混合整数計画ソルバーの並列化

品野　勇治，藤江　哲也

混合整数計画（Mixed Integer Programming：MIP）ソルバーCPLEXを並列化するParaLEX（Parallel Extension for CPLEX Mixed Integer Optimizer）を紹介する．まず，MIPソルバーについて概観し，MIPソルバーにおける並列処理の役割を示す．その後，ParaLEXの設計目標，動作概要を紹介する．数値実験結果として，代表的なベンチマーク問題集MIPLIB 2003の中でも，最も困難な問題例を解いた結果について紹介する．

キーワード：整数計画法，混合整数計画ソルバー，CPLEX，並列処理

1. はじめに

混合整数計画問題（Mixed Integer Programming problem：MIP problem）は，線形計画問題（Linear Programming problem：LP problem）の一部またはすべての変数に整数条件が付加されたものである．MIPは様々な分野に応用例をもつことが知られている．また，巡回セールスマン問題など数多くの組合せ最適化問題を記述することが可能な，汎用性の高い数理計画問題として知られている[9][11][12][15]．それと同時に，実際に解くことが非常に困難な問題であるとも認識されていた．しかし近年，MIPに対するソフトウエア（MIPソルバー）の性能が飛躍的に向上し，注目されている[10][13]．例えば，文献[13]ではILOG社のCPLEX Mixed Integer Optimizer[8]に対する計算結果が報告されており，CPLEX 7.0（2000年10月リリース）では歯がたたない問題の多くがCPLEX 10.0（2006年1月リリース）では実用的な時間で解けている．

本稿では，文献[14]で提案された，CPLEX Mixed Integer Optimizerそのものを並列化するシステム（ParaLEX：Parallel Extension for CPLEX Mixed Integer Optimizer）を紹介する．その前にまず，この研究の背景と動機について述べることにする．

1.1 MIPソルバーについて

現在のMIPソルバーの主流は，LPベースの分枝カット法である．CPLEXなど著しい成果を挙げている商用のソルバーは，ロバストなLPソルバーに加え，分枝カット法の要素技法，すなわち，前処理（preprocessing），ヒューリスティクス（heuristics），分枝変数の選択（variable selection），子問題の選択（node selection），カット（cut）等において最新の研究・開発が注ぎこまれている[13]．一方，COIN-OR[6]はオープンソースとして，分枝カット法の各要素技法の評価のためのフレームワークを提供している．COIN-ORは年々充実してきており，例えばCut Generation Libraryには，Lift-and-Projectなど最新の成果も盛りこまれている[4]．しかし，商用ソルバーにおいても，カスタマイズ可能なフレームワークが提供され，新たな理論の検証にも利用できる上，ソースコードはオープンでないものの頑健性の面では確実に優位にあるといえる状況である．

1.2 並列処理の効果について

MIPなどのNP困難な問題に対しては，並列処理の効果は解法そのものの改善効果に比べると極めて小さい．例えば，二次割当問題の未解決問題が，当時は現在ほど有名でなかったグリッド環境で解かれ[2]，組合せ最適化問題を解く環境としてグリッドが注目された．しかし，この成功の要因としては，ヒューリスティック解法の研究が進み，最適解かそれにほぼ等しいものがあらかじめ得られていたこと，そして新たな二次計画問題に基づく下界値計算法の開発によるところが，グリッド環境を利用したこと以上に大きい．つまり，最新の解法を用いていることが重要な点である．実際，MIPに対しては，良い実行可能解を得ること自体が困難であり，最新の解法を実装した商用ソルバーの優位性が決定的な現状では，並列処理の適用による顕著な成功例はない．例えば，文献[2]の研究と同

しなの　ゆうじ
東京農工大学 大学院共生科学技術研究院
〒184-8588 小金井市中町2-24-16
ふじえ　てつや
兵庫県立大学 経営学部
〒651-2197 神戸市西区学園西町8-2-1

じ環境で実現されたFATCOP[5]や，COIN-ORの分枝カット法のフレームワークであるBCP (Branch-Cut-Price) に実現されている並列処理も，ある程度の有用性は示されているが，MIPソルバーとしての顕著な成功事例とはなっていない．

