Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD

AMBROS M. GLEIXNER

STEFAN HEINZ

THORSTEN KOCH

品野勇治 (YUJI SHINANO)

SCIP Optimization Suiteを利用した 混合整数(線形/非線形)計画問題の解法

Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite

Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 12-24 (July 2012)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

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

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

SCIP Optimization Suite を利用した 混合整数 (線形/非線形) 計画問題の解法

Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite*

Timo Berthold　Ambros M. Gleixner　Stefan Heinz　Thorsten Koch

品野勇治 (Yuji Shinano)

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany.

{berthold,gleixner,heinz,koch,shinano}@zib.de

概要

この論文ではソフトウェア・パッケージ SCIP Optimization Suite を紹介し，その 3 つの構成要素：モデリング言語 Zimpl, 線形計画 (LP: linear programming) ソルバ SoPlex, そして，制約整数計画 (CIP: constraint integer programming) に対するソフトウェア・フレームワーク SCIP, について述べる．本論文では，この 3 つの構成要素を利用して，どのようにして挑戦的な混合整数線形計画問題 (MIP: mixed integer linear optimization problems) や混合整数非線形計画問題 (MINLP: mixed integer nonlinear optimization problems) をモデル化し解くのかを説明する．SCIP は，現在，最も高速な MIP,MINLP ソルバの 1 つである．いくつかの例により，Zimpl, SCIP, SoPlex の利用方法を示すとともに，利用可能なインタフェースの概要を示す．最後に，将来の開発計画の概要について述べる．

1　はじめに

線形計画（*LP: linear programming*）と**混合整数線形計画（*MIP: mixed integer linear programming*）**は，オペレーションズ・リサーチ分野において，実際に最適化問題をモデル化して解くための最も重要な技術である．Dantzig が最初に LP に対する単体法を開発 [14] し，Gomory が最初に整数計画に対する切除平面法を開発 [22] し，そして，それらを統合した分枝カット法のパラダイムが開発 [29] されて以来，継続的な理論研究と計算手法の開発により，MIP 問題に対する強力な解法が利用できるようになっている [26, 30].

MIP における解法の理論面にあまり注意を向けないなら，実務家にとって，ある最適化問題を MIP 問題として定式化することには多くの利点がある：

- 多くの現実問題に対して，どのような制約条件のもとで何を最適化するのかを直接的に表現できる，あるいは，少なくとも，線形で近似した関数と整数変数により表現できる．MIP に定式化された最適化問題は，一般的に定式化に関する特別な知識がなくても容易に理解できる．
- 今では，驚くほど大規模な MIP を解くために，簡単に利用できる強力なソルバが存在する [15, 16, 1, 9, 2]. それらが，仮に現実的な時間内に最適解を求められなくても，最適解にかなり近い解が得られることが多い．

* This paper will be published in the Proceedings of the 24th RAMP Symposium held at Tohoku University, Miyagi, Japan, 27–28 September 2012, see http://orsj.or.jp/ramp/2012/.

- 分枝限定法が動作するので，実行過程で得られている暫定解と LP 緩和問題を解いて得られる緩和問題の解から，最適値に対する上下界値を更新しながら計算が進行する．よって，仮に最適解が制限時間内に求まらなくても，得られている解には精度保証があり，現実的には十分な精度で解が得られることが多い．

現実問題から定式化された MIP のインスタンスを解くことに関しては，過去数 10 年の間にめざましい進歩を遂げている．CPLEX や XPRESS のような商用ソルバ販売会社が，顧客から集めたテスト用インスタンスセットに対する，純粋にアルゴリズムだけによる加速率は，過去 20 年間で 55,000 倍にもなる [27].

最新の MIP ソルバにおける技術的な進歩は，**制約プログラミング** *(CP: constraint programming)* と**充足可能性問題** *(SAT: satisfiability problem)* の研究分野での解法技術の統合である．いくつかの最適化問題に対して，それらの単独手法では解けなかった問題が，手法を統合して解けたという例が示されてきた．例えば， [11, 25, 40] を参照されたい．結果として，一般化された MIP と CP の統合に関する種々の概念が提案されており [5, 6, 10, 24, 35, 36]，MIP ソルバは，CP と SAT の研究分野において開発されたアルゴリズムの適用を始めている．

このような統合は，有効なアルゴリズムの一部を取り入れた統合にとどまらず，概念レベルで統合した**制約整数計画** *(CIP: constraint integer programming)* のパラダイムが開発された [3]．CIP のパラダイムが意図しているのは，CP のもつ強力なモデリング力をできる限り維持しながら，主・双対問題を意識した強力な MIP の解法技術を利用することである．CIP の柔軟性と拡張性を示す一つの例は，近年の**非凸混合整数非線形計画問題** *(MINLP: nonconvex mixed integer nonlinear programming)* [7, 41] への拡張である．

本稿では，特殊ケースとして MIP 問題を含む一般的な CIP をモデル化し解くことを実現する，ソフトウエア・パッケージ SCIP Optimization Suite を紹介する．SCIP Optimization Suite は，次の 3 つのツールから構成される：

- SCIP: 高度にカスタマイズ可能な，CIP を解くことと，分枝カット価格法 (branch-cut-and-price algorithm) を実現するソフトウェア・フレームである．ソルバ単独での利用も可能である．現在，最速の非商用 MIP, MINLP ソルバである．
- ZIMPL: 数学表記に近く容易に習得可能なモデリング言語であり，SCIP と協調して動作する．
- SOPLEX: SCIP で標準として利用され，LP を解くために単独でも動作する先進的な単体法の実装である．

本稿は，次のように構成される．2 節では，SCIP, ZIMPL, SOPLEX それぞれに固有の特徴と機能を述べる．3 節では，他のソフトウェアとの種々のインタフェースとファイル・フォーマットについて概観する．4 節では，簡単な例により，実際に SCIP Optimization Suite をどのように利用するかについて説明する．5 節では，どのように SCIP Optimization Suite が入手できるかを説明する．6 節では，実際に産学協同でのプロジェクトで，これらのツールが利用された例を示す．7 節では，まとめとして SCIP Optimization Suite の将来の開発の方向性について示す．

2 提供される機能

2.1 SCIP

SCIP *(Solving Constraint Integer Programs)* は，MIP, MINLP,CIP に対する分枝カット価格法を実現するためのソフトウェア・フレームワークである．特に，SCIP は他のソフトウェアに組み込んで利用できるソフトウェア・ライブラリであるとともに，対話シェルを持つソルバとして単独で利用することもできる．SCIP が扱えるファイル・フォーマットは多く，最も標準的な MIP の入力ファイルフォーマットである，`lp` フォーマット，あるいは `mps` フォーマットは，もちろん読み込み可能である．SCIP が目標としているのは，MIP, MINLP, CP の利点を統合し，それらの弱点を補い合うソルバの実現である．

■Plug-in **ベースのアーキテクチャ**　SCIP は，分枝カット価格法のアルゴリズムの構成要素となる機能が，モジュール化された設計になっている点が最大の特徴である．その設計により，汎用ソルバとしての技術と特定の問題専用ソルバとしての技術を自由に組合せて使うことができる．

SCIP で中心的な働きをするのは，*constraint handler* である．変数の定義域を参照することで，constraint handler は，実行可能解の探索領域を制限する．constraint handler は制約のタイプ毎に用意され，整数性判定のための constraint handler, 線形制約式のための constraint handler, 非線形制約式のための constraint handler, 論理式のための constraint handler, 組合せ制約のための constraint handler 他，多くの constraint handler が用意されている．一つの constraint handler は，与えられた解が，その handler が扱うタイプの制約を満たしている，つまり，制約のタイプに対して実行可能であるか否かを判定しなければならない．与えられた解に対する実行可能性の判定ができれば，解を全列挙する列挙解法が構成できる．問題を解く速度を加速するために，constraint handler は，制約式特有の前処理，変数の定義域の伝播，切除平面の生成などのアルゴリズムの実装ができる．

分枝変数の選択方法の指定，変数の定義域の伝播，実行可能解生成のためのヒューリスティック解法などを plug-in として追加することで，より効率的な解法が構成できる．標準で提供される SCIP 自体のアルゴリズムの多くも，plug-in として提供されており，SCIP 3.0 には，126 の plug-in が用意されている．SCIP の主たる機能は，それらの plug-in の実行順序と，plug-in 間のデータの受け渡しを管理することである．SCIP では，多くのパラメタによって，ユーザが解法を可能な限り制御できるようになっている．

配布されている標準 SCIP には，26 の constraint handler が用意されている．用意されている constraint handler の一部を利用するだけで，次のパラグラフで示すような問題のクラスを表現し解くことができる．図． 1 に，異なるクラスの問題間の関係を示す．

