特別研究報告書

コンテナ積載計画問題における重量バランスを考慮した解のZDDによる列挙

指導教員：川原 純 准教授

京都大学工学部情報学科

古渡 健太

2024年1月30日

コンテナ積載計画問題における
重量バランスを考慮した解の
ZDD による列挙

古渡 健太

内容梗概

コンテナ積載計画問題 (CSPP; Container Stowage Planning Problem) とは，各種の制約条件を満たしながら，コンテナ船へのコンテナの最適な積載配置を決定する問題である．古くは 1984 年の J. J. Shields による研究で，コンテナ船の能力を最大限活用するためにコンテナの積載計画を最適化させる重要性が述べられている．最適化の目的は，荷役コストを最小化しながらコンテナ船の利用効率を最大化することであるが，この目的に従ってコンテナの配置を決定することは，非常に幅広く複雑な制約条件と目的を併せ持つ最適化問題である．

最適化の目的としては，荷役機器の動きにかかる時間の最小化，シフト回数 (積み下ろしたい対象のコンテナ以外のコンテナを移動させる作業) の最小化，使うことができなくなるスペースの最小化，重量の分配の最適化，など様々な目的が存在し，さらにこれらの多目的最適化の研究も存在する．

また，CSPP を解く研究では，様々なアプローチが試されている．すなわち，混合整数計画モデル，タブーサーチ法，大近傍探索法，遺伝的アルゴリズム，機械学習などを用いた研究が存在する．中でも，決定木ベースのアプローチの 1 つとして，R. M. Jensen らによる，決定木の一種である BDD (Binary Decision Diagram) を用いた CSPP の解列挙の研究がある．BDD とは, DAG (Directed Acyclic Graph; 有向非巡回グラフ) の一種で，論理関数を表現することができる．CSPP の解が満たすべき制約条件を論理関数で表現することで，制約条件を満たすコンテナ配置の解の集合を表現する BDD の構築を行なっている．

本研究では，BDD の派生系である ZDD (Zero-suppressed BDD) を用いた CSPP の解列挙とそれを用いた最適化の手法を提案する．ZDD とは，DAG の一種で，BDD と同様に論理関数を表現することができる．同じ論理関数を BDD と ZDD で表現したとき，ZDD は BDD よりも少ない節点数で表現できる場合がある．ZDD を CSPP に適用した研究は著者が知る限りこれまでに存在せず，CSPP においても ZDD が BDD よりも少ない節点数で同じ解を表現できる可能性があるため，本研究ではその検証を行う．Jensen 同様，解が満たすべき制約

条件を論理関数で表現し，その論理関数を表現する ZDD を構築することによって，解集合を表現する ZDD を構築する．コンテナ重量を考慮しない問題設定において，同じ解集合を表現する BDD・ZDD について，実験を行ったインスタンス全てにおいて ZDD の節点数が BDD の節点数よりも削減されることを確認し，その削減の割合の違いの原因について考察を行った．最大で BDD よりも 63.3%少ない節点数で同じ解集合を表現できたほか，最大でコンテナ数が 36 個であるインスタンスについて，解集合を表現する BDD・ZDD を 2 分未満で構築することができた．

また，ZDD を用いることで線形重み最適化が可能であるが，この手法を用いることで，コンテナ重量を考慮した問題設定を行い，垂直方向と水平方向の重量バランスを考慮した解の列挙を行う．垂直方向の重量バランスについては，コンテナの重量と，コンテナが置かれる垂直方向の座標の積の総和によって定まる垂直モーメントを最小化することを目的とした線形重み最小化を行うことで，最適な解の取得が可能である．一方，水平方向の重量バランスについては，コンテナの重量と，コンテナが置かれる水平方向の座標の積の総和によって定まる水平モーメントの絶対値を最小化することを目的とするが，これは線形重み最小化によって求めることはできない．そこで，解集合を表現する ZDD に対して，水平モーメントが 0 以上である解の集合を表現する ZDD を，区間メモ化探索技法 と呼ばれる手法を用いた，ある重み以上の集合全てを含む ZDD を返す関数を用いて構築し，その ZDD に対して線形重み最小化を行うことで，最適な解の取得を行う．

以上の手法によって，垂直方向の重量バランスや水平方向の重量バランス，またはその両方を考慮した時の，最適解および上位解を取得できることを確認し，実験インスタンス毎の各種演算の計算時間や構築される ZDD の節点数の違いの原因を考察した．解集合の要素数すなわち制約条件を満たすコンテナ配置の組合せ数が最大で 20 億を超えるインスタンスを含む全てのインスタンスについて解集合を表現する ZDD を 20 秒未満で構築できることを確認したほか，垂直方向のバランス・水平方向のバランス・あるいはその両方が最適な値を取る解を取得することができた．

Enumeration of Weight-Balanced Solutions to the Container Stowage Planning Problem using ZDD

KOWATARI Kenta

Abstract

The Container Stowage Planning Problem (CSPP) involves finding the best way to load containers on a ship, adhering to various constraints. This challenge, crucial for maximizing ship capacity utilization, was highlighted in J. J. Shields' 1984 study. The goal is to enhance ship efficiency and reduce loading costs, but this requires navigating a complex array of constraints and goals.

Objectives include minimizing cargo handling time, reducing container shifts, optimizing space usage, and balancing weight distribution, sometimes tackled through multi-objective optimization.

Solving CSPP has seen diverse methods like mixed integer programming, tabu search, large neighborhood search, genetic algorithms, and machine learning. Notably, R. M. Jensen et al. used a decision tree-based approach, employing Binary Decision Diagrams (BDD) – a type of Directed Acyclic Graph – to map out solutions that meet these constraints. Jensen's BDD represents solutions by converting constraints into logical functions.

This study proposes a solution enumeration and optimization method for CSPP using ZDD (Zero-suppressed BDD), which is a derivative of BDD. When the same logic function is expressed by BDD and ZDD, ZDD may be able to express it with less number of nodes than BDD. Similar to Jensen, we construct a ZDD that represents a solution set by expressing the constraints that the solution must satisfy in terms of logical functions and constructing a ZDD that represents the logical functions. In the problem setting where container weight is not considered, we confirmed that the number of nodes in the ZDD is reduced compared to the number of nodes in the BDD for all instances of the BDD and ZDD that represent the same solution set in the experiment, and discussed the cause of the difference in the reduction ratio. We were able to represent the same solution set with a maximum of 63.3% fewer nodes than BDD, and we were also able to construct BDD and ZDD

that represent the solution set for instances with a maximum of 36 containers in less than 2 minutes.

In addition, ZDD enable linear weight optimization. Using this method, we set up the problem considering the container weight and enumerate the solutions considering the vertical and horizontal weight balance. For the vertical weight balance, the optimal solution can be obtained by linear weight minimization, which aims to minimize the vertical moment determined by the sum of the product of the container weight and the vertical coordinate at which the container is placed. On the other hand, for the horizontal weight balance, the objective is to minimize the absolute value of the horizontal moment determined by the sum of the product of the container weight and the horizontal coordinate at which the container is placed, which cannot be obtained by linear weight minimization. To find optimal solutions, ZDD representing solution sets with horizontal moments of 0 or more are built using Interval-Memoized Backtracking. This method generates ZDDs with sets exceeding a specific weight, upon which linear weight minimization is applied.

