Moura, L., N. Bjørner. 2008b. Z3: an efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Budapest, Hungary*. 337–340.

de Moura, L., G. O. Passmore. 2013. Computation in real closed infinitesimal and transcendental extensions of the rationals. M. P. Bonacina, ed., *Automated Deduction - CADE-24*, *Lecture Notes in Computer Science*, vol. 7898. Springer Berlin Heidelberg, 178–192.

Dolan, E. D., J. J. Moré. 2002. Benchmarking optimization software with performance profiles. *Math. Program.* **91** 201–213.

Dutertre, B. 2014. Yices 2.2. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, AU*. 737–744.

Dutertre, B., L. de Moura. 2006. *Computer Aided Verification: 18th International Conference, CAV 2006*, chap. A Fast Linear-Arithmetic Solver for DPLL(T). Springer Berlin Heidelberg, 81–94.

Eén, N., N. Sörensson. 2006. Translating pseudo-Boolean constraints into SAT. *JSAT* **2** 1–26.

Fattahi, A., S. Elaoud, E. S. Azer, M. Turkay. 2014. A novel integer programming formulation with logic cuts for the U-shaped assembly line balancing problem. *International J. Production Res.* **52** 1318–1333.

Floudas, C.A. 1995. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, New York, NY.

Fourer, R., J. Ma, R. K. Martin. 2010. OSiL: An instance language for optimization. *Comp. Opt. Appl.* **45** 181–203.

Ganzinger, H., G. Hagen, R. Nieuwenhuis, A. Oliveras, C. Tinelli. 2004. *Computer Aided Verification: 16th International Conference, CAV 2004*, chap. DPLL(T): Fast Decision Procedures. Springer Berlin Heidelberg, 175–188.

Garey, M. R., R. L. Graham, D. S. Johnson, Andrew Chi-Chih Yao. 1976. Resource constrained scheduling as generalized bin packing. *J Combin Theory, Ser A* **21** 257–298.

Gario, M., A. Micheli. 2015. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. *In Proc. of the 13th International Workshop on Satisfiability Modulo Theories (SMT)*. 373–384.

Grossmann, I. E., J. P. Ruiz. 2012. Generalized disjunctive programming: A framework for formulation and alternative algorithms for MINLP optimization. J. Lee, S. Leyffer, eds., *Mixed Integer Nonlinear Programming*. Springer New York, New York, NY, 93–115.

Hamadi, Y., S. Jabbour, L. Sais. 2009. ManySAT: A parallel SAT solver. *JSAT* **6**. IOS Press.

Hart, W. E., J.-P. Watson, D. L. Woodruff. 2011. Pyomo: modeling and solving mathematical programs in Python. *Math. Program. Comput.* **3** 219–260.

Hart, William E, Carl Laird, Jean-Paul Watson, David L Woodruff. 2012. *Pyomo–optimization modeling in Python*, vol. 67. Springer Science & Business Media.

Hooker, J. N. 2002. Logic, optimization, and constraint programming. *INFORMS J. Comput.* **14** 295–321.

Hooker, J. N. 2007. Planning and scheduling by logic-based Benders decomposition. *Oper. Res.* **55** 588–602.

Hooker, J. N., M. A. Osorio. 1999. Mixed logical-linear programming. *Discrete Appl. Math.* **96-97** 395–442.

Hooker, J. N., G. Ottoson. 2003. Logic-based Benders decomposition. *Math. Program.* **96** 33–60.

Jain, V., I. E. Grossmann. 2001. Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems. *INFORMS J. Comput.* **13** 258–276.

Kautz, H., B. Selman. 1992. Planning as satisfiability. *European Conference on Artificial Intelligence*. ECAI '92, John Wiley & Sons, Inc., New York, NY, USA, 359–363.

Lee, S., I. E. Grossmann. 2000. New algorithms for nonlinear generalized disjunctive programming. *Comput. Chem. Eng.* **24** 2125–2141.

Li, H., K. Womer. 2008. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. *J. Sched.* **12** 281–298.

Li, Y., A. Albarghouthi, Z. Kincaid, A. Gurfinkel, M. Chechik. 2014. Symbolic optimization with smt solvers. *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14*. ACM Press, New York, New York, USA, 607–618.

Lundbæk, L.-N., A. Callia D'Iddio, M. Huth. 2016. Optimizing Governed Blockchains for Financial Process Authentications URL `http://arxiv.org/abs/1612.00407`.