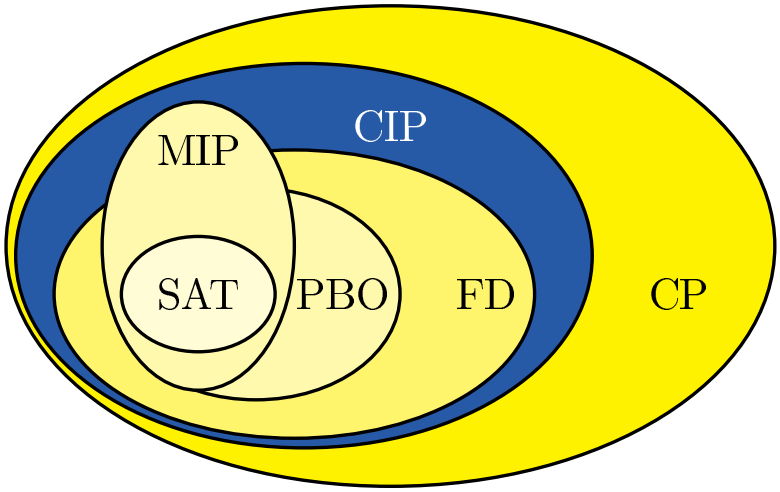

図 1　数理最適化問題のクラスの包含関係

■LP, MIP, MINLP.　一般の線形制約を扱う linear constraint handler の他に，SCIP は，線形制約の特殊ケースであるナップサック制約，変数の上下界制約 (variable bound constraints)，集合パッキング制約，集合分割制約，集合被覆制約などに対する constraint handler を用意している．これらの特殊ケースに対する constraint handler が用意されることで，その特殊性による特徴を生かした効率的なアルゴリズムが実装できる．LP は，これらの線形制約を集めたものとして表現され，MIP はいくつかの変数に整数制約を付加した問題として表現される．さらに，SCIP は数理計画問題に現れる特殊な制約として，indicator 制約と special ordered sets(SOS) の タイプ 1 とタイプ 2 をサポートしている．

解法の性能を向上させるために，他の多くの MIP 固有の plug-in が用意されている．特に顕著なものは，主問題に対するヒューリスティック解法である，丸めヒューリスティック，RINS, feasiblility pump，そして一般的な切除平面である，Gomory カット, c-MIR カット, 強 Chvatal-Gomory カット，flowcover カット, MCF カット, クリークカット等である．

MINLP は，線形制約，非線形制約，整数制約でモデル化される．式木（expression tree) として与えられる一般的な非線形関数のサポートの他に，SCIP は，二次制約，二次錐制約，二変数非線形制約，絶対値のべき乗制約を標準でサポートする．種々の MIP 固有の plug-in も，MINLP の解法中で利用可能となっている点に注意されたい．

■PBO, SAT, スケジューリング． PBO (pseudo-boolean optimization problems）は，線形制約，論理積制約，整数制約によりモデル化され解かれる．SAT (satisfiability problems) は，論理和制約と整数制約によりモデル化され解かれる．スケジューリング問題は，CP の流儀で，linking 制約と累積制約 (cumulative constraint) でモデル化され解かれる．

■分枝価格法． SCIP が広く利用される理由の１つは分枝価格法を実現できることである．SCIP を特定のクラスの問題に対する分枝価格法ソルバとしてカスタマイズするためには，ユーザは単に *variable pricer* plug-in を追加すれば良い．それだけで，分枝木の管理は全て SCIP において行われる．この特徴は，他の全ての商用 MIP ソルバにない SCIP の利点である．商用の MIP ソルバでは，コールバックルーチンで局所的に変数を追加することは SCIP ほど簡単ではない．

■性能． SCIP は，制約整数計画 (constraint integer programming) の枠組みをサポートし，より一般的な問題を扱えるが，計算速度の点でも商用，非商用の MIP ソルバと競える性能である [27]. 第三者による性能比較において，SCIP は，しばしば最速の非商用 MIP ソルバ [33] であり，PBO ソルバ [31] である．加えて，現在，SCIP は最速の非凸 MINLP ソルバ [41] である．

2.2 ZIMPL

最適化問題のモデルを構築する際，最初に行う最も本質的な選択は，どのようなクラスの最適化問題として定式化するかである．この選択によって，どの程度詳細に制約を定式化に加えるかといったことや，利用可能な解法を決定し，結果として与えられた時間内に解くことが期待できる問題のサイズが決まる．しかし，MIP が解ける時間を見積もることは極めて困難な上，解ける時間は定式化の仕方にも強く依存する（[4] 参照）ので，定式化した MIP が解けるかどうかには注意する必要がある．代数的なモデリング言語を使うと，最終的に満足のゆく定式化を決定するまでの間，簡単にモデルが変更できるので，定式化のプロセスがおどろくほど楽になる．

ここでは，モデリング言語 ZIMPL (Zuse Institute Mathematical Programming Language) を簡単に概観し，その目標と特徴を紹介する．ZIMPL は，数理計画問題を記述するための強力な言語であり，その記述を `lp`,`mps` のような LP, MIP の標準的なファイル形式に変換するための道具である．ZIMPL は SCIP と強く結びついており，SCIP を利用して最適化問題を解く際に，LP, MIP, MINLP としてモデル化するための主たる問題記述言語である．

ZIMPL の特徴は以下である．

- 数学表記に近い明確な構文を持つ.
- 学ぶのは容易ですぐに覚えられる.
- 定式化部分と具体的な係数などのデータ部分を明確に分けられる.
- 安定している.
- LGPL ライセンスのもと C 言語によるソースコードがフリーで入手できる.
- 移植性が高い (Linux, Windows, MacOS-X, Solaris, . . .).
- ソルバに依存しない.

- 呼び出し可能ライブラリとして利用できる．
- 単独でも利用でき，ソルバとリンクして使うこともできる．
- モデル構築時には有理数を使って記述することで丸め誤差を避けられる．

■モデル記述言語の歴史と現在の流行． 代数的なモデリング言語の開発は，1980 年代に GAMS [8, 12] や AMPL [18, 19] のようなシステムにより始まった．これらのツールにより，LP は数学表記に近い形式で記述でき，自動的にソルバが処理できるファイル形式に変換，あるいは，直接的に適当なソルバで処理できさえするようになった．

あいにく，今日存在するモデリング言語の多くは商業製品である．それらの製品の中で，その言語を拡張するためのソースコードが提供されているものはない．仕事仲間にさえ渡すことができない上，授業での利用は機能が制限された “学生版” を除いて認められていない．通常，限られた数の OS とアーキテクチャがサポートされ，ときにはマイクロソフト Windows だけをサポートしている場合もある．

対照的に，ZIMPL は LGPL ライセンスのもとソースコードがフリーで提供されている．このように提供されるため，学生は自分のノート PC や自宅のコンピュータで利用でき教育に便利であると同時に，自由なライセンス体系のため企業とのプロジェクトにおいての利用にも便利である．ZIMPL の開発が始まった 1999 年以来，状況が改善していることは指摘しておきたい．現在では，少なくともいくつかのソースが公開されたモデリング言語が存在する．その一つは，AMPL のサブセットを持つ GNU MATHPROG 言語であり，GNU の GLPK [21] に付属して配布されている．

商業製品としてのモデリング言語の現在の流行は，データベース問い合わせツール，ソルバ，報告書生成ツールそしてグラフィカル・ユーザインタフェースを 1 つのモデリングシステムに統合したものである．このようなシステムは，数理計画モデルの構築を，完全にグラフィカルな（商用）アプリケーション上で行うことさえ可能にする．現在，フリーで利用できるモデリング言語は，この点に関しては競争相手にならない．ZIMPL は，AMPL の機能の約 20% 程度しか実装されていないかもしれない．それでも，その最も重要な 20% で，多くの現実世界におけるプロジェクトでは十分であった．その上，ZIMPL のユーザマニュアルは約 25 ページである．以下で紹介するように，ZIMPL は 2 〜 3 時間で学べ，グラフィカルなモデリングシステムと違い，ZIMPL によるモデルは，まだ十分に数学表記に近い．

■構文． 一般に，各 ZIMPL によるモデルは，6 つのタイプのステートメントからなる:

- sets（集合），
- parameters（パラメタ），
- variables（変数），
- objective（目的），
- constraints（制約），
- function definitions（関数定義）．

ZIMPL のステートメントは，モデル中の既存部分は決して変えず，単にモデルに追加するだけである．これにより，ZIMPL のモデルを理解するのはとても容易になる．

LP, MIP と異なり，ZIMPL は多項式表現を扱えるので，SCIP との親和性はとても良い．なぜなら，SCIP は今では非凸二次制約と多項式制約を持つ整数計画問題が解ける [7, 41] からである．

■有理数演算． ZIMPL 固有の特徴として，有理数演算の利用がある． 2， 3 の例外を除くと，ZIMPL 内における全ての演算は無限の精度を持つ有理数として計算され，定式化ファイルやソルバへの入力データへ翻訳の際に丸め誤差が生じないことを保証している．これは，GNU Multi Precision Library[*1] を利用することで達成している．

[*1] http://www.gmplib.org

■自動再定式化. もう一つの特徴は，引数に決定変数を持つ関数を自動的に複数の制約式へ変換する機能である．これらの関数の引数は，上下限のついた整数変数か，0-1 変数で，関数はそれらの線形関数でなければならない．たとえば，ある変数の値に依存した制約式を持つことや，ある変数の絶対値の値の計算を可能としている．

■ソフトウェア工学. ZIMPL 開発の間，ソフトウェア工学に特別な注意が向けられてきた．ZIMPL は，テストツールによりプログラムコードの 80% 以上のコードが実行されて初めてリリースされてきている．特に，カバーしているコードは，誤ったモデル記述をユーザに通知するエラーメッセージや警告メッセージのコードを含む．assert 命令は，実行時の前提条件と不変な表明を確認するために広範囲に渡って使われている．Makefile には，ターゲットとして valgrind[*2] チェックの実行が用意されており，プログラムコードは，常にスタティックなプログラムチェッカーである flexelint[*3]にかけられている．概して言えば，結果として高品質で頑健なコードとなっており，そのようなコードでないと，企業との現実プロジェクトでは利用できない．