We confirmed that the above method can obtain optimal and superior solutions when vertical weight balance, horizontal weight balance, or both are considered, and discussed the causes of differences in the computation time of various operations for each experimental instance and the number of nodes in the constructed ZDD. We confirmed that ZDD representing the solution set can be constructed in less than 20 seconds for all instances, including instances where the number of elements in the solution set, i.e., the number of combinations of container arrangements that satisfy the constraint conditions, exceeds 2 billion at the maximum, and we were also able to obtain solutions with optimal values for the vertical balance, horizontal balance, or both.

コンテナ積載計画問題における
重量バランスを考慮した解の
ZDD による列挙

目次

1　はじめに

コンテナ積載計画問題 (CSPP; Container Stowage Planning Problem) とは，各種の制約条件を満たしながら，コンテナ船へのコンテナの最適な積載配置を決定する問題である．古くは 1984 年の J. J. Shields による研究で，コンテナ船の能力を最大限活用するためにコンテナの積載計画を最適化させる重要性が述べられている [1]．最適化の目的は，荷役コストを最小化しながらコンテナ船の利用効率を最大化することであるが，この目的に従ってコンテナの配置を決定することは，非常に幅広く複雑な制約条件と目的を併せ持つ最適化問題である．

この複雑性を具体的に述べると，まずコンテナには次のような性質の幅広さがある．すなわち長さ (主に 20ft, 40ft, 45ft)，高さ (主に 8.6ft, 9.6ft)，種類 (DRY; 通常のコンテナ, REEFER; 冷蔵・冷凍機能を備えたコンテナ, OOG; 上部や側部に天井や壁が存在しないコンテナ, TANK; タンク形状のコンテナ)，内容物の重量，内容物の種類 (IMDG Codes; The International Maritime Dangerous Goods Codes に定められたコンテナ内容物の区別) [2] などの性質が存在し，その性質に伴う船との間の制約条件や，コンテナ同士の制約条件が存在する．また，コンテナが船から下される予定の港と，コンテナ船がある港を出港してからが寄港する港の順番との兼ね合いによって生まれる，コンテナの積載順番の制約なども存在する．これら以外にも，コンテナ船の構造の違い，港によって異なる荷役機器の種類など，現実の CSPP には考慮しなければならない制約条件が非常に多く存在する．

また，このような制約条件の複雑さの他に，目的の複雑さも存在する．具体的には，荷役機器の動きにかかる時間の最小化，シフト回数 (積み下ろしたい対象のコンテナ以外のコンテナを移動させる作業) の最小化 [3]，使うことができなくなるスペースの最小化 [4]，重量の分配の最適化 [4]，などが目的とされるほか，これらの多目的最適化の研究 [4] も存在する．

CSPP を解く研究では，様々なアプローチが試されている．すなわち，混合整数計画モデル [5]，タブーサーチ法 [6]，大近傍探索法 [7]，遺伝的アルゴリズム [8]，機械学習 [9] などを用いた研究が存在する．中でも，決定木ベースのアプローチの 1 つとして，R. M. Jensen らによる，決定木の一種である BDD (Binary Decision Diagram) [10] を用いた CSPP の解列挙の研究 [11] がある．BDD とは, DAG (Directed Acyclic Graph; 有向非巡回グラフ) の一種で，論理関数を表

現することができる．Jensen [11] では，制約条件を論理関数で表現することで，制約条件を満たすコンテナ配置の解の集合を表現する BDD の構築を行なっている．

本研究では，BDD の派生系である ZDD (Zero-suppressed BDD) [12] を用いた CSPP の解列挙とそれを用いた最適化の手法を提案する．ZDD とは，DAG の一種で，BDD と同様に論理関数を表現することができる．同じ論理関数を BDD と ZDD で表現したとき，ZDD は BDD よりも少ない節点数で表現できる場合がある．ZDD を CSPP に適用した研究は著者が知る限りこれまでに存在せず，CSPP においても ZDD が BDD よりも少ない節点数で同じ解を表現できる可能性があるため，本研究ではその検証を行う．Jensen [11] 同様，解が満たすべき制約条件を論理関数で表現し，その論理関数を表現する ZDD を構築することによって，解集合を表現する ZDD を構築する．コンテナ重量を考慮しない問題設定において，同じ解集合を表現する BDD・ZDD について，実験を行ったインスタンス全てにおいて ZDD の節点数が BDD の節点数よりも削減されることを確認し，その削減の割合の違いの原因について考察を行った．最大で BDD よりも 63.3%少ない節点数で同じ解集合を表現できたほか，最大でコンテナ数が 36 個であるインスタンスについて，解集合を表現する BDD・ZDD を 2 分未満で構築することができた．

また，ZDD を用いることで線形重み最適化が可能である [13] が，この手法を用いることで，コンテナ重量を考慮した問題設定を行い，垂直方向と水平方向の重量バランスを考慮した解の列挙を行う．垂直方向の重量バランスについては，コンテナの重量と，コンテナが置かれる垂直方向の座標の積の総和によって定まる垂直モーメントを最小化することを目的とした線形重み最小化を行うことで，最適な解の取得が可能である．一方，水平方向の重量バランスについては，コンテナの重量と，コンテナが置かれる水平方向の座標の積の総和によって定まる水平モーメントの絶対値を最小化することを目的とするが，これは線形重み最小化によって求めることはできない．そこで，解集合を表現する ZDD に対して，水平モーメントが 0 以上である解の集合を表現する ZDD を，区間メモ化探索技法 [13] と呼ばれる手法を用いた，ある重み以上の集合全てを含む ZDD を返す関数を用いて構築し，その ZDD に対して線形重み最小化を行うことで，最適な解の取得を行う．

以上の手法によって，垂直方向の重量バランスや水平方向の重量バランス，ま

たはその両方を考慮した時の，最適解および上位解を取得できることを確認し，実験インスタンス毎の各種演算の計算時間や構築される ZDD の節点数の違いの原因を考察した．解集合の要素数すなわち制約条件を満たすコンテナ配置の組合せ数が最大で 20 億を超えるインスタンスを含む全てのインスタンスについて解集合を表現する ZDD を 20 秒未満で構築できることを確認したほか，垂直方向のバランス・水平方向のバランス・あるいはその両方が最適な値を取る解を取得することができた．

本論文の構成は以下のとおりである．2 章では本研究で使用する数学的概念についての定義と，ZDD の操作で使用する関数について述べる．3 章では既存手法として Jensen [11] を元に CSPP の定式化を行い，CSPP の解集合を表現する BDD の構築方法を述べる．4 章では提案手法として解集合を表現する ZDD の構築方法と，コンテナ重量を導入した重量バランス最適化の手法を述べる．5 章では 2 つの実験結果と考察を述べる．1 つめの実験はコンテナ重量を考慮しない問題設定において，同じ解集合を表現する BDD・ZDD を構築し比較を行う実験であり，2 つめの実験はコンテナ重量を考慮した問題設定を行い，垂直方向と水平方向の重量バランスを考慮した解の ZDD による列挙を行う実験である．