すなわち，MIP（そして組合せ最適化問題）の厳密解法に対する並列処理の役割は，最新の解法を加速するところにある．並列化の対象が最新の解法でなければ，並列処理するプログラムを開発したとしても，その処理時間は最新の解法を逐次実行した場合よりも遅くなることの方が多いと思われる．つまり，並列処理によって，本当に価値ある結果を望むのであれば，最新の解法を実装したソルバーを並列化するべきである．その一方で，過去の技術と新しい技術の間に10倍以上の速度差があれば，技術は新しい方へ移転するといわれている[10]．並列処理を適用すると，この速度向上は一般的に得やすい．したがって，最新の解法が常に並列化できれば，並列処理の適用は極めて有意義である．

ところで，MIPにおいて，最新の解法を常に並列化するようなプログラムの開発には，次の2つのアプローチが考えられる．

- COIN-ORで行われているように，並列処理を含む分枝カット法のフレームワークを提供し，そのフレームワークに含まれる要素技法の部分を常に最新に保つ．
- 最新の技法を実装済みの商用ソルバーが提供するAPI (Application Program Interface) を用いて，商用ソルバーそのものを並列化する．

文献[14]で提案されたParaLEXは，後者のアプローチであり，ここまで述べてきた研究背景がこのアプローチを採用する動機となっている．ParaLEXの特徴は，各PE (Processing Element：物理的に独立に計算を実行する装置．本稿では1PEに対して，プログラムの実行単位である1プロセスが対応していると仮定する）に与えられた子問題を，MIP問題としてCPLEXに解かせるものである．そして，必要に応じて未処理の子問題（未分枝の子問題）の送受信を行うよう制御する．よって概念的には，ParaLEXの生成する分枝木全体はCPLEXの生成する分枝木をつなぎ合わせたものとなっている．

2. MIPソルバーのフレームワーク

MIP全般に関しては[9][11][12][15]，分枝カット法については[7][15]を参照されたい．

MIP問題は，次のように定式化される：

$$\text{(MIP)}\quad \text{最小化}\quad c^{\top}x$$
$$\text{条　件}\quad Ax \leq b,$$
$$x_i \in \mathbb{Z}\quad (i=1,2,\cdots,p).$$

ただし，決定変数 $x=(x,x_2,\cdots,x_n)^{\top}$ は n 次元ベクトルである．また，c は n 次元ベクトル，b は m 次元ベクトル，A は $m\times n$ 行列であり，p は $p\leq n$ を満たす整数である．$x_1,\cdots,x_p$ は整数変数，$x_{p+1},\cdots,x_n$ は連続変数とよばれる．また，整数変数 x_i の取り得る値が0または1の場合，x_i を0-1変数とよび整数変数と区別されることが多い．

図1に分枝カット法のフレームワークを示す．商用ソルバーの多くは，分枝カット法における各技法を，ユーザが独自にカスタマイズすることを可能にする機構が用意されている．CPLEXの場合には，図1の（ ）内に示すようなコールバックルーチンを利用する(図1に示したのは一部である)．コールバックルーチンは，MIPソルバーの実行（CPLEXのC言語によるインタフェースでは，CPXmipopt関数呼出）の前に，各種コールバックに対応するユーザルーチンを登録することにより利用可能となる．コールバックルーチンが登録されると，分枝カット法における各タイミングで，MIPソルバーからユーザが登録したルーチンへ制御が渡る．制御を受け取り，ユーザは独自の技法を実装することができる．ユーザルーチンの登録を行わなければ，ソルバーが標準で提供している実装が実行される．また，ユーザルーチンが登録されても，ユーザルーチン中で，CPLEXの実装を利用するか，あるいは独自の実装の結果を利用するかを戻り値に示すこともできる．

3. ParaLEX

ParaLEXは，CPLEX（CPLEX Mixed Integer Optimizer）をPCクラスタ上で並列実行させることに特化したプログラムである．データ通信にはMPI (Message Passing Interface) を利用している（PCクラスタ以外では実行していないが，MPIとCPLEXが動作する並列計算機上でも実行可能なはずである)．CPLEXにはparallel optimizerが存在するが，これはスレッドレベルの並列化であり，並列処理される処理単位（粒度）が異なる．そのため，ParaLEXとCPLEXのparallel optimizerの併用により，プロセスを処理単位とする並列化と，スレッド

```
分枝カット法:
begin
  (MIP) に対する前処理を実行:
  z* := ∞:
  (MIP) を子問題プール L に入れる:
  カットプールを空にする:
  while L ≠ ∅ do begin
    子問題プール L から子問題 S を選択する:
    (node selection コールバック)
    L から S を除去する:
    カットプールに含まれる不等式を S に追加する:
    切除平面法を実行する:
    if z* より良い実行可能解が見つかった then
      x* と z* を更新する:
      (incumbent コールバック)
    if 新しい子問題が生成された then
      それらを子問題プール L に入れる
      (branch selection コールバック)
  end:
  return x*
end.
```