■ZIMPL の適用例. ZIMPL は，種々の遠距離通信における施設配置計画，チップデザインにおける三次元スタイナー木パッキング，トラックの競売，タンパク質フォールディングなど，（継続的に）多くの現実問題や教育上の問題をモデル化するために使われてきている．また，多くの大学の授業や Combinatorial Optimization at Work(http://co-at-work.zib.de 参照) のような集中講義で使われている．

2.3 SOPLEX

SOPLEX は，線形計画（LP ）を解くための改訂単体法の先進的な実装であり，前処理，疎性の利用，主・双対単体法のルーチンを持つ．SOPLEX は，`mps` 形式，または `lp` 形式のファイルを読み込んで動作する独立したソルバとして利用できる．また，C++ クラスライブラリとして，他のソフトウェアに組み込んで利用することもできる．SCIP では，標準 LP ソルバとして SOPLEX が利用される．ここでは，SOPLEX の主な特徴を示す．より詳細に関しては，[42] を参照されたい．

■前処理. SOPLEX では，単体法を動作させる前に，問題のサイズを縮小するために種々の前処理が動作する．1 変数だけの制約式や線形従属な制約式の削除などである．これらの処理により解法の実行速度が加速する．数値計算における丸め誤差の影響を少なくするために，制約行列のスケーリングが行われる．

■主・双対単体法. SOPLEX は，主単体法と双対単体法が自動的に切り替わる単体法の実装となっている．ユーザは，最初に実行するアルゴリズムを指定することもできる．数値的な安定性と実行可能な初期基底を生成するために，一時的に変数と制約式に対する上下界値をシフトする．これらは，明示的な “フェーズ I” の処理を必要としないことを示す．

■線形代数演算. 線形代数演算においては，多くの LP のデータにみられる制約行列，右辺項，目的関数での疎性をできる限り利用する．ピボッティングの間，疎 LU 分解の更新は，Forrest-Tomlin または Eta [17, 39] により行われる．

SOPLEX 固有の特徴として，標準的な列を基本とした計算（デフォルト設定）の他に，基底の行表現とよばれる形式を利用することができる．変数よりも制約式が多いような LP のインスタンスに対しては，行表現形式は基底行列の次元を小さくでき，連立一次方程式が高速に解ける．

■高精度解法. SOPLEX は浮動小数点演算による LP ソルバであり，標準では計算に倍精度浮動小数点型の数が使われている．特に数値的に不安定なインスタンスを扱う際には，long double 型の数を利用することができる．

[*2] http://www.valgrind.org
[*3] http://www.gimpel.com

表 1　SCIP が直接読めるファイル形式の一部

ファイル拡張子	説明
lp	LP と MIP の入力ファイル形式の 1 つで可読性が高い
mps	LP と MIP の伝統的な標準入力形式
cnf	SAT (SATisfiability problems) のためのファイル形式
opb	PBO （Pseudo-Boolean Optimization problems) のためのファイル形式
wbo	重み付き PBO のためのファイル形式
zpl	ZIMPL により記述された際のファイル形式
fzn	制約プログラミングのモデル化言語である FlatZinc で記述された際のファイル形式
pip	多項式混合整数計画問題のためのファイル形式
cip	SCIP の各 constraint hander によって定義される，SCIP の標準ファイル形式
osil	線形計画/非線形計画をサポートする XML ベースのファイル形式

他の LP ソルバとの大きな違いは，version 1.7 から，新しい技術である反復的高精度化 [20] が実装されている．この技術では，任意精度で解を計算するために，GNU Multiple Precision Library　を利用して，浮動小数点演算を繰り返しながら解く．

■LP ベースの分枝カット法における SoPlex.　SoPlex は SCIP の標準 LP ソルバであり，SCIP とともに継続的にテストされ改善されている．その結果，SoPlex は，最も頑健な非商用 LP ソルバとなり，互いにわずかな違いしかない LP インスタンスを繰り返し解く必要がある場合には，特に適している．

3　インターフェース

SCIP Optimization Suite を他のソフトウェアパッケージやプログラミング環境から利用する方法はいくつかある．

3.1　ファイル形式

SCIP へ問題のインスタンスを読み込む最も簡単な方法は，SCIP が直接読むことができるファイル形式によりインスタンスを与えることである（4.2.1 も参照されたい）．SCIP 3.0 は，非線形計画問題や制約プログラミングで利用されているファイル形式を含む，10 以上の異なったファイル形式を読むことができる．多くのファイル形式が扱えるおかげで，異分野研究コミュニティの研究者が，初めて SCIP Optimization Suite を使う際に敷居が低い．表 1 に，SCIP において最もよく使われるファイル形式を示す．

3.2　モデリング言語と Matlab インタフェース

最適化問題は，モデリング言語を使って定式化するのが最も自然である．ZIMPL の他にも，直接 SCIP とのインターフェースを持ついくつかのモデリング・ツールが存在する．その中には，制約プログラミングのモデリング言語である COMET [32] と非線形最適化問題のモデル化に適した GAMS [12] も含まれる．

SCIP 3.0 では，初となる Matlab インターフェースのベータ版が含まれる．それにより，Matlab の行列とベクトル型により定義される MIP と LP が解けるようになる．

3.3 Python インタフェース

第三者により，SCIP Optimization Suite へのいくつかの Python インタフェースが提供されている．例えば，NUMBERJACK [23] と PYTHON-ZIBOPT [34] である．NUMBERJACK は Python で実装された制約プログラミングのためのプラットフォームであり，多くの異なったソルバをサポートしており，その 1 つとして SCIP Optimization Suite も利用できる．PYTHON-ZIBOPT は，Ryan J. O'Neil により開発された SCIP Optimization Suite に対する Python 拡張である．

3.4 Java と C++ インタフェース

SCIP は C 言語により書かれているので，ライブラリは直接的に C++ 言語から利用できる．もし，ユーザが自分自身で C++ 言語により plug-in を開発したい場合，標準で配布される SCIP の `src/objscip` ディレクトリに全ての異なる plug-in に対する C++ 言語で書かれたラッパークラスが用意されている．また，SCIP 3.0 では，SCIP ライブラリの基本的な関数に対する JNI による Java インタフェースが初めて提供される．

4 利用方法の例

4.1 ZIMPL

ここでは，ZIMPL モデルをどのように構築するかについて説明する．構築された ZIMPL モデルは，直接的に，あるいは，`lp` または `mps` ファイルへ変換して SCIP へ読み込むことができる．例として，[37, Section 58.5] などに掲載されている**対称巡回セールスマン問題** *(TSP: symmetric Traveling Salesman Problem)* に対する指数本の制約式からなる定式化を取り上げる．

$G = (V, E)$ を完全グラフとする．ここで，V は都市の集合であり，E は都市間を結ぶ枝の集合である．各枝 $(i, j) \in E$ に対して，d_{ij} を枝間の移動距離とし，巡回路がその枝を利用するとき 1，そうでないとき 0 となる $0-1$ 変数 x_{ij} を導入することで，TSP の定式化は以下となる：

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in E} d_{ij} x_{ij} & \\
\text{subject to} \quad & \sum_{(i,j)\in \delta_v} x_{ij} = 2 & \text{for all } v \in V, \\
& \sum_{(i,j)\in E(U)} x_{ij} \leq |U| - 1 & \text{for all } U \subseteq V, \emptyset \neq U \neq V, \\
& x_{ij} \in \{0, 1\} & \text{for all } (i, j) \in E.
\end{aligned}
$$

ファイルから読み込まれるデータは，都市と各都市の位置を示す xy 座標である．ここでは，距離はユークリッド距離を仮定する．以下に例を示す：

```
# City       X    Y
Berlin     5251 1340
Frankfurt  5011  864
Leipzig    5133 1237
Heidelberg 4941  867
```

```
Karlsruhe  4901  840
Hamburg    5356  998
Bayreuth   4993 1159
Trier      4974  668
Hannover   5237  972
```

```
Stuttgart  4874  909
Passau     4856 1344
Augsburg   4833 1089
Koblenz    5033  759
```

ZIMPL の構文を示すために，以下に ZIMPL による定式化を示す（以下が `tsp.zpl` ファイルに記述されているものとする）．上記データが入力ファイル `tsp.dat` に記述されているものとし，そのデータが，頂点集合 `V` と座標パラメタ `px` および `py` の初期化に利用されている．

```
set V              := { read "tsp.dat" as "<1s>" comment "#" };
set E              := { <i,j> in V * V with i < j };
set P[]            := powerset(V);
set K              := indexset(P);

param px[V]        := read "tsp.dat" as "<1s> 2n" comment "#";
param py[V]        := read "tsp.dat" as "<1s> 3n" comment "#";

defnumb dist(a,b) := sqrt((px[a]-px[b])^2 + (py[a]-py[b])^2);

var x[E] binary;

minimize cost: sum <i,j> in E : dist(i,j) * x[i, j];

subto two_connected: forall <v> in V do
   (sum <v,j> in E : x[v,j]) + (sum <i,v> in E : x[i,v]) == 2;

subto no_subtour:
   forall <k> in K with
      card(P[k]) > 2 and card(P[k]) < card(V) - 2 do
      sum <i,j> in E with <i> in P[k] and <j> in P[k] : x[i,j]
      <= card(P[k]) - 1;
```

