コンテナプリマーシャリング問題における探索空間の削減

小池　英勝[1]

要　旨

本論文は，コンテナプリマーシャリング問題（CPMP）の最適解を効率的に得るための探索空間の削減について論じる．CPMP に関する既存の研究の多くがヒューリスティックアプローチをとるのに対して，本論文では，解の最適性を保証しながら効率化するアプローチを追求する．そして，提案するアプローチが，最適性と効率性両面で，既存のヒューリスティックアプローチを凌駕する可能性を示す．

キーワード：組み合わせ最適化，コンテナプリマーシャリング問題，最適化，NP 困難，ロジスティクス

1　はじめに

本研究の目的は，コンテナプリマーシャリング問題（Container Pre-Marshalling Problem: CPMP）の最適解を計算機で高速に解くための方法を開発することである．CPMP は NP 困難な問題クラスに属すると考えられているため，既存の手法の多くは，ヒューリスティックスや近似アルゴリズムを用いて計算量を抑えながら近似解を求める．一方，最適解を得るための計算量はそれよりも遥かに多くなるため，実用規模で問題の最適解を得る試みは少ない．本研究では，CPMP の最適解をできるだけ高速に求めるための手法とそれに基づくプログラムを開発する．本研究の目的が達成されれば，流通上のコンテナ混雑問題を改善すると同時に，実用的な規模の NP 困難な問題に対する新しいアプローチを提案することができる可能性がある．以降本論文は以下の様に構成される．2 節で，コンテナプリマーシャリング問題と既存の研究について概説する．3 節で，本研究の動機とそれに基づく方針について述べる．4 節で，CPMP を厳密に議論するために，CPMP とその関連項目を定義する．5 節で，CPMP の性質に関して議論した後，最適性を保証しながら計算効率を向上させるためのアプローチについて議論する．6 節で，本論文が提案する方法を実装した実験プログラムの性能評価を行う．7 節でまとめる．

[1] 札幌学院大学 経済学部; koike@sgu.ac.jp.

2　コンテナプリマーシャリング問題

2.1　CPMP の概要

CPMP とは，コンテナ埠頭などで起こる流通混雑問題解決のための，コンテナ並び替え問題の一つである．コンテナが陸上から船に搬入され輸送される前に，陸送されてきたコンテナはコンテナヤードに一時集積される．集積されたコンテナは，船に搬入される前に，その行先，内容，重さなどから，搬入の優先度が決まる．搬入時は，その優先度に従って搬入順序が決定され，もし，高い優先度を持つコンテナが低い優先度をもつコンテナの下にある場合，上にあるコンテナを移動した後に目的のコンテナを取り出さなくてはならない．この余剰な移動は船積みの時間を増加させ，結果として船の停泊時間が長くなる．船の停泊時間は港のパフォーマンスの重要な指標であり，流通の混雑に大きく関わっている．もし，コンテナの優先度が決まってから船が到着するまでの時間を使って，低い優先度のコンテナが高い優先度のコンテナの上に一つもない状態（クリーンな状態）に並べ替えておけば，船積みの時間を最短にできる．CPMP とは，優先度が考慮されずに集積されたコンテナの初期の状態から，クリーンな状態に並べ替えるための最短の手順を発見する問題である．CPMP では，移動に用いるクレーン，スペー

ス，そして，安全面等の制約から，コンテナはBayと呼ばれる幅と高さが決められた2次元空間でのみ移動可能である．並べ替え中は，Bay中のコンテナの総数は変わらない．異なるコンテナが同じ優先度を持つことが許される．

2.2　既存の研究

これまでのCPMPの研究は，問題のNP困難性（Caserta *et al*., 2011）から，近似解を求めるアプローチがほとんどであった．最適解アプローチを扱う文献は少なく，著者が知る限り，実用規模と比較して小規模な問題しか扱えないか，ヒューリスティックアプローチとの比較に用いられるものだった．2015年には，最適解アプローチを扱った文献（Zhang *et al*., 2015）が現れ，文献中で扱える問題の規模が実用規模であると主張されていたが，ヒューリスティックアプローチと比較すると，その規模はまだ小さいといえる．しかし，ヒューリスティックアプローチでは，その性質上最適解を見逃す可能性があり，このような制限は実用的にも問題になりうる．CPMPの一般的な定義は，最小（もしくは最短）のコンテナの移動の手順という最適解を求めることであるが，ヒューリスティックアプローチは，近似解を求めることにより問題設定を変えているとも言える．多くのヒューリスティックアプローチが，その手続きと実験結果・性能比較については議論されるが，高速化の結果何が失われているかに関して，ほとんど考察されていない．