2 準備

2.1 BDD

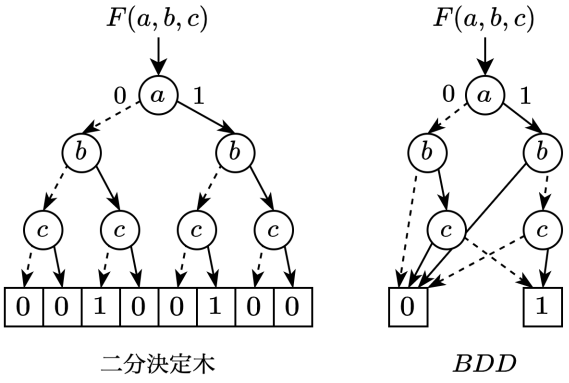

図 1: 二分決定木，BDD

BDD は DAG の一種であり，n 個の論理変数 $x_i (x_i \in \{0,1\}, i \in \{1,2,\ldots,n\})$ を引数とする論理関数 $F : B^n \to B, B = \{0,1\}$ を DAG で表現する手法である．BDD の節点には終端節点と非終端節点と呼ばれる 2 種類の節点がある．1 つの非終端節点は 1 つの論理変数 x_i に対応する．非終端節点は自身を始点とする 0-枝と 1-枝と呼ばれる 2 つの有向辺をもち，それぞれ始点と異なる別の終端節点もしくは非終端節点を終点に持つ．0-枝は始点となる非終端節点に対応する論理変数に 0 を割り当てる，すなわち $x_i = 0$ を表現し，1-枝は始点となる非終端節点に対応する論理変数に 1 を割り当てること，すなわち $x_i = 1$ を表現する．終端節点には 0-終端節点と 1-終端節点と呼ばれる 2 つの節点がそれぞれ 1 つずつ存在する．0-終端節点は F の返り値 0 に対応し，1-終端節点は F の返り値 1 に対応する．BDD には自身を終点とする有効辺が存在しない非終端節点を 1 つ持ち，これを始点として 0-枝もしくは 1-枝を辿ることで，最終的に 0-終端節点もしくは 1-終端節点に至る．最終的に 1-終端節点に至ったとき，訪れた非終端節点と 0-枝もしくは 1-枝に対応する，論理変数に対する値の割り当てが，F の返り値が 1 となる割り当てであることを意味する．0-終端節点に至った場合は F の返り値が 0 となる割り当てであることを意味する．なお，訪れた非終端節点に含まれない論理変数については，0 または 1 どちらの値が割り当てられても F の返り値は変わらない．例として $F(a,b,c) = a\bar{b}c \vee \bar{a}b\bar{c}$ を二分決定木・BDD で表現したものを図 1 に示す．丸い枠は非終端節点を表現し，枠内の文字は対応する論理変数を表す．四角い枠は終端接点を表現し，枠内の数字は F の返り値を表す．破線の矢印が 0-枝，実線の矢印が 1-枝を表す．BDD は，論理関数の値を全ての変数について場合分けした結果を表す二分決定木に対して，簡約化処理を可能な限り加えることによって構築することができる．これによって，論理関数をコンパクトかつ一意に表せることが知られている．

この二分決定木に対して，場合分けする変数の順序を固定し，(1) 冗長な節点を削除する (図 2)，(2) 等価な節点を共有する，という簡約化規則を可能な限り加えることで，BDD を得ることができる．なお，冗長な節点とは，自身を視点とする 0-枝と 1-枝の両方が同じ節点を指している非終端節点である．冗長な節点を BDD から取り除き，この節点を終点としていた 0-枝または 1-枝またはその両方の終点を 0-枝と 1-枝が指していた節点とする，また，等価な節点とは，相異なる非終端節点であって，それぞれの 0-枝の終点が互いに同じ節点であり，かつそれぞれの 1-枝の終点も互いに同じ節点であるような非終端節点である．

等価な節点の一方を BDD から取り除き，取り除いた BDD を終端節点としていた 0-枝または 1-枝またはその両方の終点を，もう一方の節点とする．

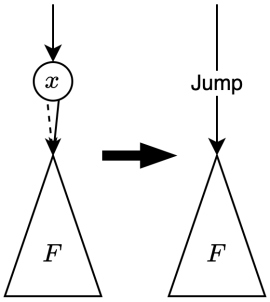

図 2: BDD における冗長な節点の削除規則

また，BDD を用いて組合せ集合を表現することもできる．組合せ集合は，特性関数と呼ばれる論理関数に対応付けることができるが，この特性関数を BDD で表すことにより，その組合せ集合を非明示的に表現できる．

2.2 ZDD

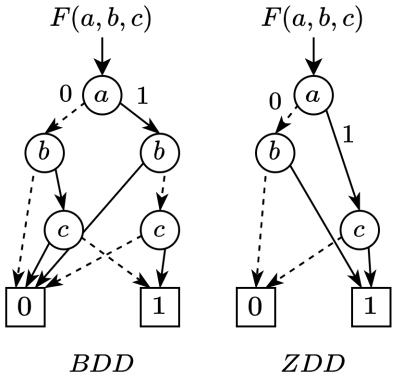

図 3: BDD，ZDD

ZDD は，組合せ集合データを表現・操作するのに特化した BDD の派生系である．$F(a,b,c) = a\bar{b}c \vee \bar{a}b\bar{c}$ を BDD・ZDD で表現した例を図 3 に示す．等価な節点を共有する規則は BDD と同様であるが，図 4 に示す通り冗長な節点を削除する規則が異なる．ZDD の場合は，1-枝が 0-終端節点を指している場合に，こ

れを取り除き，この節点を終点としていた 0-枝または 1-枝またはその両方の終点を 0-枝と 1-枝が指していた節点とする，という規則を用いる．この規則を用いた場合でも，表現の一意性は失われない．ZDD の場合，訪れた非終端節点に含まれない論理変数については，0 が割り当てられるものとする．

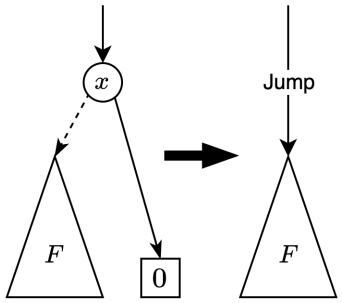

図 4: ZDD における冗長な節点の削除規則

一般に，組合せ集合に類似する部分組合せが多数含まれる場合，それを表現する BDD には等価なサブグラフが多く出現する．これらが互いに共有されることによって，コンパクトに圧縮された表現が可能となる，特に疎な組合せ集合の場合，ZDD ではその簡約化規則から，組合せ集合に登場しない要素に関する節点が削除されるため，同じ組合せ集合を表現する BDD よりも効率よく組合せ集合を表現・操作することができる．BDD と ZDD のデータ圧縮率の違いは，組合せ集合の各要素に含まれるアイテムの出現頻度に依るが，例えばアイテムの出現頻度が 1%である場合，ZDD は BDD よりも 100 倍コンパクトになる可能性がある．