ユーザは，この zpl ファイルを直接 SCIP の入力とすることができる．または，zimpl tsp.zpl コマンドを使って，lp ファイルを生成することもできる．あるいは，zimpl -t mps tsp.zpl コマンドを使って，mps ファイルを生成するこもできる．コマンドラインで利用できるオプションは， zimpl -help により表示できる．

ZIMPL による定式化で，P[] が都市の部分集合の全てを持っていることに注意されたい．このため，このモデルが解ける都市の数は極めて限られている．13 都市で，LP は 78 変数，8 021 制約，154 596 の非ゼロ要素をもつ制約行列となる．どのようにして，より大規模な TSP を解くかに関しては，CONCORDE の WEB ページ[*4]を参照されたい．上記データにおける最適巡回路は，Berlin, Leipzig, Passau, Augsburg, Stuttgart, Heidelberg, Karlsruhe, Trier, Koblenz, Frankfurt, Hannover, Hamburg, Berlin であり，標準的に利用されているようなコンピュータなら SCIP により 1 秒未満で解ける．

正確な ZIMPL の構文と ZIMPL によるモデル化の例をさらに知りたい場合は，http://zimpl.zib.de にあるマニュアルを参照されたい．

4.2 SCIP

ここでは，SCIP の利用方法について簡単に紹介する．紹介する内容は，対話シェルの利用方法と呼び出し可能ライブラリの利用法の初歩である．

4.2.1 対話シェルの利用方法

SCIP を使い始めるには，まず単なる MIP, MINLP あるいは他の最適化問題ソルバとして，対話シェルを試すのが良い．コンパイル済みの SCIP の実行形式ファイル[*5]と試すための問題ファイルは必要である．SCIP は，LP, MPS, ZPL, WBO, FZN, PIP などのフォーマット（表 1 参照）で書かれた問題が読める．MIP のテスト用のインスタンスは，例えば，MIPLIB のページ[*6]のインスタンスを使えば良い．この節では， MIPLIB 3.0 にある stein27 を例として利用する．

[*4] http://www.tsp.gatech.edu
[*5] http://scip.zib.de/download.shtml から，Linux, Mac, Windows 用の実行形式ファイルがダウンロードできる
[*6] http://miplib.zib.de，stein27 のファイル http://miplib.zib.de/miplib3/miplib3/stein27.mps.gz

■問題の読み込みと最適化の実行. SCIP の実行形式ファイルは引数を付けずに実行しても動き，対話シェルが開く．“help” コマンドにより利用可能なコマンドが表示される．ブラケット (<>) はサブ・メニューがあることを示す．

```
SCIP> help

  <display>             display information
  <set>                 load/save/change parameters
...
  read                  read a problem
```

最初に知るべき最も大事なコマンドは，問題ファイルを読む “read <path/to/file>” と，読み込んだ最適化問題を解くことを指示する “optimize”，そして解いた結果の最良解（実行が完了していない場合を含めているので最良解）を表示する “display solution” である．解は，デフォルト設定では非ゼロ要素の値を持つ変数だけを表示する．

```
  SCIP> read check/instances/MIP/stein27.mps.gz
  original problem has 27 variables (27 bin, 0 int, 0 impl, 0 cont) and 118 constraints
  SCIP> optimize

  presolving:
  (round 1) 0 del vars, 0 del conss, 0 chg bounds, 0 chg sides, 0 chg coeffs, 118 upgd conss, 0 impls, 0 clqs

    time | node  | left  |LP iter|LP it/n|mdpt |frac |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
  t 0.0s|     1 |     0 |    34 |     - |   0 |  21 |  27 | 118 |   0 |   0 |   0 | 1.300000e+01 | 2.700000e+01 | 107.69%
  R 0.0s|     1 |     0 |    34 |     - |   0 |  21 |  27 | 118 |   0 |   0 |   0 | 1.300000e+01 | 2.600000e+01 | 100.00%
  s 0.0s|     1 |     0 |    34 |     - |   0 |  21 |  27 | 118 |   0 |   0 |   0 | 1.300000e+01 | 2.500000e+01 |  92.31%
    0.0s|     1 |     0 |    44 |     - |   0 |  21 |  27 | 120 |   2 |   0 |   0 | 1.300000e+01 | 2.500000e+01 |  92.31%
...
    0.1s|     1 |     2 |   107 |     - |   0 |  24 |  27 | 131 |  13 |   0 |  24 | 1.300000e+01 | 1.900000e+01 |  46.15%
  R 0.1s|    14 |    10 |   203 |   7.4 |  13 |   - |  27 | 124 |  13 |   0 | 164 | 1.300000e+01 | 1.800000e+01 |  38.46%
    0.1s|   100 |    54 |   688 |   5.9 |  13 |  20 |  27 | 124 |  13 |   0 | 206 | 1.300000e+01 | 1.800000e+01 |  38.46%
...
  SCIP Status        : problem is solved [optimal solution found]
  Solving Time (sec) : 0.73
  Solving Nodes      : 4192
  Primal Bound       : +1.80000000000000e+01 (283 solutions)
  Dual Bound         : +1.80000000000000e+01
  Gap                : 0.00 %

  SCIP> display solution

  objective value:                                 18
  x0001                                             1   (obj:1)
  x0003                                             1   (obj:1)
...
```

■出力の解析. SCIP が問題を解き始めると，まず数回，前処理を行い，その後，分枝限定法を実行する．問題が解き終わるか，前もって定義した終了条件（計算を終了する制限時間や分枝ノード数が指定可能）に達するか，ユーザが計算を中断すると，解法の実行状況に関する短いメッセージが表示される．

例のインスタンスの場合，前処理は極めて短く，単に線形制約式が，ナップサック制約などのより特殊な制約式のタイプへアップグレードしただけである．1 回の前処理の結果について 1 行が表示され，最後に簡単な総括メッセージが表示される．次に，解法が動作している過程に関する情報が表示される．上記の例では，‘t’, ‘R’, ‘s’ の文字で始まる最初の 3 行は，暫定解が主問題に対するヒューリスティック解法によって発見されたことを示す．この暫定解により，“primalbound” の値が，27 から 25 に減っていることがわかる．

4 行目の “row” の値が増えたことから，2 つの “カット” が追加されたことがわかる．そこまでに， SCIP （正確には SoPlex) は，“LP iter”(ピボット）を 44 回行っている：最初の LP を解くために 34 回，カットを追加後にさらに 10 回である．その少し後に，ルートとなる問題の計算が終わっている． “left” の列に 2 つのノード（部分問題）があることにより分枝が確認できることに注意されたい；分枝は，緩和問題の解の整数変数の内，小数値となった 24 の変数の中の 1 つで分枝していることを “frac” 列が示している．これより後は，100 ノード（部分問題）実行されるたび，あるいは，暫定解が見つかった際（上の例では 14 ノード目）に，メッセージが出力される．

SCIP の最適化処理が終了した後，あるいは，他の理由で停止した後，“display solution” コマンドにより，解で非ゼロの値を持つ変数とその値を表示できる．

SCIP の性能は，厳密にはコンピュータのアーキテクチャや動作している OS などによって変わる．よって，インストールした環境によって，計算時間や解いたノード数は多少増減する．また，このインスタンスは，同じ最適値 18 を持つ，2000 以上の異なる最適解（つまり変数の値の異なる解）を持つので，計算時間や解いたノード数の増減は自然である．このような振る舞いは，performance variability と呼ばれる．この振る舞いに興味を持たれたら，[27] を参照されたい．

■詳細な統計情報の表示． 解法の実行過程に関する統計情報，インスタンスに関する情報，そして，ソルバの各構成要素に関する，より詳細な情報にアクセスすることは，もちろん可能である．ある種の plug-in（例えば，主問題に対するヒューリスティック解法，分枝変数選択，カット生成など）に対する情報は，“display <plugin-type>” コマンドにより取り出せる．解法実行過程における統計情報に関しては，“display statistics” により取り出せる．解いているインスタンスに関する情報に関しては，“display problem” によって取り出せる．

```
SCIP> display heuristics
 primal heuristic     c priority freq ofs  description
 ----------------     - -------- ---- ---  -----------
...
 rounding             R    -1000    1   0  LP rounding heuristic with infeasibility recovering
 shifting             s    -5000   10   0  LP rounding heuristic with infeasibility recovering also using continuous variables
...
SCIP> display statistics
...
  oneopt           :       0.01          4          1
  coefdiving       :       0.02         57          0
...
  primal LP        :       0.00          0          0       0.00          -
  dual LP          :       0.20       4187      14351       3.43   71755.00
...
```

上の例において，rounding と shifting は，最適化過程の最初の部分で解を生成した主問題に対するヒューリスティック解法である．それらは，出力行の先頭文字 ‘R’ と ‘s’ により知ることができる．上の例の出力における “freq” 列の数値から，rounding が各ノードで呼び出され，shifting が分枝木の深さが 10 毎に呼び出されることがわかる．