Malik, S., L. Zhang. 2009. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM* **52** 76.

Manolios, P., V. Papavasileiou. 2013. *Computer Aided Verification: 25th International Conference, CAV 2013*, chap. ILP Modulo Theories. Springer Berlin Heidelberg, 662–677.

Maravelias, C. T. 2006. A decomposition framework for the scheduling of single- and multi-stage processes. *Comput. Chem. Eng.* **30** 407–420.

Maravelias, C. T., I. E. Grossmann. 2004. A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Comput. Chem. Eng.* **28** 1921–1949.

Maravelias, C. T., C. Sung. 2009. Integration of production planning and scheduling: Overview, challenges and opportunities. *Comput. Chem. Eng.* **33** 1919–1930.

Misener, R., C. A. Floudas. 2014. ANTIGONE: Algorithms for coNTinuous Integer Global Optimization of Nonlinear Equations. *J. Glob. Optim.* **59** 503–526.

Mistry, M., R. Misener. 2017. Integrating mixed-integer optimisation and satisfiability modulo theories: Application to scheduling. C. Maravelias, J. Wassick, eds., *Computer-Aided Chemical Engineering*. Foundations of Computer-Aided Process Operations.

Narodytska, N., F. Bacchus. 2014. Maximum satisfiability using core-guided MaxSAT resolution. *AAAI Conference on Artificial Intelligence*. 2717–2723.

Nelson, G., D. C. Oppen. 1979. Simplification by cooperating decision procedures. *ACM T. Program. Lang. Syst.* **1** 245–257.

Nelson, G., D. C. Oppen. 1980. Fast decision procedures based on congruence closure. *J. ACM* **27** 356–364.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA.

Nieuwenhuis, R., A. Oliveras. 2006. *Theory and Applications of Satisfiability Testing - SAT 2006*, chap. On SAT Modulo Theories and Optimization Problems. Springer Berlin Heidelberg, 156–169.

Papageorgiou, D. J., F. Trespalacios. 2016. Pseudo basic steps: Bound improvement guarantees from Lagrangian decomposition in convex disjunctive programming URL `http://www.optimization-online.org/DB_FILE/2016/09/5627.pdf`.

Park, M., S. Park, F. D. Mele, I. E. Grossmann. 2006. Modeling of purchase and sales contracts in supply chain optimization. *2006 SICE-ICASE International Joint Conference*. 5727–5732.

Raman, R., I. E. Grossmann. 1994. Modelling and computational techniques for logic based integer programming. *Comput. Chem. Eng.* **18** 563–578.

Raman, V., N. Piterman, H. Kress-Gazit. 2013. Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. 4075–4081.

Rodriguez, M. A., A. Vecchietti. 2009. Logical and generalized disjunctive programming for supplier and contract selection under provision uncertainty. *Ind. Eng. Chem. Res.* **48** 5506–5521.

Ruiz, J. P., J.-H. Jagla, I. E. Grossmann, A. Meeraus, A. Vecchietti. 2012. *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, chap. Generalized Disjunctive Programming: Solution Strategies. Springer Berlin Heidelberg, Berlin, Heidelberg, 57–75.

Sahinidis, N. V. 1996. BARON: general purpose global optimization software package. *J. Glob. Optim.* **8** 201–205.

Sawaya, N. W., I. E. Grossmann. 2005. A cutting plane method for solving linear generalized disjunctive programming problems. *Comput. Chem. Eng.* **29** 1891–1913.

Sebastiani, R. 2007. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation* **3** 141–224.

Sebastiani, R., S. Tomasi. 2015. Optimization modulo theories with linear rational costs. *ACM T. Comput. Log. (TOCL)* **16** 1–43.

Sebastiani, R., P. Trentin. 2015. *Computer Aided Verification: 27th International Conference, CAV 2015*, chap. OptiMathSAT: A Tool for Optimization Modulo Theories. Springer International Publishing, 447–454.

Shostak, R. E. 1979. A practical decision procedure for arithmetic with function symbols. *J. ACM* **26** 351–360.

Shostak, R. E. 1982. *6th Conference on Automated Deduction: New York, USA*, chap. Deciding combinations of theories. Springer Berlin Heidelberg, Berlin, Heidelberg, 209–222.

Silva, J. P. M., K. A. Sakallah. 1996. Grasp: a new search algorithm for satisfiability. *IEEE/ACM International Conference on Computer-aided Design*. ICCAD '96, IEEE Computer Society, Washington, DC, USA, 220–227.