2.3 BDD・ZDD の演算

論理関数 f と g を BDD・ZDD で表現したとき, この論理関数同士の AND/OR 演算は，同じ論理関数を特性関数とした組合せ集合の交わり (intersection)/結び (union) の演算にそのまま対応する．すなわち，論理関数 $f \wedge g, f \vee g$ を表現する BDD を，f と g を表現する BDD のまま計算するアルゴリズムが存在する [14]．ZDD の場合も同様であり，論理関数 $f \wedge g, f \vee g$ を表現する ZDD を，f と g を表現する ZDD のまま計算するアルゴリズムが存在する [12]. ZDD は ZDD 同士の様々な演算を，圧縮されたデータ量にほぼ比例する計算時間で実行でき

るという利点も持つ．すなわち，圧縮されたデータを，元のデータ量に戻すことなく，圧縮されたままの状態で高速に演算処理が可能である [12].

本研究で使用する演算は AND(∧)/OR(∨)/NOT(¬) の 3 つである．AND は 2 つの BDD・ZDD が表す論理関数の論理積，言い換えると集合族の交わり (intersection) を意味する．OR は 2 つの BDD・ZDD が表す論理関数の論理和，言い換えると集合族の結び (union) を意味する．NOT は 1 つの BDD・ZDD が表す論理関数の否定，言い換えるとべき集合との差を意味する．また，BDD・ZDD による⇒を含む論理式の表現は，次のように行うことができる．論理関数 f と g が与えられたとき，$f \Rightarrow g$ は $\bar{f} \vee g$ と等価である．また，論理関数は BDD で表現することができ，$\wedge$ や $\vee$ の論理演算を BDD のまま計算するアルゴリズムが知られている．従って，f と g を表現する BDD が与えられたときに，$\bar{f} \vee g$ を計算することで，$f \Rightarrow g$ を表現する BDD が構築できる．ZDD でも同様である．

2.4 ZDD による線型重み最小化

ZDD を用いることで，線型重み最適化が可能である [13]．ZDD の各変数に対して重みを設定することで，ZDD が表現する組合せ集合の中から，重みの総和が最小・最大の組合せを取得したり，上位複数個の解を表現する ZDD を取得したり，重みの総和が指定した値以上・以下・未満・より大きい組合せを表現する ZDD を取得したりすることができる．

本研究では，getMinimum 関数，weightGE 関数，weightRange 関数，そして getKLightest 関数を使用する．以下に，それぞれの関数の仕様を説明する．

getMinimum

ZDD 変数毎の重みを指定する整数を要素とするベクトル $weights$ を引数とするメソッドで，ZDDf に含まれる集合のうち，重み最小の集合とその重みを返す．

weightGE

ZDDf，整数 $bound$，そして ZDD 変数毎の重みを指定するための，整数を要素とするベクトル $weights$ を引数とし，f に含まれる集合のうち，重みが $bound$ 以上の集合からなる ZDD を返す関数である．

内部的には，区間メモ化探索技法 [13] と呼ばれる手法を用いている．区間メモ化探索技法を用いて，まず f に含まれる集合のうち，重みが $bound - 1$ 以下の集合からなる ZDD を構築し，これを weightLE($bound - 1$) とする．

f と weightLE($bound - 1$) の差集合を求めることで，目的の ZDD を得る．

weightRange

ZDDf，整数 $lower\_bound$，整数 $upper\_bound$，そして ZDD 変数毎の重みを指定する整数を要素とするベクトル $weights$ を引数とし，f に含まれる集合のうち，重みが $lower\_bound$ 以上，$upper\_bound$ 以下である集合のみからなる ZDD を返す関数である．

getKLightest

ZDD 変数毎の重みを指定するための整数を要素とするベクトル $weights$，整数 k を引数とするメソッドで，ZDDf に含まれる集合のうち，重みが小さい順に k 個以上の集合が含まれる ZDD を返す．

3 既存手法

3.1 CSPP の定式化

CSPP の解列挙の問題を，Jensen [11] に基づき定式化する．セル (コンテナが置かれる可能性がある位置) の配置については簡単のため 2 次元とし，水平方向 T，垂直方向 V の $T \cdot V = M$ 箇所のセルとする．セルの位置を表現する変数として $c_i (i \in \{1, 2, \ldots, M\})$ を用意する．図に示す通り，添え字は下左端セルを 1 として水平方向に増えていき，T 個ずつ垂直方向に重なるように並ぶ．

コンテナは 1 つずつ個別にではなく，コンテナ複数を含むグループ単位で区別されるものとし，グループの番号を $1, 2, \ldots, G$ とする．また，値 0 を「コンテナが置かれていない」を表現する特別なグループ番号とする．$c_i = g (g \in \{0, 1, \ldots, G\})$ によってセル c_i にグループ g のコンテナが置かれていることを表現する ($g = 0$ の場合はコンテナが置かれていないことを表現する)．グループごとのコンテナの個数については，インスタンスごとに定められているものとし，グループ 0 も含めたグループ毎のコンテナの数の合計がセルの総数 M になるものとする．

以上のように表現されるセルへのコンテナの配置に対して，様々な制約条件を考え，制約条件を満たすコンテナの配置を全て求める問題が，CSPP の解列挙の問題である．

先行研究において考慮される制約条件としては，次のようなものが存在する [11].

1. 1 つのセルに 20ft と 40ft のコンテナを同時に置くことはできない．

2. 20ft コンテナは 40ft コンテナの上に置くことはできない.
3. あるセルに 20ft コンテナが置かれるならば，同じセルにもう 1 つの 20ft コンテナが必ず置かれる.
4. コンテナは宙に浮かない (垂直座標が 2 以上の位置のセルにコンテナが置かれたとき，その下には必ず別のコンテナが置かれている).
5. 1 つのコンテナは 1 つのセルにのみ置かれる.
6. 1 つの列 (水平座標が等しいセルの集合) に置かれるコンテナの総重量は，重量上限を超えない.
7. 荷下ろしされる港の番号を，寄港する順番に $1, 2, \ldots$ とすると，水平座標が等しいセル集合の中で，コンテナの荷下ろし港の番号は垂直座標が小さい方から順に降順に並ぶ.

本研究では簡単のため，コンテナのサイズの 20ft と 40ft を区別しないこと，コンテナを個別にではなくグループで区別すること，列ごとの重量制限を考慮しないこと，コンテナの輸送先港を区別しないこととする．従って，上記 4. の制約条件のみを採用する.

3.2 解集合を表現する BDD の構築方法

Jensen [11] では以上の問題設定の上で解集合を表現する BDD の具体的な構築方法が述べられていないが，用いられていると考えられる方法を示す.