図1　分枝カット法のフレームワーク

を処理単位とする並列化が同時に実現でき，CPLEX の parallel optimizer をさらに加速することも可能なはずである（ただし，parallel optimizer のライセンスを多数所有していないと実現できない）．

3.1　設計目標

ParaLEX の設計目標と，それを達成するための手段は次の通りである．

1. **CPLEX の提供している MIP Optimizer の機能は全面的に利用**：CPLEX が行う前処理（変数の固定など）はそのまま利用するとともに，CPLEX が生成するグローバルカットや，ローカルカット，実装されているヒューリスティックスを並列処理に際しても利用する実装とする．
2. **将来のバージョンの CPLEX もコードを変更することなく並列化**：CPLEX のソルバー・フレームワークが提供する API のみを利用する．さらに，もっともプリミティブで仕様が変更されることの少ない CPLEX callable library の API を利用する．
3. **最も単純な並列化の実現**：基本的には，単純な Master-Worker パラダイムによる並列化を実現する．

3.2　実装

ParaLEX は，C++言語によって記述された，MPI を用いた SPMD (Single Program Multiple Data stream) 型のプログラムである．まず，ParaLEX による並列化に際して鍵となる，**ParaLEX インスタンス**と，PE 間で送受信される**転送ノード**について述べる．CPLEX におけるノードとは，分枝木におけるノードを意味し，分枝カット法における子問題を指す．

3.2.1　ParaLEX インスタンス

ParaLEX で保持されるインスタンスデータは，元問題に前処理を適用し，さらにルートノードで生成されるグローバルカットを制約式として加えたものである．CPLEX におけるこれらの処理は極めて強力であり，数十台による並列処理の効果を軽く上回る．したがって，これらの処理は必ず行うべきである．一方，これらの処理には，かなりの時間を要する．ルートノードに対する処理時間は，それ以外のノードの 10 倍以上になることも一般的である．そのため，各 PE でこれらの処理を実行することは好ましくなく，ParaLEX ではこれらの処理が施された後のインスタンスを保持することとした．

3.2.2　転送ノード

ParaLEX では，CPLEX が保持する未分枝ノードを取り出して転送する．この処理は branch selection コールバックルーチンで行われる．branch selection コールバックルーチンへは，計算（図 1 における切除平面法）が終わったノードが渡される．ノードの転送が必要となった際には，このノードを転送ノードとし，送り側のコールバックルーチンでは新たな子ノードの生成を抑止する．転送ノードは，送り側で計算が終わっているため，そのノードに施されたローカルカットおよび LP 緩和問題の最適基底が取り出せる．ローカルカットが施された問題と ParaLEX インスタンスの差分，および，最適基底をノードの情報として転送する．ノードの受け取り側では，送られてきたノードをルートノードとして処理する．つまり，転送ノードは 2 度解かれることになる．しかし，受け取り側では，初期基底が与えられることで LP 緩和問題を高速に解くことができる．また，ルートノードとして解くため，新たなカットの追加により定式化は強化され，またヒューリスティック解法の動作により，実行可能解の更新が促される．全体の実行時間を考慮すると，このような処理が再度実行されることには，ノードを 2 回解くデメリットに匹敵する程度の価値はある．