Sitek, P. 2014. A hybrid CP/MP approach to supply chain modelling, optimization and analysis. *Computer Science and Information Systems (FedCSIS)*. 1345–1352.

Thorsteinsson, E. S. 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. *Principles and Practice of Constraint Programming - CP 2001* **2239** 16–30.

Trespalacios, F., I. E. Grossmann. 2014. Review of mixed-integer nonlinear and generalized disjunctive programming methods. *Chemie Ingenieur Technik* **86**.

Trespalacios, F., I. E. Grossmann. 2016. Cutting plane algorithm for convex generalized disjunctive programs. *INFORMS J. Comput.* **28** 209–222.

Türkay, M., I. E. Grossmann. 1996. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Comput. Chem. Eng.* **20** 959–978.

Vecchietti, A., S. Lee, I. E. Grossmann. 2003. Modeling of discrete/continuous optimization problems: Characterization and formulation of disjunctions and their relaxations. *Comput. Chem. Eng.* **27** 433–448.

Zhang, H. 2002. Generating college conference basketball schedules by a SAT solver. *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing*. 281–291.

Appendix A. `ManyOpt`Statistics for Solved Benchmarks

Statistics for solved benchmarks

Benchmark	Type	#Var	#Con	Found objective	Solved by	#Iter	Solving time	Total time
gear	INLP	5	1	0.0007	allinone.ubs	48	0.19	0.28
nvs04	INLP	3	1	0.72	allinone.ubs	55	0.32	0.38
nvs06	INLP	3	1	1.7703	onebyone.nve	28	0.40	0.47
nvs07	INLP	4	3	4.0	allinone.hyb	37	0.23	0.29
nvs16	INLP	3	1	0.7031	allinone.nve	36	0.24	0.28
prob02	IQCP	7	9	112235.0	bin_flat.hyb	36	17.71	17.76
prob03	IQCP	3	2	10.0	bin_flat.ubs	5	0.00	0.03
nvs03	IQCQP	3	3	16.0	bin_flat.nve	4	0.02	0.06
nvs10	IQCQP	3	3	-310.8	bin_flat.hyb	12	0.13	0.19
nvs11	IQCQP	4	4	-431.0	bin_flat.nve	9	2.71	2.72
nvs12	IQCQP	5	5	-481.2	bin_flat.nve	12	992.20	999.18
nvs15	IQP	4	2	1.0	bin_flat.ubs	4	0.02	0.03
st_miqp2	IQP	5	4	2.0	allinone.hyb	43	0.31	0.38
st_miqp3	IQP	3	2	-6.0	nobb.ubs	14	0.02	0.06
st_test1	IQP	6	2	-32.0059	bin_flat.ubs	20	0.11	0.14
st_test2	IQP	7	3	-9.25	bin_flat.nve	3	26.04	26.07
st_test3	IQP	14	11	-7.0	bin_flat.ubs	6	41.77	41.93
st_test4	IQP	7	6	-7.0	nobb.ubs	14	0.06	0.08
st_testph4	IQP	4	11	-80.5	bin_flat.nve	6	0.03	0.06
ex1221	MBNLP	6	6	7.6671	bin_flat.hyb	4	0.13	0.14
ex1226	MBNLP	6	6	-17.0	bin_flat.hyb	2	0.02	0.03
gear2	MBNLP	29	5	0.0003	allinone.nve	10	1.35	1.45
st_e15	MBNLP	6	6	7.6671	bin_flat.hyb	4	0.12	0.15
autocorr_bern20-03	MBQCP	22	2	-72.0	bin_flat.nve	74	0.35	0.41
autocorr_bern25-03	MBQCP	27	2	-92.0	bin_flat.nve	94	3.65	3.78
sporttournament06	MBQCP	17	2	12.0	bin_flat.hyb	12	0.05	0.07
st_e13	MBQCP	3	3	2.0	bin_flat.nve	2	0.01	0.03
alan	MBQP	9	8	2.9252	bin_flat.nve	7	0.06	0.08
gbd	MBQP	5	5	2.2	allinone.ubs	12	0.00	0.03
st_e27	MBQP	5	7	2.0	bin_flat.hyb	4	0.14	0.16
gear3	MINLP	9	5	0.0003	allinone.nve	50	0.29	0.37
gear4	MINLP	7	2	1.6434	bin_flat.hyb	52	18.97	19.37
nvs01	MINLP	4	4	12.4696	allinone.hyb	92	1.73	1.87
nvs21	MINLP	4	3	-5.6844	onebyone.ubs	157	1.78	1.96
st_e38	MINLP	5	4	7197.7277	allinone.ubs	56	2.37	2.48