3.2.1 バイナリエンコード

セルを表現する変数をセル変数 c_i とする．セル変数 c_i に割り当てるグループ番号は一般に 2 値では表せない場合がある．一方，BDD と ZDD はともに 0-1 論理変数のみを扱うことができる．取りうる値が 2 値よりも多い変数を BDD・ZDD で扱うことは，次のように変数のバイナリエンコードを行うことによって実現できる.

1. 変数 x_i が取りうる値の数を $N(2 \leq N)$ とした時，1 変数あたり $\lceil \log(N) \rceil$ 個の 0-1 変数を用意して，$x_i^{\lceil \log(N) \rceil - 1}, x_i^{\lceil \log(N) \rceil - 2}, \ldots, x_i^0$ とする.
2. $x_i^{\lceil \log(N) \rceil - 1} x_i^{\lceil \log(N) \rceil - 2} \ldots x_i^0$ を x_i がとる値の 2 進数表示であるとする.

以上より，取りうる値が 0 から G である本問題設定では，1 つのセルあたり $\lceil \log(G+1) \rceil$ 個の 0-1 論理変数でバイナリエンコードを行う.

3.2.2 制約条件の表現

次の 2 つの制約条件を BDD で表現する.

制約条件 (1)

コンテナは宙に浮かない．

制約条件 (2)

グループ $g(g \in \{0, 1, \ldots, G\})$ のコンテナの数を $k_g(k_g \in \{0, 1, \ldots, M\},$ $\sum_{g \in 0,1,\ldots,G} k_g = M)$ とした時，値 g を取るセル変数 c_i の数は k_g 個である．

まず，制約条件 (1) は，コンテナは下から床または別のコンテナによる支えがない限り，置くことはできないということを意味する．この制約条件は，あるセルにコンテナが置かれないならば，その上のセルにもコンテナは置かれない，と言い換えられる．さらに，あるセル変数にグループ番号 0 が割り当てられたならば，垂直方向でそのセルの 1 つ上のセルを表す変数にも，グループ番号 0 が割り当てられなければならない，と言い換えることができる．これを論理式で表現すると，

$$c_i = 0 \Rightarrow c_{i+T} = 0, 0 \leq i \leq M - T \tag{1}$$

と表現できる．この論理式に対応する BDD を構築することで，制約条件 (1) を表現する．

続いて，制約条件 (2) については，次に示す手法で BDD として表現できる．まず，n 個の論理変数のうち，k 個の論理変数が 1 を取る時に真，それ以外のとき偽となる論理関数を考える．図 5 に示すように，論理変数に対応する節点を規則的に配置する．終端節点を左端から $1, 2, \ldots, k, \ldots, n$ 番目の終端節点としたとき，k 番目の終端節点のみを 1-終端節点，それ以外を 0-終端節点とする．こうして構築された DAG に対して BDD の簡約化規則を用いて得られた BDD は目的の論理関数を表現している．これを，${}_nC_k$BDD と呼ぶこととする．

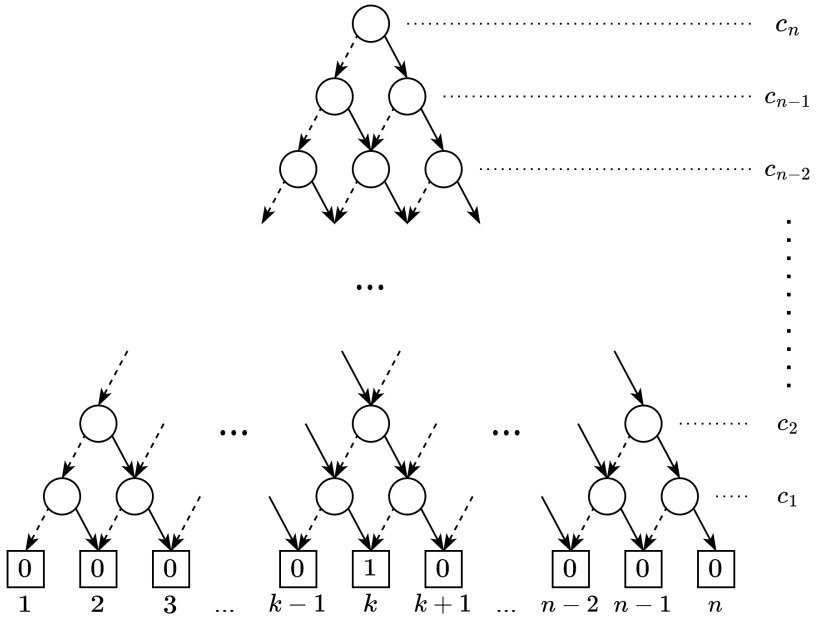

図 5: ${}_nC_k$ を表現する DAG

次に, 変数がバイナリエンコードされている場合を考える．ここでは, バイナリエンコードする前の変数をメタ変数, バイナリエンコードした変数をバイナリ変数と呼ぶこととする．n 個のメタ変数のうち, k 個のメタ変数が値 v を取るとき真, それ以外のとき偽となる論理関数を考える．例として 1 つのメタ変数あたり, 3 個のバイナリ変数によってバイナリエンコードされている場合に, 図 6 のような BDD は,「メタ変数が 5 すなわち 2 進数表示で 101_2 を取る時に真, それ以外のとき偽となる」という論理関数を表現する．これを, バイナリエンコードする前の, ${}_nC_k$BDD の簡約化前の DAG の節点の位置にそれぞれ配置することで, 図 7 のような DAG を得る．これは $n = 3, b = 3, k = 2, v = 5$ の場合の例である．この DAG に対して BDD の簡約化規則を用いて得られた BDD は, 目的の論理関数を表現している．任意の $n(n \geq 1), k(k \leq n), b(1 \leq b), v(0 \leq v \leq 2^b - 1)$ に対して,「1 個のメタ変数あたり b 個のバイナリ変数でバイナリエンコードされた n 個のメタ変数のうち, k 個のメタ変数が値 v を取るとき真, それ以外のとき偽となる」という論理関数を ${}_nC_k(b, v)$ とし，以降，これを表現する BDD を ${}_nC_k(b, v)$BDD と呼ぶこととする．変数が取りうる全ての v に対して ${}_nC_k(b, v)$BDD を構築することで，制約条件 (2) を表現する．