統計情報は，極めて多いので，ここでは 2,3 行について説明する．全ての計算過程における全ての種類の plug-in に関する情報が表示される．省略している部分は除き，上に示された情報から，oneopt ヒューリスティックは 4 回呼ばれて 1 個の解を生成し，coefdiving は 57 回呼ばれたが，すべて解の生成には失敗していることがわかる．また，全ての LP が双対単体法で解かれたことがわかり，約 0.7 秒の最適化過程の内，0.2 秒が LP を解く事に費やされたことがわかる．

■パラメタの変更． 1 つの構成要素（plug-in) の性能に関する情報が表示されることで，ユーザはそれらの振る舞いをパラメタを使って変えたいかもしれない．例として，最適化過程で，rounding と shifting ヒューリスティック解法を利用しないことにすると，次のような実行となる．

```
SCIP> set
...
  <heuristics>         change parameters for primal heuristics
...

SCIP/set> heuristics
...
  <shifting>           LP rounding heuristic with infeasibility recovering also using continuous variables
...

SCIP/set/heuristics> shifting

  <advanced>           advanced parameters
  freq                 frequency for calling primal heuristic <shifting> (-1: never, 0: only at depth freqofs) [10]
  freqofs              frequency offset for calling primal heuristic <shifting> [0]

SCIP/set/heuristics/shifting> freq
current value: 10, new value [-1,2147483647]: -1
heuristics/shifting/freq = -1

SCIP> se he rou freq -1
heuristics/rounding/freq = -1

SCIP> re check/instances/MIP/stein27.mps
original problem has 27 variables (27 bin, 0 int, 0 impl, 0 cont) and 118 constraints
SCIP> o

feasible solution found by trivial heuristic, objective value  2.700000e+01
...
z 0.1s|     3 |     4 |   140 |  10.5 |1060k|   2 |  22 |  27 | 118 |  27 | 123 |  14 |   0 |  66 | 1.300000e+01 | 1.900000e+01 |  46.15%
```

```
...
SCIP Status        : problem is solved [optimal solution found]
Solving Time (sec) : 0.75
Solving Nodes      : 4253
Primal Bound       : +1.80000000000000e+01 (287 solutions)
Dual Bound         : +1.80000000000000e+01
Gap                : 0.00 %

SCIP>
```

一つずつ説明すると，まず，実行するヒューリスティックのパラメタを変えるため，“set”コマンドを実行し，“heuristics” を選択し， その後 “shifting”を選択する．すると，このヒューリスティック解法に対するパラメタが表示される（サブ・メニュー “advanced” を選択すると，さらに詳細なパラメタ設定が可能となる）ので，呼び出し頻度（calling frequency) の値に-1 を設定することで，呼び出さないようにすることができる．もし，ユーザが，最も深いサブ・メニューまでのパスを事前に知っているなら，ユーザは直接そのパスをタイプして移動できる（上の例では，rounding ヒューリスティックの設定例でこの移動を適用している)．ユーザはパラメタ名を正確に全てタイプする必要はない．そのパラメタ名が一意に識別できるだけの文字をタイプすれば移動できる．二度目に問題を解く際には，ユーザは問題データを再度読み込み最適化を指示しなければならない．

パラメタチューニングを容易にするため，SCIP は，前処理（presolving)，主問題に対するヒューリスティック解法（heuristics)，カット生成 (separation), 探索過程 (search process) に対して，いくつかのメタ・パラメタが用意されており，`emphasis` メニューから設定できる．FAQ のページも同時に参照されたい．ここでは，主問題に対するヒューリスティック解法に関するメタ・パラメタの設定例を取り上げる：

```
SCIP> set default
reset parameters to their default values
SCIP> set heuristics emphasis

  aggressive            sets heuristics <aggressive>
  fast                  sets heuristics <fast>
  off                   turns <off> all heuristics

SCIP/set/heuristics/emphasis> aggr
heuristics/veclendiving/freq = 5
...
heuristics/crossover/minfixingrate = 0.5
SCIP> read check/instances/MIP/stein27.mps
original problem has 27 variables (27 bin, 0 int, 0 impl, 0 cont) and 118 constraints

SCIP> opt
...
D 0.1s|     1 |     0 |   107 |     - | 971k|   0 |  24 |  27 | 122 |  27 | 131 |  13 |   4 |   0 | 1.300000e+01 | 1.800000e+01 |  38.46%
  0.1s|     1 |     0 |   107 |     - | 971k|   0 |  24 |  27 | 122 |  27 | 131 |  13 |   4 |   0 | 1.300000e+01 | 1.800000e+01 |  38.46%
  0.1s|     1 |     0 |   119 |     - |1111k|   0 |  24 |  27 | 122 |  27 | 132 |  14 |   4 |   0 | 1.300000e+01 | 1.800000e+01 |  38.46%
  0.1s|     1 |     2 |   119 |     - |1112k|   0 |  24 |  27 | 122 |  27 | 132 |  14 |   4 |  24 | 1.300000e+01 | 1.800000e+01 |  38.46%
 time | node  | left  |LP iter|LP it/n| mem |mdpt |frac |vars |cons |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
  0.2s|   100 |    59 |   698 |   5.8 |1138k|  14 |  11 |  27 | 122 |  27 | 123 |  14 |   4 | 204 | 1.300000e+01 | 1.800000e+01 |  38.46%
  0.2s|   200 |    91 |  1226 |   5.6 |1155k|  14 |   - |  27 | 122 |  27 | 123 |  14 |   4 | 207 | 1.300000e+01 | 1.800000e+01 |  38.46%
^Cpressed CTRL-C 1 times (5 times for forcing termination)

SCIP Status        : solving was interrupted [user interrupt]
Solving Time (sec) : 0.32
Solving Nodes      : 216
Primal Bound       : +1.80000000000000e+01 (283 solutions)
Dual Bound         : +1.30000000000000e+01
Gap                : 38.46 %

SCIP>
```

この例では，まず，`set default` により全てのパラメタをデフォルト設定に戻している．次に，主問題に対するヒューリスティック解法におけるメタ・パラメタとして，“aggressive” を選択し，多くのヒューリスティックがより頻繁に動作するような設定に変更している．SCIP は，どのパラメタがどのように変更されたのかを表示する．この設定では，デフォルト設定では利用されない設定になっていたヒューリスティック解法が設定されたため，最適解がルートノードの処理中で生成されている．200 ノードの処理を終えたところで，ユーザが `CTRL-C` を押すことで，最適化処理を中断している．その結果，計算途中の最適値に対する上下界値と相対誤差を示す，簡易バージョンの結果出力が表示されている．問題はまだ完全に解かれてはいない．それでも，ユーザは暫定解を表示したり出力でき，パラメタの値の変更も可能である．その後，`optimize` コマンドを実行することで，中断時点からの計算を再開できる．

■ファイル出力． SCIP では，様々な情報をファイルに出力できる．例えば，暫定解をファイルに出力でき，問題のデータを異なったフォーマットの問題としてファイルへ出力できる（MPS フォーマットと比べると，LP フォーマットは可読性の高いフォーマットである．SCIP は，多くのフォーマットをサポートしているので，フォーマットの変

換にも有効なツールである).

```
SCIP> write solution stein27.sol

written solution information to file <stein27.sol>

SCIP> write problem stein27.lp
written original problem to file <stein27.lp>

SCIP> q
...
```

4.2.2 呼び出し可能ライブラリの利用方法

SCIP を利用する次のステップとして呼び出し可能ライブラリの利用がある．これは，ユーザ自身のプログラム中で SCIP を利用するということであり，ユーザの嗜好に応じて C 言語または C++ 言語で書かれたプログラム中で利用できる．ユーザは，SCIP のデータ・オブジェクトを生成し，C 言語で書かれている SCIP ライブラリの公開関数を呼び出すことで，SCIP のデータ・オブジェクトの内容を変更したり，データを取り出したりすることができる．ユーザは，`scip.h`, `pub_<...>.h` と，それぞれの plug-in のヘッダ・ファイル（例えば，linear constraint handler の場合，`cons_linear.h`）中で宣言されている全ての関数を呼び出すことができる．

ユーザが SCIP の変数 (variable) や制約 (constraint) などの，ある種のオブジェクトについての情報（どういう操作ができるのかなど）を探す場合，最初に，そのオブジェクトに対応する `pub_<...>.h` を探すと良い．例えば，制約 (constraint) に対する操作を探す場合，`pub_cons.h` を調べると良い．ユーザが，問題全体に関する操作を必要とする場合，scip.h の中を探すべきである．その中には， SCIP の 1 つの構成要素のみに対する操作では解決できない “入り組んだ” 操作が全て含まれている．それらの操作に対して，ユーザは常に SCIP ポインタ（後述）を必要とする．

ユーザが SCIP を呼び出し可能ライブラリとして利用する場合，最初に行うことは常に SCIP データ・オブジェクトの生成である．この生成は，`SCIPcreate()` 関数により行い，この関数により SCIP データ・オブジェクトへの SCIP ポインタが得られる．この SCIP ポインタは，多くの他の関数を呼び出す際の引数となる．そして一般的に次に行うことは， `SCIPincludeDefaultPlugins()` によりデフォルトの plug-in を組み込むことである．その後，`SCIPcreateProb()` の呼び出しにより，ユーザはそれぞれの解きたい問題のデータを SCIP データ・オブジェクト中に構築することを始める．