3　本研究の動機と方針

CPMPに関する現状をみると，最適解アプローチでは，解の最適性を保証できるが計算時間がかかり，ヒューリスティックアプローチでは，計算時間を短縮できるが解の最適性が保証できない．しかし，著者は，どちらのアプローチも最終的な目的は同じなので，両アプローチ共に究極まで改善すると，非常に近いものになるのではないかと考えた．そして，可能な限り最適解アプローチを追及しながら探索空間を縮小させる方法を開発することで，両アプローチに有用な成果が出せるのではないかと考えた．

本研究の大きな特徴は，組み合わせ最適化問題でありNP困難に属するCPMPの最適解を効率的に求めることを追求している点である．本研究では，最適解が得られないことが証明された探索空間だけを省略する．よって，得られる解が最適解であることが保証される．そして，その探索空間をいかに高速に探索するかを追求する．このことを実現するために，問題の表現方法の開発，アルゴリズム最適化，並列化，実装の最適化を行う．より具体的には，独自のデータ構造とアルゴリズムの開発，プログラムの正当性の保証，そして，現状のCPUやメモリを含む計算リソースの構造と特性を踏まえてそれを有効活用するための高度なプログラム最適化を行う．既存の研究で開発されている実験プログラムが十分に計算リソースに最適化される状況は少ないと考えられる．しかし，プログラムの最適化のレベルによって同じアルゴリズムでも数百倍の性能差が起こりうる．アルゴリズムを実験によって正しく評価するには，効率的なプログラムを記述するための技術が必要不可欠である．よって，本研究では，データ構造とアルゴリズムの最適化と実装の最適化を両方とも妥協せずに行う．

4　CPMPの定義

本節では，CPMPを本研究で厳密に議論するための定義を行う．

4.1　前提条件

CPMPはコンテナの並べ替え問題であるが，類似の問題，例えば，コンテナリロケーション問題（Container Relocation Problem, CRP）(Kim & Hong, 2006）などと明確に区別するためにコンテナに関する前提条件を以下に明示する．

(1)　コンテナは，同一のBayでのみ移動可能である．並べ替え中にコンテナの総数は変わらない．
(2)　全てのコンテナは，同じサイズである．
(3)　全てのコンテナには，前もって優先順位を示す番号が与えられている．
(4)　異なるコンテナが同じ優先順位を持つことが許される．
(5)　コンテナのスタック間の移動（すなわち，横の移動）のコストは無視できる．

前提条件(5)は，クレーンによるコンテナの上下の移動が，横の移動よりもコストが大きいことから仮定される（Bortfeldt & Forster, 2012）．前提条件(5)によって，Bayをスタックのマルチセットとみなすことが可能になり，本研究ではこの概念によって，計算量を削減することに成功した．マルチセットに関する詳細は

別の論文で述べる．

4.2 要素の定義

上の前提条件を基にして，CPMP の構成要素を定義し，更にそれを用いて，問題の定義を行う．

(Bay の定義 1) コンテナの移動について議論する場合，Bay は，スタックの列である．

(Bay の定義 2) Bay の等価性について議論する場合，Bay は，スタックのマルチセットである．Bay を構成するスタックの数を S で表す．

(スタックの定義) スタックは，スロットの列であり，LIFO でコンテナを出し入れする．同一の Bay を構成するスタックは，すべて同じ長さを持ち，それを H で表す．

(スロットの定義) スロットは，コンテナを高々 1 個格納できる．スロットがコンテナを持たないとき，スロットは空であるという．スタックの長さ H は，スタックが持つスロットの個数でもある．

(コンテナの位置) Bay は，S × H の 2 次元配列ともみなせる．Bay 中のコンテナの位置を (s, h) で表す．ここで，s はコンテナが含まれるスタックを表し，h は，コンテナの縦の位置を表す．s は 0 から始まる整数で，0 は最左を表す．h は 0 から始まる整数で，0 はスタックの底に位置することを表す．$0 \leq s < S$，$0 \leq h < H$ である．スロットの位置も，コンテナと同様の表現で表す．

(コンテナ下のスロットの制約) コンテナが (s, h_t) に位置するとき，その下 (s, h_b)，$0 \leq h_b < h_t$ にある全てのスロットは空ではない．

(コンテナの優先度とグループ) 全てのコンテナには，優先度を表す整数 p が与えられる．優先度が最も低いコンテナに P が与えられるとき，$0 \leq p \leq P$ である．コンテナが優先度 0 を持つとき，もっとも高い優先度を持つ．コンテナに p が与えられているとき，コンテナはグループ p に属するという．異なるコンテナが同じグループに属することがある．