ParaLEX は SPMD 型の単一プログラムであるが，

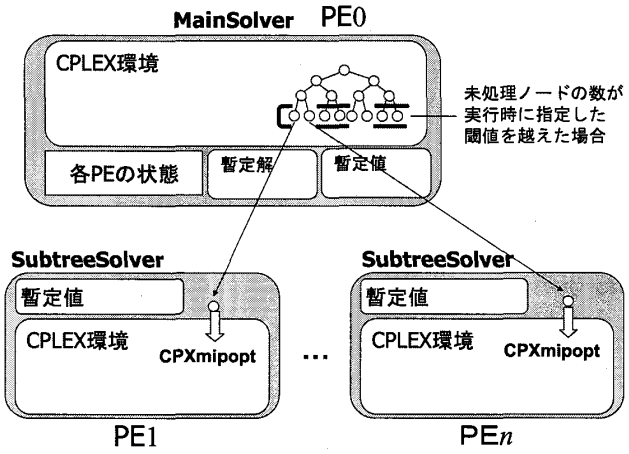

図2　ParaLEX の初期化段階

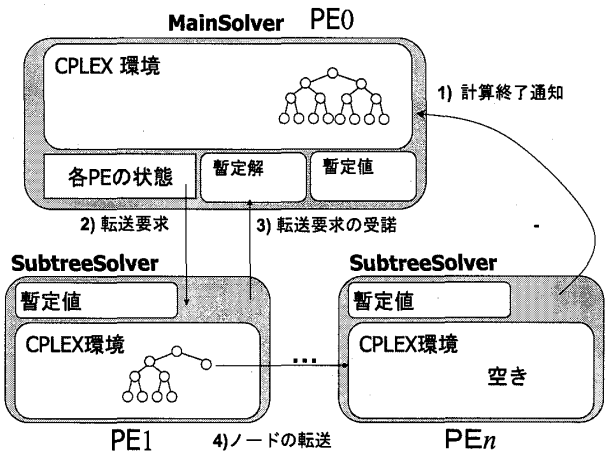

図3　ノード転送時のメッセージ交換

起動時の MPI rank により，次に示す 2 種類のソルバーのいずれか一方が動作する．MPI rank は各 PE に対して割り当てられる ID と考えられるので，本稿では PE rank と呼ぶ．

MainSolver（PE rank＝0）　このソルバーは，問題データを読み込み ParaLEX インスタンスを作成する．通常の Master のように，PE の管理と負荷分散を行うだけでなく，CPLEX によって子問題（ノード）を MIP として解く処理も行う．システム全体の暫定解，暫定値を保持し，計算終了時には結果を出力する．

SubtreeSolver（PE　rank≥1）　受け取ったノードを CPLEX によって MIP として解く．MainSolver からの要求により，未分枝ノードを他の SubtreeSolver，または，MainSolver へ転送する．暫定解が更新されたら，それを MainSolver へ返す．

ParaLEX の動作概要について，簡単に紹介する．まず，MainSolver が CPLEX により ParaLEX インスタンスを解き始める．MainSolver 内の CPLEX 環境における未分枝ノード数（CPLEX が保持する未分枝ノード数）が，実行時パラメタで指定した数（詳細は文献[14]）を超えると，ノードは 1 つずつ SubtreeSolver へ転送される（図 2 参照）．SubtreeSolver がノードを受け取ると，そのノードをルートノードとして CPLEX により解き始める．暫定解が更新されると，それを MainSolver へ返す．また，実行時パラメタで指定した一定回数の分枝ごとに，CPLEX が保持する未分枝ノード数，および，最良下界値を MainSolver へ伝える．MainSolver では，これらを PE の情報として管理する．

SubtreeSolver において受け取ったノードの処理が終了した場合，その SubtreeSolver は空きとなり，計算を終了したことが MainSolver へ伝えられる．その PE へノードを転送するため，MainSolver は PE の情報を見て，最も下界値の良いノードを保持している PE へ転送要求を出す．ただし，MainSolver が最良下界値をもつ場合には，即座にノードを転送する．PE の情報の更新には遅れがあるので，転送要求を受け取った SubtreeSolver には，既に十分なノード数がない（この閾値は実行時パラメタで与えられる）場合もある．その場合，SubtreeSolver は要求を拒否する．SubtreeSolver に十分なノード数があれば，ノード選択を行い，ノードは転送要求により示された SubtreeSolver へ転送される．この際，ノード選択として CPLEX が標準で用いているノード選択と，最良下界値による選択が指定可能である．現状では，処理をできる限り単純化するため，1 つの要求メッセージに対して，1 ノードだけが転送される（図 3 参照）．

暫定解の更新にともない，CPLEX が保持する未分枝ノードに対する限定操作が可能となる．これを実現するため，CPXmipopt 関数の実行前に，分枝を途中で中断するための分枝数を指定している．分枝が中断した際，CPLEX のパラメタ設定関数を利用して，ある値以上の下界値をもつノードの計算を抑止する指定を行い，CPXmipopt 関数を再度呼び出すことで限定操作を実現している．

MainSolver, SubtreeSolver で分枝カット法を実行する際，CPLEX が用意しているパラメタのほとんどは，パラメタファイルに記述することで指定可能である．