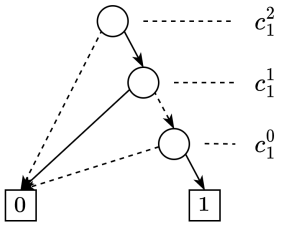

図 6: $c_1 = 5 = 101_2$ を表現する BDD

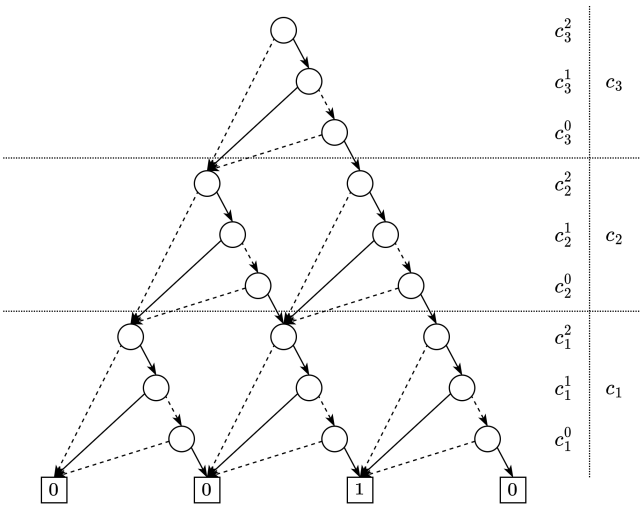

図 7: ${}_3C_2(3,5)$ を表現する DAG

3.2.3 解集合を表現する BDD の構築

解集合を表現する BDD は，次の手順で示すように．解が満たすべき制約条件を表す BDD 全ての intersection を取ることで得られる．

手順 1. 制約条件 (1) を表現する BDD を全ての i について構築し，全ての intersection を取る．これによって得られた BDD を NoFloatingBDD とする．

手順 2. 制約条件 (2) を表現する BDD を全ての v について構築し，これによって得られた BDD を v の値ごとに ${}_nC_k(b,v)$BDD とする．

手順 3. NoFloatingBDD と全ての v についての ${}_nC_k(b,v)$BDD の intersection を取る．

4 提案手法

既存手法の場合と同様の問題設定に対して，ZDD によって解集合を表現する手法と，コンテナに対して重量を導入し，線型重み最適化を行う手法を提案する．

4.1 解集合を表現する ZDD の構築方法

既存手法の場合と同様の問題設定に対して，ZDD によって解集合を表現する手法を提案する．

4.1.1 制約条件の表現

BDD の場合と同様，以下 2 つの制約条件を考える．

制約条件 (1)

コンテナは宙に浮かない．

制約条件 (2)

グループ $g(g \in \{0, 1, \ldots, G\})$ のコンテナの数を $k_g(k_g \in \{0, 1, \ldots, M\}, \sum_{g \in 0,1,\ldots,G} k_g = M)$ とした時，値 g を取るセル変数 c_i の数は k_g 個である．

まず，(1) については，論理式:$c_i = 0 \Rightarrow c_{i+T} = 0, \forall i$ に対応する ZDD を構築する．続いて，(2) については，${}_nC_k$BDD を参考にして構築する点は BDD の場合と同じであるが，バイナリエンコードされている場合の ZDD の構築方法が異なる．例として 1 つのメタ変数あたり，3 個のバイナリ変数によってバイナリエンコードされているとき，例えば図 8 のような ZDD は，「メタ変数が 5 すなわち 2 進数表示で 101_2 を取る時に真，それ以外のとき偽となる」という論理関数を表現する．これを，バイナリエンコードする前の，${}_nC_k$BDD の簡約化前の DAG の節点の位置にそれぞれ配置することで，図 9 のような DAG を得る．この DAG に対して ZDD の簡約化規則を用いて得られた ZDD は，目的の論理関数，すなわち ${}_nC_k(b, v)$ を表現している．以降，${}_nC_k(b, v)$ を表現する ZDD を ${}_nC_k(b, v)$ZDD と呼ぶこととする．

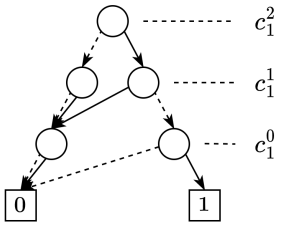

図 8: $c_1 = 5 = 101_2$ を表現する ZDD

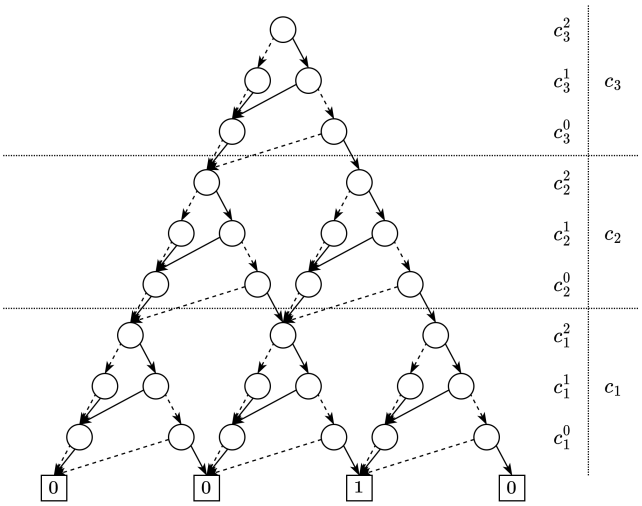

図 9: ${}_3C_2(3,5)$ を表現する DAG

4.1.2 解集合を表現する ZDD の構築

解集合を表現する ZDD は，次の手順で示すように．解が満たすべき制約条件を表す ZDD 全ての intersection を取ることで得られる．

手順 1. 制約条件 (1) を表現する ZDD を全ての i について構築し，全ての intersection を取る．これによって得られた ZDD を NoFloatingZDD とする．

手順 2. 制約条件 (2) を表現する ZDD を全ての v について構築し，これによって得られた ZDD を v の値ごとに ${}_nC_k(b,v)$ZDD とする．

手順 3. NoFloatingZDD と全ての v についての ${}_nC_k(b,v)$ZDD の intersection を取る．

4.2 コンテナ重量の表現と重心の最適化

ここまでの問題設定では，グループ毎の値は，「コンテナを置かない」を表現する 0 という値を除いて，グループ同士の区別以上の情報を持たないものだった．そこで，グループ毎の値に何らかの意味を付与することによって，より複雑な問題設定を検討する．ここでは，グループ毎の値がコンテナの重量に対応した値であるという設定を導入する．コンテナの重量は，コンテナが持つ情報の中でも，船の安定性や構造上の制約を考える上で重要な性質である．これを考慮することで，現実に近い問題を考えることができる．本研究では，グルー

プ番号 g のコンテナは $5 \cdot g$ トンであるとする．コンテナの重量を導入した際に，制約条件と目的のそれぞれで新たに考慮すべき点が生まれる．

まず，新たな制約条件として，

制約条件 (3) あるコンテナの上には，そのコンテナより重いコンテナをおいてはならない

という制約条件を考える．これは，コンテナの構造上の制約条件や積まれたコンテナ集合の安定性などを考慮する上で用いられる制約条件である．この制約条件は，あるセル変数にグループ番号 $j(j \in \{1, 2, \ldots, G\})$ が割り当てられたならば，垂直方向でそのセルの 1 つ上のセルを表す変数には，グループ番号 $0, 1, \ldots, j$ のいずれかが割り当てられなければならない，と言い換えることができる．これを論理式で表現すると，

$$c_i = j \Rightarrow \bigvee_{k=0,\ldots,j} c_{i+T} = k, i \in \{1, 2, \ldots, M-T\}, j \in \{1, 2, \ldots, G\} \qquad (2)$$