(well-placed) コンテナ c が以下の条件のいずれかを満たすとき c は well-placed であるという．

- c がスタックの底に位置する．
- c が p_1 に属し，かつ，p_2 に属する well-placed なコンテナの直上に位置し，かつ，$p_1 \leq p_2$ である．

スタックが空の場合，または，スタックが格納する全てのコンテナが well-placed のとき，そのスタックは well-placed であるという．

(badly-placed) コンテナが well-placed でないとき badly-placed であるという．スタックが well-placed でないとき，そのスタックは badly-placed であるという．

(fixed) コンテナ c が以下の条件のいずれかを満たすとき c は fixed であるという．

- c が最も低い優先度を持ち，かつ，well-placed である．
- c が well-placed で，かつ，c より優先度の低いコンテナが全て well-placed である．

コンテナが fixed である場合，そのコンテナは決して移動しない．スタックが格納するコンテナが全て fixed であるとき，そのスタックは fixed であるという．Bay 中の全てのコンテナが fixed のとき，その Bay は fixed であるという．

(コンテナの移動) コンテナの移動とは，スタックの最上位にあるコンテナを別のスタックの最上位に移動することである．スタック S_b にあるコンテナがスタック S_a に移動したとき，この移動を (S_b, S_a) と表す．$m_i\ (0 \leq i \leq N)$ を移動とするとき，$m_0, ..., m_N$ を移動の列と呼ぶ．移動の列を Bay に適用するとは，m_0 から順に m_N までその内容に従って Bay 中のコンテナを移動することである．

4.3 CPMP の定義

(CPMP) コンテナプリマーシャリング問題 (CPMP) とは，与えられた Bay を fixed にするための最短のコンテナの移動の列を求めることである．

5 CPMP の性質と計算の効率化

5.1 計算パスの重複

CPMP の解は一般に複数ある．その理由は，与えら

れたBayからコンテナを移動して得られるfixedなBayが複数ありうることと，一つのBayから得られる一つのfixedなBayに対して，それを得るための移動の列が複数ありうることである．このことから，計算中の探索パスには実質的に重複するものが多数現れると推測できる．そのことの傍証として計算中のBayの状態を履歴として記憶し，探索の重複を取り除く処理を行うと，計算効率を改善することができる．

5.2　コンテナの移動の種類

Bortfeldt & Forster（2012）では，コンテナの移動をBG，GG，BB，GBという4つのクラスに分類している．BGに属する移動はbadly-placed(B)なコンテナをwell-placed(G)の状態に移動する．他のクラスも同様にして2つの頭文字でその意味が表されている．BGに属する移動は，Bayをfixedの状態に近づけるので，できるだけBGだけで解を構成するようにすれば，それが最適解になる．これまでの実験結果から，多くの場合，最適解の移動の列にはBGに属する移動が最も多く，その10〜20%程度のBB，GGが含まれ，GBは最も少ない．GBが少ない理由は，GBの移動を行うと，その後で必ず余剰なBGの移動が起こるため，移動の列を長くするからである．ただし，BG以外の移動を省略すると，最適解が得られない例題は存在するため，たとえGBであっても，安易に選択肢から排除することはできない．実際にGBの移動が，最適解を構成する例がある．

5.3　解の長さの下界

もし，計算対象のBayでn個の移動の列の解が見つかった場合，その解の最適性の証明のためには，n-1個以下の解があるかを探索すればよい．もし，n-1以下の解が存在しないことが明らかになった場合，n個の移動の列が最適解である．しかし，解が存在しないことを計算するには1個からn-1個までの移動の列を全て探索する必要があるので，ある程度のサイズの問題になると大きな計算コストが生じる．探索以外の方法で，解の長さの下界（lower bound）を求める方法が提案されている．ここでの下界は，整数値で表され，解は少なくとも下界以上の長さを持つ，という意味で使われる．もし，下界がnの時，n個の移動の列が解として得られた場合，その解は最適解である．

最も簡単な下界の計算方法は，Bay中のbadly-placedなコンテナの総数を下界とするものである．この方法は，BGに属する移動の最小数を求めることになり計算コストが低いという利点があるが，一般的にはBG以外の移動が必要なので，精度は低い．BG以外の移動の数も考慮に入れた下界の計算方法がBortfeldt & Forster（2012）で示されている．この方法は，GGとGBの数の計算のためのコストが高いが，より高い精度で下界を求めることができる．著者は，この下界の計算方法を拡張して，より精度を高めることに成功した．この方法については別の論文で述べる．