3.3 数値実験結果

ここで，30台のCPLEX Mixed Integer Optimizer (version 10.1) を用いた数値実験結果の一部を紹介する．計算機環境は，3.4 GHz pentium D 950 (2 Gbytes RAM) のPC 15台をギガビットイーサネット接続したクラスタであり，MPI libraryはmpich 2-1.0.5p4を用いた．表1は，ベンチマーク問題集MIPLIB 2003[1][1]のうち，最近解かれた問題をParaLEXで解いた結果を示している．ただし，ノード転送の際のノード選択（図3参照）は，最良下界値によるものである．また，表中の“MIP Emphasis”はCPLEXのパラメータの一つであり，改善解の発見を重視する，最適性の保証を重視する，等を指定することができる．複数ある選択肢の中で，BALANCED (最適性と解の発見のバランスを保つ：CPLEXのデフォルト設定)，OPTIMALITY（解の発見よりも最適性の保証を重視する)，BESTBOUND（最良下界値選択によって，より最適性の保証を重視する）を利用した．予備実験の結果では，問題やパラメータ設定によって並列化の効果はさまざまであったが，現段階において表1に挙げる問題を解くことができた．これらの問題は現在も難しいとされており，例えばroll 3000は，BESTBOUNDによる単体実行では230464.65秒まで動かしても解くことができず，2213435の未分枝ノードが残った．したがって，この問題に対して超線形加速が得られたことがわかる．

図4は，atlanta-ipに対して，時間の関数としてすべてのPEの上界値と下界値をプロットしたものである．横軸の1目盛りは1日に相当しており，29日強で終了している（表1参照)．また，図5はPEごとに上下界値をプロットしたものを表示している．

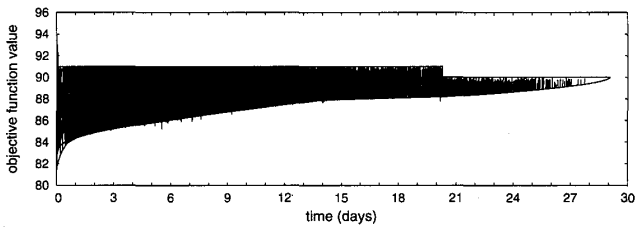

図4 atlanta-ipに対する上界値と下界値の推移（すべてのPE)

[1] URL : http://miplib.zib.de

表1 ParaLEXの計算結果

問題名	制約式	変数	整数変数	0-1変数	連続変数	計算時間 (秒)	MIP Emphasis
a1c1s1	3312	3648	0	192	3456	142960.36	BESTBOUND
atlanta-ip	21732	48738	106	46667	1965	2512451.70	BALANCED
arki001	1048	1388	123	415	850	3924.87	OPTIMALITY
glass4	396	322	0	302	20	2001.50	BALANCED
roll3000	2295	1166	492	246	428	173.73	BESTBOUND