次に，ユーザが行うのは，おそらく `SCIPcreateVar()` による変数の生成と，`SCIPaddVar()` による変数を SCIP データ・オブジェクト内の問題データとして登録することである．制約に関しても，同様に行われるはずである．例えば，一般的な MIP の制約は，`SCIPcreateConsBasicLinear()` （あるいは，より詳細な設定が必要な場合は，`SCIPcreateConsLinear()`) により生成し，`SCIPaddCons()` により SCIP データ・オブジェクトへ登録し，それが終わると `SCIPreleaseCons()` により解放する．全ての変数と制約式を SCIP データ・オブジェクト内に設定した後，`SCIPsolve()` により最適化計算を開始する．最後に，`SCIPgetBestSol()` を用いることで，ユーザは最適解（あるいは，もし途中で計算が停止したなら暫定解）を得ることができる．`SCIPgetSolVal()`, `SCIPgetSolVals()`, `SCIPgetSolOrigObj()` により，それぞれ解における 1 つの変数の値，全ての変数の値，そして目的関数値を取り出せる．変数へのポインタが必要なくなったら，`SCIPreleaseVar()` の呼び出しを忘れてはならない．

これらの関数のほとんどが，`SCIP_RETCODE` の値を返すことに注意されたい．関数呼び出しにおいて全てが上手く動いた場合，`SCIP_OKAY` が戻り値となる．そうでない場合，15 以上のエラーコードの内の 1 つが戻り値となるので，それを参照し，ユーザは処理を継続するかどうかを決める必要がある．`type_retcode.h` の中に，SCIP の内部で利用されているエラーコード一覧があるので参照されたい．

極めた簡単な (toy)MIP の例として， $\min\{1.2x + y \mid x + y \geq 2.5, x \geq 0, y \in \mathbb{Z}_{\geq 0}\}$ を使って説明する．この例を図解すると，図 2 となる．薄く背景色のついた領域が LP 緩和問題の実行可能解の領域である；平行線が混合整数計画問題の実行可能解である．目的関数の減少方向と，最適解が示されている．以下に C 言語によるプログラムにより，SCIP の呼び出し可能ライブラリを利用して，この問題を生成して解くプログラム例を示す．

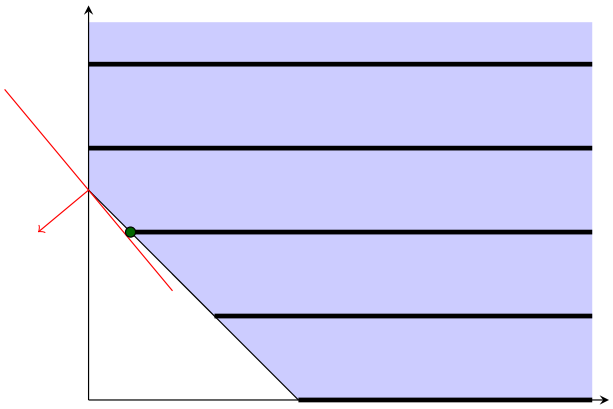

図 2　2 制約 2 変数 (連続変数一つと整数変数一つ) の MIP

```
#include "scip/scip.h"
#include "scip/scipdefplugins.h"
...
   SCIP* scip = NULL;
   SCIP_VAR* x = NULL;
   SCIP_VAR* y = NULL;
   SCIP_CONS* cons = NULL;
   SCIP_SOL* sol = NULL;

   /* initialize SCIP */
   SCIP_CALL( SCIPcreate(&scip) );
   SCIP_CALL( SCIPincludeDefaultPlugins(scip) );
   SCIP_CALL( SCIPcreateProbBasic(scip, "example") );

   /* create and add variables */
   SCIP_CALL( SCIPcreateVarBasic(scip, &x, "x", 0.0, SCIPinfinity(scip), 1.2, SCIP_VARTYPE_CONTINUOUS) );
   SCIP_CALL( SCIPcreateVarBasic(scip, &y, "y", 0.0, SCIPinfinity(scip), 1.0, SCIP_VARTYPE_INTEGER) );
   SCIP_CALL( SCIPaddVar(scip, x) );
   SCIP_CALL( SCIPaddVar(scip, y) );

   /* create empty constraint, add variables to it and release when done with modification */
   SCIP_CALL( SCIPcreateConsBasicLinear(scip, &cons, "constraint", 0, NULL, NULL, 2.5, SCIPinfinity(scip)) );
   SCIP_CALL( SCIPaddCoefLinear(scip, cons, x, 1.0) );
   SCIP_CALL( SCIPaddCoefLinear(scip, cons, y, 1.0) );
   SCIP_CALL( SCIPaddCons(scip, cons) );
   SCIP_CALL( SCIPreleaseCons(scip, &cons) );

   /* solve the problem and output solution values */
   SCIP_CALL( SCIPsolve(scip) );
   sol = SCIPgetBestSol(scip);
   printf("x: %f y: %f\n", SCIPgetSolVal(scip, sol, x), SCIPgetSolVal(scip, sol, y));

   /* release varibles and free SCIP */
   SCIP_CALL( SCIPreleaseVar(scip, &x) );
   SCIP_CALL( SCIPreleaseVar(scip, &y) );
   SCIP_CALL( SCIPfree(&scip) );
...
```

4.2.3　plug-in の実装方法

SCIP の最も凝った利用方法は，plug-in の開発である．ユーザが定義可能なコール・バック・オブジェクトが存在し，そのインターフェースを通じて，フレームワークとのやり取りによる細かな解法の制御が可能となる．SCIP の plug-in 概念は，ユーザ自身の制約整数計画問題のモデルをユーザ自身が組み込んだ plug-in により解くソルバの実装を可能とする．

各 plug-in のタイプに対して，`src/scip` ディレクトリ内にテンプレート`<plugintype>_xyz.{c,h}` が用意されている．ユーザ独自の plug-in を実装するために，ユーザが行うべき事は，その plug-in の優先順位や呼び出し頻度などの属性の指定と，SCIP 中に plug-in として含めるために必要となるインターフェース関数の実装，そして，いくつ

かのコール・バック関数の実装である．例えば，variable pricer を実装するためには，ユーザは， PRICERREDCOST を実装しなければならない．そして，実行可能または実行不可能な LP 解に対して price-and-cut のループ中に呼び出される PRICERFARKAS コール・バックを実装しなければならない．また，各 plug-in で局所的に利用されるデータの初期化と消去だけでなく，各 plug-in のコール・バックに対して必要のある初期化と終了処理を行う必要がある．

4.2.4 デバッグとテストの方法

SCIP のパッケージには，コードのデバッグのサポートと，自動テストおよび結果の評価スクリプトがついている．

SCIP を使ったユーザのコード，あるいは，SCIP そのものをデバッグしたいなら，まず，`make OPT=dbg` によりデバッグモードでコンパイルするべきである．このコンパイルにより，いたるところに存在する assert 命令が有効になると同時に，一環性をチェックするルーチンが有効となる．デバッグのために，ユーザ自身も自らのコードにも同様に assert 命令を追加することを強く勧める．もし，バグが SCIP の特定の構成要素と強く関連していて，その構成要素に関する処理の情報を必要とするなら，`#define SCIP_DEBUG` をファイルの先頭に加えると良い．そうすれば，そのファイル中のデバッグ用の出力が有効になる．そして，assert 命令と同様，ユーザ自身のコード内でも `SCIPdebugMessage()` の利用を勧める．さらに，デバッグのために解を与えることもできる．そのような解を与えることで，解法のどこで与えられた実行可能解が削除されたのかを確認できる．

`make TEST=mytestset SETTINGS=mysettings test` を実行すると，`check/testset/mytestset.test` の中に記述されている全てのインスタンスに対して，`settings/mysettings.set` のパラメタ・セットを利用した自動テストが実行される．自動テスト時には，`TIME`, `NODES`, `MEM` のようなオプションも追加可能である．`TIME` では制限時間が指定でき，`NODES` では制限分枝数が指定できる．また，`MEM` により利用メモリ量も制限できる．自動テストの計算の間，SCIP は自動的に出力を `check/results` ディレクト内の (`*.out`) へ，そして計算結果を `check/results` ディレクト内の (`*.res`) に書き込む．(`*.res`) は，各インスタンスに対して，計算時間，分枝限定法で解いたノード数，単体法の繰り返し回数が出力され，全ての testset 中のインスタンスの結果に対する，それらの数の総数と幾何平均値を，性能を集約した数値として出力する．さらに， `check/allcmpres.sh` を利用すると，`*.res` の内容を多くの異なる評価基準により比較できる．

4.2.5 Help の探し方

■オンライン・ドキュメント． SCIP には，詳細なオンライン・ドキュメントが

http://scip.zib.de/doc/html/index.html

に存在する．そのドキュメントには，FAQ(frequently asked questions), 対話シェルをどのように利用するか，どのように特定のタイプの plug-in を実装するか，どのように公開関数やパラメタを探すかなどが記述されている．さらに進んだ利用者のために，ソースコードがオンライン・ドキュメントでも参照でき，ソースには十分なコメントが記述されている．本稿で示されている制約のタイプに関する正確な定義は，各 constraint handler 内のコメントを参照すると良い．