Benchmark	Type	#Var	#Con	Found objective	Solved by	#Iter	Solving time	Total time
st_e40	MINLP	5	9	30.4142	bin_flat.nve	5	0.62	0.67
tln2	MIQCP	9	13	5.3	bin_flat.hyb	8	0.06	0.08
chance	NLP	5	4	29.8946	nobb.ubs	14	0.24	0.26
ex14_1_1	NLP	4	5	0.0009	nobb.ubs	13	1.47	1.49
ex14_1_5	NLP	7	7	0.0	nobb.ubs	6	0.02	0.04
ex4_1_1	NLP	2	1	-7.4867	nobb.hyb	4	0.02	0.04
ex4_1_3	NLP	2	1	-443.6709	nobb.nve	12	0.08	0.12
ex4_1_4	NLP	2	1	0.0	nobb.nve	6	0.02	0.05
ex4_1_5	NLP	3	1	0.0	nobb.ubs	31	0.15	0.18
ex4_1_6	NLP	2	1	7.0	nobb.nve	3	0.00	0.03
ex4_1_7	NLP	2	1	-7.5	nobb.nve	5	0.02	0.04
ex4_1_8	NLP	3	2	-16.7388	nobb.nve	5	0.03	0.05
ex4_1_9	NLP	3	3	-5.5078	nobb.hyb	16	0.06	0.11
ex7_2_2	NLP	7	6	-0.3885	nobb.nve	12	123.07	123.11
ex7_3_1	NLP	5	8	0.3427	nobb.nve	60	5.57	5.67
ex7_3_2	NLP	5	8	1.0903	nobb.ubs	15	0.11	0.14
ex8_1_3	NLP	3	1	3.0	nobb.hyb	10	0.28	0.30
ex8_1_4	NLP	3	1	0.0	nobb.hyb	2	0.00	0.03
ex8_1_5	NLP	3	1	-1.0312	nobb.ubs	12	0.18	0.21
ex8_1_6	NLP	3	1	-10.086	nobb.nve	5	0.08	0.11
mathopt1	NLP	3	3	0.0	nobb.nve	18	0.09	0.12
mathopt2	NLP	3	5	0.0	nobb.hyb	2	0.01	0.02
mathopt5_4	NLP	2	1	0.0	nobb.hyb	4	0.02	0.04
mathopt5_7	NLP	2	1	-4.4365	nobb.nve	11	0.08	0.10
mathopt5_8	NLP	2	1	-0.6857	nobb.nve	6	0.03	0.05
prob09	NLP	4	2	0.0	nobb.nve	4	0.02	0.04
rbrock	NLP	3	1	0.0	nobb.ubs	42	0.19	0.24
st_e06	NLP	4	4	0.0	nobb.hyb	2	0.01	0.02
st_e11	NLP	4	3	189.3116	nobb.nve	9	1.69	1.72
st_e12	NLP	5	4	-4.5139	nobb.ubs	16	1.86	1.88
st_e17	NLP	3	2	376.2924	nobb.ubs	18	0.07	0.10
st_e19	NLP	3	3	-118.7046	nobb.ubs	22	0.18	0.20
circle	QCP	4	11	4.5751	nobb.ubs	13	3.45	3.48
ex3_1_1	QCP	9	7	7049.2485	nobb.ubs	25	0.31	0.33
ex3_1_4	QCP	4	4	-3.9997	nobb.ubs	14	0.16	0.19
ex5_2_2_case1	QCP	10	7	-399.9995	nobb.ubs	28	0.48	0.50