と表現できる．この論理式に対応する ZDD を構築する．解集合を表現する ZDD は，次の手順で構築する．

手順 1. 制約条件 (1) を表現する ZDD を全ての i について構築し，全ての intersection を取る．これによって得られた BDD を NoFloatingZDD とする．

手順 2. 制約条件 (2) を表現する ZDD を全ての v について構築し，これによって得られた ZDD を v の値ごとに ${}_nC_k(b, v)$ZDD とする．

手順 3. 制約条件 (3) を表現する ZDD を構築し，これを NoHeavierAboveZDD とする．

手順 4. NoFloatingZDD と NoHeavierAboveZDD と全ての v についての ${}_nC_k(b, v)$ZDD の intersection を取る．

続いて，重心の位置の最適化を考える．重心の位置は船の航行の安全上重要な指標の一つである．垂直方向と水平方向 2 つの重心を考えることとし，それぞれ次のように定義する．i 番目のセルに置かれたコンテナの重量を w_i とし，i 番目のセルの垂直方向の座標 v_i を

$$v_i = \lceil \frac{i}{T} \rceil, \forall i \qquad (3)$$

とし，水平方向の座標 t_i を T が偶数のとき

$$t_i = \begin{cases} ((i-1) \bmod T) + \frac{1-T}{2} & (((i-1) \bmod T) > \frac{T}{2}) \\ ((i-1) \bmod T) - \frac{T}{2} & (((i-1) \bmod T) \leq \frac{T}{2}) \end{cases} \forall i \qquad (4)$$

T が奇数のとき

$$t_i = ((i-1) \bmod T) + \frac{1-T}{2}, \forall i \tag{5}$$

とする．また

$$\text{垂直重心} = \frac{\sum_{i=1,\ldots,M} w_i \cdot v_i}{\sum_{i=1,\ldots,M} w_i} \tag{6}$$

$$\text{水平重心} = \frac{\sum_{i=1,\ldots,M} w_i \cdot t_i}{\sum_{i=1,\ldots,M} w_i} \tag{7}$$

とする．また，本問題設定においては，インスタンス毎のコンテナの総重量は，コンテナの積載の仕方によらず定数であるため，最適化のための計算の際には，

$$\text{垂直モーメント} = \sum_{i=1,\ldots,M} w_i \cdot v_i \tag{8}$$

$$\text{水平モーメント} = \sum_{i=1,\ldots,M} w_i \cdot t_i \tag{9}$$

を用いる．

4.2.1 コンテナ重量の表現

b 個のバイナリ変数によってバイナリエンコードされたセル変数 c_i の $l(1 \leq l \leq b)$ 桁めが $5 \cdot 2^{l-1}$ トン分の重量を表現しているものとすると，セル c_i に置かれたグループ番号 g のコンテナ，すなわち $5 \cdot g$ トンのコンテナの重量は次のように表現できる．

$$\text{セル } c_i \text{に置かれたコンテナの重量} = \sum_{l=1,2,\ldots,b} c_i^{l-1} \cdot (5 \cdot 2^{l-1}), \forall i \tag{10}$$

すると，このバイナリ重量と垂直方向の座標および水平方向の座標の積によって，垂直方向と水平方向それぞれにおける，最適化のためのバイナリ変数毎の重みを次のように設定することができる．すなわち，

$$\text{垂直重み} = 5 \cdot 2^{l-1} \cdot v_i, \forall i, \forall l \tag{11}$$

$$\text{水平重み} = 5 \cdot 2^{l-1} \cdot t_i, \forall i, \forall l \tag{12}$$

とする．

4.2.2 重心の最適化

ZDD を用いると，getMinimum 関数を利用することで線型重み和が最小の解を得られるだけでなく，weightGE 関数や getKLightest 関数を用いることで，線型重み和が一定の値以上の複数の解を表現する ZDD を得たり，線型重み和最小化の複数の上位解を表現する ZDD を得ることができる．また，weightRange 関数を用いることで，線型重み和が一定の範囲内の値を取る複数の解を表現する ZDD を得ることができる．実用上，多様な解を選択肢として提示することができることは，解の決定を行う上で有用であると考えられる．

次の方法で，解集合を表現する ZDD の中から，垂直モーメントが最小の解を 1 つおよび上位解を複数，水平モーメントの絶対値が最小の解を 1 つおよび上位解を複数取得する．また，水平モーメントが一定の範囲内の値を取る解を取得したのち，その中から垂直モーメントが最小の解を 1 つおよび上位解を複数取得する．

求解 (I) 垂直モーメントが最小の解 1 つの取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．

(b) 式 11 で定められるバイナリ変数毎の重みを引数とし，getMinimum 関数を targetZDD に適用する．

求解 (II) 垂直モーメント最小化における複数の上位解の取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．なお，求解 (I) で求めた targetZDD を再利用する．

(b) 式 11 で定められるバイナリ変数毎の重みを引数とし，K=10, strict=1 として getKLightest 関数を targetZDD に適用する．

求解 (III) 水平モーメントの絶対値が最小の解 1 つの取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．なお，求解 (I) で求めた targetZDD を再利用する．

(b) 式 12 で定められるバイナリ変数毎の重みを引数とし，bound を 0 として weightGE 関数を targetZDD に適用する．これによって得られる水平重み和が 0 以上の解集合を表現する ZDD を GE0_targetZDD とする．

(c) 式 12 で定められるバイナリ変数毎の重みを引数とし，getMinimum 関数を GE0_targetZDD に適用する．

求解 (IV) 水平モーメントの絶対値最小化における複数の上位解の取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．なお，求解 (I) で求めた targetZDD を再利用する．

(b) 式 12 で定められるバイナリ変数毎の重みを引数とし，bound を 0 として weightGE 関数を targetZDD に適用する．これによって得られる水平重み和が 0 以上の解集合を表現する ZDD を GE0_targetZDD とする．なお, 求解 (III) で求めた GE0_targetZDD を再利用する．

(c) 式 12 で定められるバイナリ変数毎の重みを引数とし，K=10, strict=1 として getKLightest 関数を GE0_targetZDD に適用する．

求解 (V) 水平モーメントが一定の範囲内の値を取る解の中からの，垂直モーメントが最小になる解の取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．なお，求解 (I) で求めた targetZDD を再利用する．

(b) 式 12 で定められるバイナリ変数毎の重みを引数とし, lower_bound を -10, upper_bound を 10 として weightRange 関数を targetZDD に適用する．これによって得られる水平モーメントが -10 以上 10 以下の解集合を表現する ZDD を Range_targetZDD とする．

(c) 式 11 で定められるバイナリ変数毎の重みを引数とし，getMinimum 関数を Range_targetZDD に適用する．

求解 (VI) 水平モーメントが一定の範囲内の値を取る解の中からの，垂直モーメント最小化における複数の上位解の取得は，次の方法で行う．

(a) 解集合を表現する ZDD を構築する．これを targetZDD とする．なお，求解 (I) で求めた targetZDD を再利用する．

(b) 式 12 で定められるバイナリ変数毎の重みを引数とし, lower_bound を -10, upper_bound を 10 として weightRange 関数を targetZDD に適用する．これによって得られる水平モーメントが -10 以上 10 以下の解集合を表現する ZDD を Range_targetZDD とする．なお, 求解 (V) で求めた Range_targetZDD を再利用する．