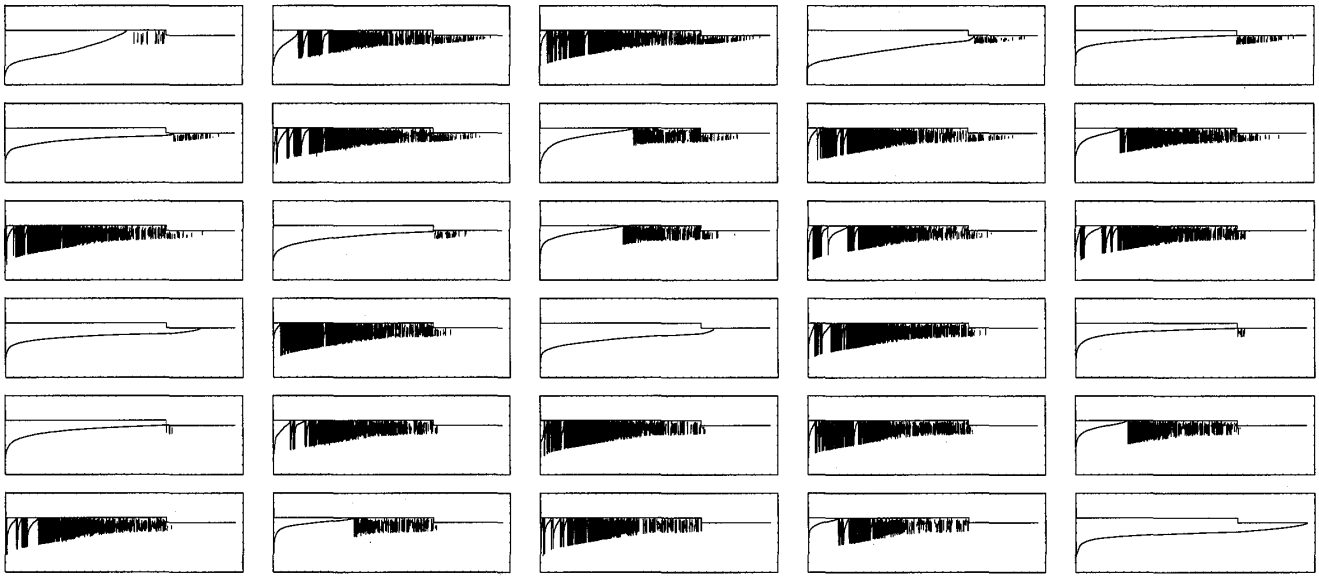

図5 atlanta-ipに対する上界値と下界値の推移（各PE)

4. おわりに

ParaLEXの開発は，まだ初期段階にある．シンプルな実装を目標とし，単純なMaster-Workerの実装も行ったが，CPLEX環境からのノードの取り出し方が，CPLEXが提供するAPIに頼るしかないため，うまく動作しなかった．そこで，SubtreSolver間でノード転送を行うよう変更した．並列処理では，空きのMainSolver, SubtreeSolverも生じるいまだ不十分な状態ではあるが，解くことが困難な問題に対しては，並列化の高い効果（超線形加速）が比較的頻繁に生じることを確認した．その結果，MIPLIBのベンチマーク問題における最も困難な問題のいくつかは解ける状態となった．

真に大規模な問題を解く際に，並列分枝限定法によって，超線形加速が期待できる例は比較的少ない．混合整数計画問題では，良い実行可能解の発見そのものがいまだに困難な状況にあるため，並列分枝限定法本来の特徴が生きる．一方，ParaLEXのようなアプローチにより，今後，より効果的な並列処理を実現するためには，CPLEXそのものの動作との協調が不可欠である．CPLEXの動作について詳細な調査を行い，次期バージョンでは，よりCPLEXと協調した，さらに効果的な並列処理を実現したい．

参考文献

[1] T. Achterberg, T. Koch and A. Martin: "MIPLIB 2003," *Operations Research Letters*, **34**, pp. 361-372, 2006.

[2] K. M. Anstreicher, N. W. Brixius, J.-P. Goux and J. Linderoth: "Solving large quadratic assignment problems on computational grids," *Mathematical Programming*, **91**, pp. 563-588, 2002.

[3] L. Atamtürk and W. P. Savelsbergh: "Commercial Integer Programming Software Systems," *Annals of Operations Research*, **140**, pp. 67-124, 2005.

[4] E. Balas and P. Bonami: "New Variants of Lift-and-Project Cut Generation from the LP Tableau: Open Source Implementation and Testing," in *Integer Programming and Combinatorial Optimization-IPCO 2007*, M. Fischetti and D. P. Williamson, eds., LNCS **4513**, pp. 89-103, 2007.

[5] Q. Chen, M. Ferris and J. T. Linderoth: "FATCOP 2.0: Advanced Features in an Opportunistic Mixed Integer Programming Solver," *Annals of Operations Research*, **103**, pp. 17-32, 2001.

[6] COIN-OR (Computational Infrastructure for Operations Research): http://www.coin-or.org/

[7] A. Fügenschuh and A. Martin: "Computational Integer Programming and Cutting Planes," in *Discrete Optimization*, K. Aardal, G. L. Nemhauser, R. Weismantel, eds., Elsevier, 2005.

[8] ILOG, Inc.: CPLEX. http://www.ilog.co.jp/

[9] 今野浩："整数計画法，" 講座・数理計画法 6, 産業図書, 1981.

[10] 今野浩："役に立つ一次式—整数計画法「きまぐれな王女」の50年，" 日本評論社, 2005.

[11] 今野浩，鈴木久敏(編)："整数計画法と組合せ最適化，" ORライブラリー7, 日科技連出版社, 1982.

[12] 久保幹雄, 田村明久, 松井知己(編)："応用数理計画ハンドブック，" 朝倉書店, 2002.

[13] 宮代隆平, 松井知己："ここまで解ける整数計画，" システム/制御/情報, **50**, pp. 363-368, 2006.

[14] Y. Shinano and T. Fujie: "ParaLEX: A Parallel Extension for the CPLEX Mixed Integer Optimizer," *Proceedings of the 14th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2007)*, to appear.

[15] L. Wolsey: "Integer Programming," John Wiley & Sons, 1998.