Benchmark	Type	#Var	#Con	Found objective	Solved by	#Iter	Solving time	Total time
ex5_2_2_case2	QCP	10	7	-599.9997	nobb.ubs	30	0.42	0.43
ex5_2_2_case3	QCP	10	7	-749.9996	nobb.ubs	30	0.47	0.52
ex5_3_2	QCP	23	17	1.8647	nobb.ubs	12	52.57	52.60
ex5_4_2	QCP	9	7	7512.2304	nobb.ubs	20	0.41	0.45
ex9_1_1	QCP	14	13	-13.0	nobb.nve	9	0.00	0.03
ex9_1_2	QCP	11	10	-16.0	nobb.nve	4	0.00	0.03
ex9_1_4	QCP	11	10	-37.0	nobb.nve	3	0.02	0.04
ex9_1_5	QCP	14	13	-1.0	nobb.hyb	4	0.00	0.02
ex9_1_8	QCP	15	13	-3.25	nobb.nve	4	0.01	0.02
ex9_2_3	QCP	17	16	0.0002	nobb.ubs	18	0.36	0.38
haverly	QCP	13	10	-399.9997	nobb.ubs	28	0.58	0.62
house	QCP	9	9	-4499.9999	nobb.ubs	32	0.98	1.03
pointpack02	QCP	6	4	1.9999	nobb.ubs	14	0.17	0.18
st_e01	QCP	3	2	-6.666	nobb.ubs	16	253.42	253.46
st_e02	QCP	4	4	201.1593	nobb.nve	2	0.04	0.06
st_e05	QCP	6	4	7049.2495	nobb.ubs	34	0.83	0.85
st_e07	QCP	11	8	-399.9997	nobb.hyb	53	205.10	205.15
st_e08	QCP	3	3	0.7419	nobb.ubs	12	0.06	0.09
st_e18	QCP	3	5	-2.8283	nobb.ubs	12	0.06	0.08
st_e30	QCP	15	16	-1.581	nobb.ubs	14	23.73	23.75
st_e33	QCP	10	7	-399.9997	nobb.ubs	28	193.08	193.14
st_e34	QCP	7	5	0.0156	nobb.nve	6	0.02	0.03
wastewater02m1	QCP	20	15	130.7026	nobb.nve	97	309.32	309.38
dispatch	QCQP	5	3	3155.2883	nobb.nve	69	38.64	38.77
ex5_2_4	QCQP	8	7	-449.9992	nobb.ubs	28	0.76	0.79
ex9_2_4	QCQP	9	8	0.5	nobb.nve	3	0.02	0.04
ex9_2_6	QCQP	17	13	-0.9992	nobb.ubs	12	0.31	0.34
ex9_2_8	QCQP	7	6	1.5	nobb.ubs	3	0.01	0.03
st_e09	QCQP	3	2	-0.499	nobb.ubs	10	0.04	0.06
ex2_1_3	QP	14	10	-14.999	nobb.hyb	33	19.71	19.77
ex2_1_4	QP	7	6	-11.0	nobb.nve	3	0.01	0.03
immun	QP	22	8	0.0	nobb.ubs	76	24.21	24.26
nemhaus	QP	6	6	31.0	nobb.hyb	2	0.01	0.02
sambal	QP	18	11	3.9686	nobb.ubs	30	219.76	219.78
st_bpaf1b	QP	11	11	-42.9625	nobb.ubs	22	0.35	0.39
st_bpk1	QP	5	7	-12.9994	nobb.ubs	18	0.13	0.14