■パラメタ． 解法を制御するため，SCIP 3.0 のデフォルト plug-in によって利用されるパラメタ数は 1400 以上ある．これらは，対話シェルにおいて `set` メニューから見ることができ，その一覧は機能毎に並べられている：

```
SCIP> set

  <branching>           change parameters for branching rules
  <conflict>            change parameters for conflict handlers
  <constraints>         change parameters for constraint handlers
  <display>             change parameters for display columns
  <emphasis>            predefined parameter settings
  <heuristics>          change parameters for primal heuristics
  <limits>              change parameters for time, memory, objective value, and other limits
  <lp>                  change parameters for linear programming relaxations
  <memory>              change parameters for memory management
```

```
<misc>              change parameters for miscellaneous stuff
<nlp>               change parameters for nonlinear programming relaxations
<nlpi>              change parameters for NLP solver interfaces
<nodeselection>     change parameters for node selectors
<numerics>          change parameters for numerical values
<presolving>        change parameters for presolving
<pricing>           change parameters for pricing variables
<propagating>       change parameters for constraint propagation
<reading>           change parameters for problem file readers
<separating>        change parameters for cut separators
<timing>            change parameters for timing issues
<vbc>               change parameters for VBC tool output
default             reset parameter settings to their default values
diffsave            save non-default parameter settings to a file
load                load parameter settings from a file
save                save parameter settings to a file
```

多くのパラメタは，advanced となっており，明示的に参照したいユーザだけが<advanced>メニューから参照するようになっている．全ての利用可能なパラメタはオンライン・ドキュメントにあり，かつ，ユーザは，対話シェルの set save コマンドによりパラメタをファイルへ出力することもできる．

■公開インタフェース関数． SCIP の関数と SCIP の plug-in における主たる情報源は，オンライン・ドキュメント上の対応するヘッダファイルで， *Files* タブから辿ることができる．

SCIP 本体が持つ公開関数は，scip.h と pub_<...>.h の中の関数である．scip.h の関数は，SCIP の異なる構成要素を使って実現される複雑な操作である．他のヘッダファイルは， pub_var.h 中の変数のタイプを返す SCIPvarGetType() のように，１つのオブジェクトに対する操作を実現する関数である．

加えて， SCIP の関数は，明確な命名規則に従っている．よって，名前の最初の部分を想像できれば，オンライン・ドキュメントの探索ボックスから，しばしば容易に探索できる．

pub_misc.h は，優先順付きキュー，ハッシュテーブル，ハッシュマップのようなデータ構造に対する関数と，ソート，数を扱う関数群，乱数，文字列操作，ファイル操作などの関数を含む．

■plug-in の実装． オンライン・ドキュメントにおける *How to add ...* 節には，どのようにユーザ自身の SCIP plug-in を実装するかに関して，一歩一歩段階的に実装できるような記述が用意されている．ユーザが実装したい plug-in のコール・バック関数の記述を探しているなら，対応する type_<...>.h を探すと良い．例えば，constraint handler のコール・バックに対する記述は，type_cons.h の中にある．

■サンプル・プロジェクト． 一般的なドキュメントの他に，SCIP 3.0 には，特定の問題向けの１つあるいは複数の plug-in をどのように実装するかを示すいくつかのサンプル[*7]が含まれている．

constraint handler の実装に対しては，C 言語で実装した線形順序付け問題 LOP に対する例，あるいは，主問題に対するヒューリスティック解法を C++ 言語で実装している TSP の例を参照されたい．分枝価格法に対しては，Binpacking，あるいは，Coloring の例を参照されたい．それらの例では，variable pricer，分枝変数選択，問題のデータや１つの変数へのデータの追加方法が示されている．

■メーリング・リスト． さらに助けが必要な場合には，ユーザに SCIP のメーリングリスト scip@zib.de への参加を勧めている．現在，メーリング・リストには 200 以上の開発者とユーザが登録されており，特定の質問に対して迅速に答えるよう努めている[*8]．

■プレゼンテーション・論文他． SCIP のページの http://scip.zib.de/further.shtml には，SCIP と CIP に関して，入門解説，プレゼンテーション，練習問題，および，論文が集められている．

[*7] http://scip.zib.de/examples.shtml
[*8] http://listserv.zib.de/mailman/listinfo/scip

4.3 SoPlex

SoPlex は，lp フォーマットあるいは mps フォーマットの線形計画問題を直接解くコマンドラインインタフェースを持っている．Netlib のテストセット[*9]に含まれる capri インスタンスを解くには，単に soplex capri.mps とタイプすれば，以下の結果を得る：

```
Loading LP file capri.mps
LP has 271 rows 353 columns 1767 nonzeros
LP reading time: 0.00

Solving LP ...
IEQUSC01 Equilibrium scaling LP
IMAISM69 Main simplifier removed 26 rows, 43 columns, 246 nonzeros, 82 col bounds, 0 row bounds
IMAISM74 Reduced LP has 245 rows 310 columns 1521 nonzeros
ISOLVE01 iteration = 0          lastUpdate = 0          value = 0.000000e+00
ISTEEP01 initializing steepest edge multipliers
ISOLVE01 iteration = 180        lastUpdate = 180        value = 3.861101e+03
ISOLVE01 iteration = 311        lastUpdate = 131        value = 4.637225e+03
ISTEEP01 initializing steepest edge multipliers
ISOLVE01 iteration = 313        lastUpdate = 2          value = 4.860032e+03
ISOLVE02 Finished solving (status=OPTIMAL, iters=313, leave=311, enter=2, flips=0, objValue=4.860032e+03)

SoPlex statistics:
Factorizations     :          4
  Time spent       :       0.00
Solves             :        881
  Time spent       :       0.00
Solution time      :       0.02
Iterations         :        313

Solution value is: 2.6900129e+03
```

この例では，SoPlex は，元 LP 問題のサイズを表示し，equilibrium scaling を制約行列に適用し，前処理により 43 変数と 26 制約を削除し，313 回の単体法の繰り返し (311 回が双対単体法 (leave)，2 回が主単体法 (enter)) により解かれている．最後の統計情報は，4 回のクリーンな LU 分解が行われ，881 の線形方程式が解かれたことを示している．元問題に対する最適目的関数値 2,690 と，前処理後の問題に対する最適値 4,860 を混同してはならない．単体法が実行されている過程においては，前処理後の問題での目的関数値が表示されている．

■コマンドライン・オプション． soplex を入力ファイル名無しで実行すると，解法を制御するためのオプションのリストが表示される．-f オプションと-o オプションでは，それぞれ，主・双対の実行可能解の許容精度 (feasibility tolerance) が指定できる．デフォルトでは，いずれも 1e-6 である．単体法の繰り返し回数の上限は，-L オプションにより指定できる．-l により，計算の制限時間を秒単位で指定できる．例として，主実行可能解に対する許容精度を 10^{-8} にし，単体法の繰り返し回数を 1000 回に制限して実行する場合，soplex -f1e-8 -L1000 capri.mps とタイプして実行すれば良い．

出力メッセージをどの程度詳細にするかは，-v オプションにより制御できる．単に最適目的関数値だけを表示するのではなく，主・双対解を表示するなら，ユーザは，-x と-y を追加して実行すれば良い．-q を付加することで，解の質をチェックできる．

■基底のファイル入力およびファイル出力． 単に数値解を得るだけではなく，基底の情報を得るためには，-bw オプションを付加し LP のデータファイルの後に，基底出力ファイル名をタイプすれば良い．例えば， soplex -bw capri.mps capri.bas に実行すれば良い．基底を読み込んで，与えられた基底から実行するためには，-br オプションを指定するれば良い．この基底の指定は，SoPlex の前処理の一部 (simplifier) を無効にすることに注意されたい．

[*9] http://www.netlib.org/netlib/lp

■列表現と行表現． デフォルトでは，SoPlex は標準的な列ベースで計算を行う．LP のインスタンスで，変数よりも制約式が多い場合には，行表現に切り替えると良いかもしれない．この切替は，線形代数演算ルーチンの速度をしばしば劇的に加速する．この切替は，`-r` オプションを指定することで行う．

■主・双対単体法． 2.3 節で説明したように，SoPlex は，主・双対単体法の切替を自動的に行う単体法の実装となっている．しかしながら，ユーザは，どちらのアルゴリズムから開始するかを指定できる．

デフォルトでは，列表現と行表現のどちらが使われているかによって変わる．列表現の場合，SoPlex は双対単体法で開始する．行表現の場合，主単体法で開始するが，これは`-e` オプションを指定することで双対単体法に変えることができる．例として，行表現を使うが，双対単体法で開始したい場合， `soplex -r -e capri.mps` のようにタイプして実行すれば良い．

■Pricing 戦略とスケーリング他． 単体法のアルゴリズムの性能は，pricing アルゴリズムに強く依存する．`-p[0-6]` オプションを付けることで，異なった戦略を試すことができる．デフォルト設定では，`-p4`，steepest edge pricing となっている．

その他のオプションで，性能や数値的な安定性に影響するものは，スケーリング戦略（`-g` で指定可能），初期基底（`-c` で指定），ratio テスト (`-t` で指定）である．

■Class インタフェース． コマンドラインでの利用の他に，C++ 言語の Class インタフェースを利用して，SoPlex は他のプログラム中で利用可能である．オブジェクト指向ソフトウェア設計により，pricing 戦略や，スケーリングのアルゴリズムの再実装が容易である．詳細については，http://soplex.zib.de にあるオンライン・ドキュメントを参照されたい．