5.4　その他の最適性を保証した効率化

最適解を求めようとした場合，Bayに対して可能な全てのコンテナの移動を試す必要がある．Bayに対して，次の一つの移動を考えると一般には複数の候補がある．ヒューリスティックアプローチでは，この複数の候補の中から一つ，もしくは，いくつかを選択することで計算効率を改善するが，このときに解の最適性を保証できなくなる．最適解アプローチでは，この時に候補から外せるのは，それが確実に最適解を構成しないと証明された時だけである．そのような例として，単純なものでは，直前に移動したコンテナを対象とする移動は明らかに無駄なのでそれを候補から外しても最適性は損なわれない．Zhang *et al.*（2015）では，候補の移動と，これまでの移動の列を結合し，その結合がそれより短い列に変換可能であれば，その移動は候補から外すという方法が述べられている．

5.5　並列処理

今後，上記のような最適性を保証した効率化の技術が改善されれば，移動の候補はより減少するが，現状はBayに対して次の移動の候補は一般に複数存在する．これらの移動の候補の適用は互いに独立して行えるので並列処理が行える．本研究によって，特定の問題に対してはマルチスレッドによる並列処理がCPUの論理コア数以上の性能改善を示すことが分かっている．例えば，Expósito-Izquierdo *et al.*（2012）で述べられているBay生成器でランダムに作成した問題に対して，8論理コアのCPUでシングルスレッドの場合と比較して，8.4倍の性能改善の例がある．これは，解が複数存在する問題ではマルチスレッドでの探索の方がシングルスレッドよりも早く解を発見する確率が高いためと考えられる．これらの詳細については別の

論文で議論する．CPMP において，コア数の増加に対して本研究の提案するアプローチがどのように性能改善するかはまだ良くわかっていない．コア数の増加の実験では GPGPU も検討したが，現状では CPU による並列化の方が高い効果を得られるようである．その理由は，CPMP のアルゴリズムは再帰的に記述され (Bortfeldt & Forster, 2012; Zhang *et al*., 2015)，スレッドで実行されるコードは比較的複雑なものになることと，履歴を用いた効率化では，スレッドが履歴を共有するためボトルネックになることと，そして，履歴は計算中に現れた全ての Bay の状態を記録するので現状の GPGPU のメモリでは容量が少ないということがある．

6　実験プログラムの性能評価

本研究では CPMP に対して，ヒューリスティックアプローチではなく，解の最適性を保証した効率化のみで高速な計算の実現を目指す．本研究で開発，または，採用した効率化を整理すると，履歴，下界，マルチスレッド，最適性を保証した候補の除外の 4 つに分類できる．最適性を保証した候補の除外は，実際には様々なテクニックが組み合わされるので，更にクラス分け出来るが，詳細については他の論文で議論する．

これまでの CPMP に関する論文で公開された例題の中でも，著者が知る限り，最大のものが Bortfeldt & Forster (2012) で示されている．この例題には現在の実用的なサイズとみなせるものが含まれる．表 1 は，この例題を本研究の方法に従い動作する実験プログラムで計算した結果である．表の第 1 列は，テスト

表 1　実験結果

Test case	No. of stacks	Stack height	No. of containers	No. of container groups	No. of badly placed containers	Mean no. of moves	Mean time(s)	No. opt. answers
BF1	16	5	48	10	29	29.1	<1.0	20
BF2	16	5	48	10	36	36	<1.0	20
BF3	16	5	48	20	29	29.1	<1.0	20
BF4	16	5	48	20	36	36	<1.0	20
BF5	16	5	64	13	39	40.9	<1.0	20
BF6	16	5	64	13	48	48.85	<1.0	20
BF7	16	5	64	26	39	41.5	<1.0	20
BF8	16	5	64	26	48	49.3	2.1	20
BF9	16	8	77	16	47	50.05	1.9	17
BF10	16	8	77	16	58	58.7	<1.0	14
BF11	16	8	77	31	47	50.5	<1.0	14
BF12	16	8	77	31	58	58.35	<1.0	19
BF13	16	8	103	21	62	75.35	33.2	0
BF14	16	8	103	21	78	92.2	29.4	0
BF15	16	8	103	42	62	77.15	29	0
BF16	16	8	103	42	78	93.5	31.6	0
BF17	20	5	60	12	36	36.25	<1.0	20
BF18	20	5	60	12	45	45	<1.0	20
BF19	20	5	60	24	36	36.45	<1.0	20
BF20	20	5	60	24	45	45	<1.0	20
BF21	20	5	80	16	48	50.65	<1.0	20
BF22	20	5	80	16	60	60.7	<1.0	19
BF23	20	5	80	32	48	50.4	<1.0	20
BF24	20	5	80	32	60	60.65	<1.0	20
BF25	20	8	96	20	58	61.25	1.8	11
BF26	20	8	96	20	72	72.25	<1.0	18
BF27	20	8	96	39	58	61.25	4.1	14
BF28	20	8	96	39	72	72.45	<1.0	18
BF29	20	8	128	26	77	94.25	22.1	0
BF30	20	8	128	26	96	112.2	35	0
BF31	20	8	128	52	77	94.65	23	0
BF32	20	8	128	52	96	111.65	34	0