Benchmark	Type	#Var	#Con	Found objective	Solved by	#Iter	Solving time	Total time
st_bpv1	QP	5	5	10.0	nobb.ubs	25	0.00	0.04
st_bpv2	QP	5	6	-7.9994	nobb.ubs	18	0.06	0.08
st_bsj2	QP	4	6	1.0005	nobb.hyb	16	0.40	0.41
st_bsj3	QP	7	2	-86768.5498	nobb.ubs	44	5.15	5.22
st_cqpf	QP	5	7	-2.75	nobb.hyb	4	0.01	0.03
st_cqpjk2	QP	4	2	-12.499	nobb.ubs	13	0.06	0.08
st_e22	QP	3	6	-84.9998	nobb.ubs	20	0.11	0.14
st_e25	QP	5	9	0.8908	nobb.nve	75	35.62	35.69
st_e26	QP	3	5	-185.7792	nobb.hyb	18	0.11	0.12
st_glmp_fp1	QP	5	9	10.0	nobb.ubs	18	453.69	453.75
st_glmp_fp3	QP	5	9	-12.0	nobb.hyb	4	0.01	0.02
st_glmp_kk90	QP	6	8	3.0001	nobb.ubs	16	405.05	405.09
st_glmp_kk92	QP	5	9	-12.0	nobb.hyb	6	0.04	0.05
st_glmp_kky	QP	8	14	-2.4998	nobb.nve	46	827.40	827.46
st_glmp_ss1	QP	6	12	-24.5705	nobb.nve	115	901.45	901.52
st_ht	QP	3	4	-1.5997	nobb.ubs	14	0.08	0.10
st_pan1	QP	4	5	-5.2831	nobb.ubs	16	0.44	0.48
st_ph11	QP	4	5	-11.2807	nobb.ubs	18	0.15	0.19
st_ph12	QP	4	5	-22.6246	nobb.ubs	20	0.19	0.21
st_ph13	QP	4	11	-11.2807	nobb.ubs	18	0.21	0.23
st_ph14	QP	4	11	-229.7221	nobb.ubs	26	0.56	0.60
st_ph3	QP	7	6	-420.2347	nobb.hyb	52	423.17	423.25
st_phex	QP	3	6	-84.9996	nobb.ubs	17	0.09	0.11
st_qpk1	QP	3	5	-2.9995	nobb.ubs	16	0.07	0.12
st_z	QP	4	6	0.0	nobb.nve	3	0.03	0.06
graphpart_2pm-0044-0044	BQP	49	17	-13.0	bin_flat.nve	12	178.03	178.18
fac1	MBNLP	23	19	160912612.351	bin_flat.ubs	34	9.36	9.40
elf	MBQCP	55	39	0.1916	bin_flat.hyb	20	64.50	64.54
sporttournament08	MBQCP	30	2	24.0	bin_flat.hyb	19	1035.19	1035.30
ex1263a	MIQCP	25	36	19.6	bin_flat.hyb	8	2.20	2.27
ex1264a	MIQCP	25	36	8.6	bin_flat.hyb	7	1.37	1.42
ex1265a	MIQCP	36	45	10.3	bin_flat.nve	9	1.23	1.28
tln4	MIQCP	25	25	8.3	bin_flat.hyb	6	27.37	27.43
tln5	MIQCP	36	31	10.3	bin_flat.nve	12	1532.82	1539.89
ex14_2_5	NLP	5	6	0.0002	nobb.ubs	13	7.74	7.80
ex4_1_2	NLP	2	1	-663.5	nobb.ubs	18	3.10	3.13

Benchmark	Type	#Var	#Con	Found objective	Solved by	#Iter	Solving time	Total time
linear	NLP	25	21	89.0006	nobb.nve	595	4.28	4.68
kall_congruentcircles_c41	QCP	13	25	0.8584	nobb.hyb	2	0.32	0.35
prolog	QCQP	21	23	0.0	nobb.ubs	27	1028.42	1028.49
st_qpc-m3c	QP	11	11	0.0	nobb.hyb	4	0.54	0.57
st_qpc-m4	QP	11	11	0.0	nobb.nve	3	0.62	0.64
hmittelman	BNLP	17	8	13.0	bin_flat.hyb	2	0.04	0.06
graphpart_2g-0044-1601	BQP	49	17	-954077.0	bin_flat.nve	28	773.26	773.45
ex1263	MBQCP	93	56	19.6	bin_flat.nve	12	1.67	1.69
ex1264	MBQCP	89	56	8.6	bin_flat.nve	8	0.41	0.45
nous1	MBQCQP	51	44	None	allinone.nve	1	0.01	0.03
nous2	MBQCQP	51	44	None	allinone.nve	1	0.01	0.02
hybriddynamic_fixed	MBQP	72	80	1.4737	bin_flat.nve	2	0.01	0.04
ex1266a	MIQCP	49	54	16.3	bin_flat.hyb	16	19.50	19.55
tloss	MIQCP	49	54	16.3	bin_flat.nve	19	20.15	20.41
tltr	MIQCP	49	55	48.0666	bin_flat.nve	9	8.82	8.88
fac2	MBNLP	67	34	331837498.177	bin_flat.ubs	48	1022.48	1022.65
ex1265	MBQCP	131	75	10.3	bin_flat.ubs	16	0.72	0.75
ex1266	MBQCP	181	96	16.3	bin_flat.hyb	2	3.05	3.27
autocorr_bern20-05	MBNLP	22	2	-416.0	bin_flat.nve	418	193.18	193.63
fac3	MBQP	67	34	31982309.8485	bin_flat.ubs	46	41.29	41.40
portfol_roundlot	MINLP	18	12	None	allinone.ubs	1	0.02	0.04
edgecross10-010	MBQCP	92	482	1.0	bin_flat.nve	13	1.66	1.71
edgecross10-020	MBQCP	92	482	11.0	bin_flat.ubs	24	1734.18	1734.31