5 ライセンスと可用性

SCIP Optimization Suite は，*ZIB* **アカデミック・ライセンス**と利用形態に基づく商用ライセンスのもとで利用可能である．ZIB アカデミック・ライセンスでは，研究活動においては研究者はソフトウェアを自由に利用できる．加えて，Zimpl は LGPL3 ライセンスで利用可能である．商用の場合，リクエストにより注文に応じて特別なライセンスが可能である．より詳細なライセンスに関しては，SCIP [2] の WEB ページを参照されたい．

全てのライセンスにおいて， SCIP, SoPlex,Zimpl のソースコードへのアクセスが含まれている．ソースへアクセスできるということは，研究者が完全にその解法を制御できるということである．研究を成功させるためには，アルゴリズム中で何が行われているのかが正確にわかることと，プログラムのどの部分でも変更できることが必要であると我々は考えている．商業ベースで利用する場合には，ソースコードが公開されていない商業ベースのソルバと比較すると，ユーザはより自由にアプリケーションを開発できる．

Linux, Mac, Windows の 3 つの広く利用されている OS 環境においては，それ用にコンパイルされた実行ファイルを WEB ページからダウンロードできる．それらのバイナリは移植性を高めるために，可能な限り共有ライブラリを利用しない静的にリンクされた実行形式となっている．さらに，ライブラリ，特に Windows の DLL もダウロードできるようになっている．Linux と Mac に対しては，ソースから簡単にコンパイルできるビルドシステムも用意されている．

6 実際に利用されている SCIP Optimization Suite

SCIP Optimization Suite は，現在，純粋な研究利用および企業との共同研究利用の両方において，世界の 100 以上の大学・研究機関で利用されている．同様に，ベンチャー企業から大企業まで多くの企業においても利用されてい

る．ABB, Google, SAP, シーメンスなどのグローバル企業にも SCIP はライセンスされており，それらの企業のプロジェクトにおいて定常的に使われている．

多くのプロジェクトは，厳密整数計画（exact integer programming），一般列生成法（generic column-generation）あるいは整数計画における対称性（symmetries in integer programming）など，MIP と MINLP 研究の最前線の研究が基礎を成している．

SCIP が実際に使われ成功した応用分野は，多層遠隔通信ネットワーク，チップ設計検証，公共交通機関サービス，ガスネットワークにおける最適化など多岐にわたる．

7 今後の開発計画

現在の SCIP Optimization Suite のベースとなる開発の歴史は，約 20 年前に Zuse Institute において始まった．それ以来，継続的に線形計画と整数計画に対するコードとツールが開発されている．世界中の多くの研究者と企業が SCIP Optimiziation Suite を利用しているので，我々は開発が今後も継続すると確信している．以下に，SCIP Optimization Suite の将来開発計画のいくつかを紹介する．

■並列化． チップ開発企業が，必然的にクロックの向上からコアを増やす方向への転換をしたので，並列に効率良く動作するアルゴリズムの実装は，特に重要な課題の一つである．整数計画問題の 1 インスタンスを，将来のエクサスケールのシステムを利用して解く際の可能性は [28] に研究されている．現在でも既に，SCIP は，混合整数計画問題のソルバとして最もスケーラブルなソルバの一つとなっている [38].

■厳密整数計画（Exact Integer Programming）． ほとんどの商用アプリケーションにおいて，解はわずかな数値誤差を許容する浮動小数点演算による MIP ソルバにより計算されており，それで十分である．厳密に実行可能解であるかどうかや，厳密に最適性が証明[*10]されているかどうかは，良い実行可能解をできる限り速く見つけることと比較すると，実用的な面での重要性は低くみられている．しかしながら，特別なクラスの問題に対しては，特に実行可能性に関する問題では，数値誤差に対して安全であることは重要である．さらに，研究目的では，MIP の解が厳密に最適であることを証明することが望まれる．理論的には厳密であり，許容できる計算時間で解を得るソルバの開発は重要な仕事である．[13] に書かれているように，多くの解決されなければならない点もあるが，この方向性に関する顕著な進歩もある．

■MINLP と CP． この数十年間の汎用 MIP の進歩は極めて大きいものである．これらの進歩と非線形最適化と制約プログラミングの研究コミュニティにおける技術を統合し，これによってさらなる挑戦的なクラスの最適化問題，特に MINLP に適用することが我々の目標である．SCIP は，この方向性において最先端のソルバであり続けたい．

■Citius, altius, fortius（より速く，より高く，より強く）． より多くの，より大規模なインスタンスをより速く解くことは，数理計画において常に目的であり，SCIP Optimization Suite の開発者たちの永遠の目標である．

謝辞

本研究は，DFG Research Center MATHEON *Mathematics for key technologies* in Berlin (http://www.matheon.de) のサポートを受けている．日本語原稿を読んでコメントを下さった兵庫県立大学の藤江哲也教授に感謝する．

[*10] ここで，“証明” は数値誤差を除いた証明である，よって，最適性ギャップがある小さな値イプシロンではなくゼロになったということを意味する．

参考文献

[1] IBM ILOG CPLEX Optimizer. http://www.ilog.com/products/cplex.

[2] SCIP. Solving Constraint Integer Programs. http://scip.zib.de/.

[3] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[4] T. Achterberg, T. Koch, and A. Tuchscherer. On the effects of minor changes in model formulations. Technical Report 08-29, Zuse Institute Berlin, Takustr. 7, Berlin, 2009.

[5] E. Althaus, A. Bockmayr, M. Elf, M. Jünger, T. Kasper, and K. Mehlhorn. SCIL – symbolic constraints in integer linear programming. In *Algorithms – ESA 2002*, pages 75–87, 2002.

[6] I. D. Aron, J. N. Hooker, and T. H. Yunes. SIMPL: A system for integrating optimization techniques. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2004*, volume 3011 of *Lecture Notes in Computer Science*, pages 21–36, 2004.

[7] T. Berthold, S. Heinz, and S. Vigerske. Extending a CIP framework to solve MIQCPs. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 427–444. Springer, 2011.

[8] J. Bisschop and A. Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, 20:1–29, 1982.

[9] R. Bixby, Z. Gu, and E. Rothberg. Gurobi optimization, 2010.

[10] A. Bockmayr and T. Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, 1998.

[11] A. Bockmayr and N. Pisaruk. Solving assembly line balancing problems by combining IP and CP. Sixth Annual Workshop of the ERCIM Working Group on Constraints, June 2001.

[12] M. R. Bussieck and A. Meeraus. General algebraic modeling system (GAMS). In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages 137–158. Kluwer, 2004.

[13] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. An exact rational mixed-integer programming solver. In O. Günlük and G. J. Woeginger, editors, *IPCO 2011: Proceedings of the 15th International Conference on Integer Programming and Combinatoral Optimization*, volume 6655 of *Lecture Notes in Computer Science*, pages 104–116, 2011.

[14] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[15] FICO. Xpress. http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx.

[16] J. Forrest and R. Lougee-Heimer. COIN branch and cut, user guide. http://www.coin-or.org/Cbc.

[17] J. J. Forrest and J. A. Tomlin. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[18] R. Fourer, D. M. Gay, and B. W. Kernighan. A modelling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[19] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. Brooks/Cole—Thomson Learning, 2nd edition, 2003.

[20] A. Gleixner, D. Steffy, and K. Wolter. Improving the accuracy of linear programming solvers with iterative refinement. Technical report, Zuse Institute Berlin, 2012.

[21] GNU. the GNU linear programming kit. http://www.gnu.org/software/glpk.

[22] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[23] E. Hebrard, E. O'Mahony, and B. O'Sullivan. Constraint programming and combinatorial optimisation in numberjack. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *LNCS*, pages 181–185. Springer, 2010.

[24] J. N. Hooker and M. A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96-97(1):395–442, 1999.

[25] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4):258–276, 2001.

[26] M. Jünger, T. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors. *50 Years of Integer Programming 1958-2008*. Springer, 2009.

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

[28] T. Koch, T. Ralphs, and Y. Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, 76:67–93, 2012.

[29] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[30] A. Lodi. MIP computation. In M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer, 2009.

[31] V. Manquinho and O. Roussel. Pseudo-boolean competition 2012. http://www.cril.univ-artois.fr/PB12/.

[32] L. Michel and P. V. Hentenryck. The comet programming language and system. In P. van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference*, volume 3709 of *LNCS*, pages 881–881, 2005.

[33] H. Mittelmann. Decision tree for optimization software: Benchmarks for optimization software. http://plato.asu.edu/bench.html.

[34] R. J. O'Neil. PYTHON-ZIBOPT. http://code.google.com/p/python-zibopt/.

[35] P. Refalo. Tight cooperation and its application in piecewise linear optimization. In *Principles and Practice of Constraint Programming, CP 1999*, volume 1713 of *Lecture Notes in Computer Science*, pages 375–389, 1999.

[36] R. Rodosek, M. G. Wallace, and M. T. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86(1):63–87, 1999.

[37] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

[38] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. ParaSCIP – a parallel extension of SCIP. In C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, editors, *Competence in High Performance Computing 2010*, pages 135–148. Springer, February 2012.

[39] L. M. Suhl and U. H. Suhl. A fast LU update for linear programming. *Annals of Operations Research*, 43:33–47, 1993.

[40] C. Timpe. Solving planning and scheduling problems with combined integer and constraint programming.

OR Spectrum, 24(4):431–448, November 2002.

[41] S. Vigerske. *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming.* PhD thesis, Humboldt-Universität zu Berlin, 2012. Submitted.

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