ケースの識別子である．第2列は，Bayのスタック数である．第3列は，スタックの高さである．第4列は，Bay中のコンテナ数である．第5列は，グループ数である．第6列は，初期のBayが持つbadly-placedなコンテナの個数である．第6列は，解の平均の長さである．第7列は，解が得られるまでの実行時間の平均である．第8列は，最適解が得られた問題の数である．各テストケースは20個の例題を持つ．実験に用いたPCはIntel Xeon E5-2687Wv2を2個搭載し32論理並列実行可能で，128GBのメモリを搭載する．OSはWindows10 Pro 64bitである．実験プログラムはC＋＋で記述されている．

実験時，処理時間は一つの問題に対して60秒を上限として解を探索させた．処理効率についての比較は，利用している計算機が異なるので省略する．表の第7列が示す通り，Bortfeldt & Forster（2012）の実験結果よりも解の短さで本研究の結果が上回っている．本研究のアプローチは，時間を十分与えればいつか最適解が得られることを強調したい．ただし，最適解が得られなかった問題の多くは，現状では数日かけても最適解が得られない．

7　ま　と　め

本研究は，CPMPに対して，ヒューリスティックアプローチを用いずに最適解を求めようとするものである．効率化の手法は，全て解の最適性を保存するものだけを採用する．上述の実験結果が示すように，このアプローチで解の最適性，処理効率両面で良好な結果が得られた．

本研究の履歴による効率化は有効であるが，サイズの大きな問題に対して履歴がPCのメモリの容量を超えることがある．これに対応する処理は，コストが高く，今後この改善が必要である．現状で最適解が得られない問題に対する今後の取り組みとしては，最適性を保証した効率化の技術の開発が重要であると考える．今後これらの開発を進め，より効率的な解法を提案したい．

謝辞　本研究は，2014年度札幌学院大学研究奨励金（C）SGU-CS14-198023-01の助成を受けた．

参考文献

[1] Bortfeldt, A. and Forster, F. (2012). A Tree Search Procedure for The Container Pre-Marshalling Problem. European Journal of Operational Research, 217(3), 531-540.
[2] Caserta, M., Schwarze, S. and Voß, S. (2011). Container Rehandling at Maritime Container Terminals. In: Böse, J.W. (ed.) Handbook of Terminal Planning, Operations Research/Computer Science Interfaces Series, 49, Springer New York, 247-269.
[3] Expósito-Izquierdo, C., Melián-Batista B. and Moreno-Vega, J.M. (2012). Pre-Marshalling Problem: Heuristic Solution Method and Instances Generator, Expert Systems with Applications, 39, 8337-8349.
[4] Kim, K.H. and Hong, G.P. (2006). A heuristic rule for relocating blocks. Computers & Operations Research, 33, 940-954.
[5] Zhang, R., Jiang, Z. and Yun, W.Y. (2015). Stack pre-marshalling problem: A heuristic-guided branch-and-bound algorithm, International Journal of Industrial Engineering, 22(5), 509-523.

The Proceedings of the Research Institute of Sapporo Gakuin University Vol.4, 9–15 (2017)

Reducing Search Space of the Container Pre-Marshalling Problem

Hidekatsu KOIKE[1]

Abstract

This paper discusses an efficient search method for optimal solutions of the container pre-marshalling problem by reducing the search space. This paper pursues an optimization method preserving its optimality without heuristics, while most of the existing approaches introduce heuristics for efficiency. Further, this paper demonstrates how the proposed approach surpasses heuristic approaches in both optimality and efficiency.

Keywords: Combinational Optimization, Container Pre-Marshalling Problem, Optimization, NP-Hardness, Logistics.

[1]Department of Economics, Sapporo Gkuin University; koike@sgu.ac.jp.