(c) 式 11 で定められるバイナリ変数毎の重みを引数とし，K=10, strict=1 として getKLightest 関数を Range_targetZDD に適用する．

5 実験結果

5.1 実験 1: BDD と ZDD による解集合の表現

5.1.1 インスタンス

実験 1 として，BDD と ZDD による，コンテナ重量を考慮しない解集合の表現を行う．表 1 で示すインスタンスに対して，実験を行った．T は水平方向のセルの数，V は垂直方向のセルの数，M は $M = T \cdot V$ で計算されるセル数の合計，b はバイナリエンコードの桁数，n はバイナリ変数の数，$k_g (0 \leq g \leq 2^b - 1, g$ は整数$)$ は値 g を取るメタ変数の数すなわちグループ g のコンテナの数をそれぞれ表す．

インスタンス a から n については，セル数 M の増加に伴う BDD・ZDD の構築時間や節点数の増減の比較，および M が同じインスタンスについては，グループ毎のコンテナ数の設定の違いによる BDD・ZDD の構築時間や節点数の増減の比較を行うためのインスタンスである．また，M もグループ毎のコンテナ数の設定も同じインスタンス (g と h，および j と k) については，バイナリエンコードの桁数による違いを比較するためのインスタンスである．また，インスタンス o から s については，水平方向と垂直方向のセル数の違いによる BDD・ZDD の構築時間や節点数の増減の比較を行うためのインスタンスである．

実験プログラムは C++で実装し，MacBook Pro 2017 (CPU; 2.3GHz デュアルコア Intel Core i5, メモリ; 8GB) で実行した．

表 1: 実験 1 のインスタンスのリスト

インスタンス	T	V	b	M	n	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7
a	3	3	2	9	18	3	3	3	0	-	-	-	-
b	3	3	2	9	18	0	3	3	3	-	-	-	-
c	3	3	2	9	18	2	2	2	3	-	-	-	-
d	3	3	3	9	27	3	3	3	0	0	0	0	0
e	3	3	3	9	27	0	0	0	3	0	0	3	3
f	3	3	3	9	27	2	1	1	1	1	1	1	1
g	4	4	2	16	32	4	4	4	4	-	-	-	-
h	4	4	3	16	48	4	4	4	4	0	0	0	0
i	4	4	3	16	48	2	2	2	2	2	2	2	2
j	5	5	2	25	50	7	6	6	6	-	-	-	-
k	5	5	3	25	75	7	6	6	6	0	0	0	0
l	5	5	3	25	75	4	3	3	3	3	3	3	3
m	6	6	2	36	72	9	9	9	9	-	-	-	-
n	6	6	2	36	72	18	9	9	0	-	-	-	-
o	3	12	3	36	108	5	5	5	5	4	4	4	4
p	4	9	3	36	108	5	5	5	5	4	4	4	4
q	6	6	3	36	108	5	5	5	5	4	4	4	4
r	9	4	3	36	108	5	5	5	5	4	4	4	4
s	12	3	3	36	108	5	5	5	5	4	4	4	4

5.1.2 結果

表 1 で示すインスタンスに対して, 制約条件を満たす解集合を表現する BDD・ZDD を構築した結果を表 2 に示す. インスタンス毎に, 解集合を表現する BDD (targetBDD)・ZDD (targetZDD) の構築 (制約条件を表現する BDD・ZDD 全ての intersection を取る) にかかった時間 (単位は μs) と, targetBDD・targetZDD の節点数を示す. timeout は 120 秒以上としている.

表 2: 実験 1 の結果

	BDD		ZDD	
インスタンス	time (μs)	節点数	time (μs)	節点数
a	193	214	74	112
b	59	161	131	147
c	251	228	597	165
d	266	305	246	112
e	113	260	135	246
f	1,076	1,542	766	1,052
g	1,391	3,361	957	2,673
h	5,318	4,698	7,267	2,673
i	12,281	21,395	14,601	16,903
j	13,001	26,995	23,992	23,882
k	72,437	37,623	76,532	23,882
l	1,131,134	817,167	968,694	675,249
m	155,246	175,701	147,714	160,599
n	98,911	71,054	96,424	43,150
o	1,224,631	1,065,986	1,042,252	934,274
p	2,848,767	2,472,732	2,430,412	2,053,327
q	16,620,412	11,476,057	13,193,775	9,959,210
r	101,059,689	48,450,191	113,317,629	45,461,165
s	timeout	-	timeout	-

5.1.3 考察

BDD・ZDD ともに構築時間および節点数の違いは，解となるコンテナ配置の組合せ数の大きさによるものと考えられる．また，インスタンス m から s の結果で確認できるように，セル数 M が同じであっても，水平方向のセル数と垂直方向のセル数が異なると，解を表現する BDD・ZDD の構築時間と節点数が大幅に変わることが分かった．これは，内部的なバイナリ変数の順番と制約条件との兼ね合いによるものと考えられる．制約条件 (1) は，上下に隣り合うセル同士の関係を示しているが，BDD・ZDD の変数順はセル変数の添え字の順番になっているため，上下に隣り合うセルそれぞれに対応するバイナリ変数の BDD・ZDD 内部での深さの差は，T の大きさに比例して大きくなる．この深さの差が大きいほど，同じ制約条件を適用した解であっても，それを表現する

BDD・ZDD の節点数が大きくなるものと考えられ，実験の結果はこれによるものと考えられる．全てのインスタンスについて，解集合を表現する BDD・ZDD の節点数については ZDD の節点数の方が BDD の節点数よりも少なくなっている．節点数の削減率を

$$1 - \frac{\text{ZDD の節点数}}{\text{BDD の節点数}} \tag{13}$$

とすると，インスタンス d において削減率が最大で 63.3%，インスタンス e において削減率が最小で 5.39%となっている．ZDD は，BDD との簡約化規則の違いから，値 0 を取る変数が多いほど，BDD と比較して節点数が少なくなる傾向があるが，このインスタンス d と e の違いとしては，前者はグループ番号が 2 進数表記で $000_2, 001_2, 010_2$ で表現されるコンテナが 3 つずつ，後者は前者はグループ番号が 2 進数表記で $011_2, 011_2, 111_2$ で表現されるコンテナが 3 つずつ，というインスタンスである．解となる変数への値の割り当てにおいて，0 を割り当てられる変数の数の違いが，削減率の違いの原因となっているものと考えられる．

5.2 実験 2: 重量バランスを考慮した解の ZDD による列挙

5.2.1 インスタンス

実験 2 として，重量バランスを考慮した解の ZDD による列挙を行う．表 3 で示すインスタンスに対して，実験を行った．

インスタンス A から D については，セル数が 20 の設定で，一部のグループのみが均等に含まれているもの (A, B)，全てのグループが含まれており可能な限り均等なコンテナ数の配分になっているもの (C)，全てのグループが含まれておりコンテナ数の配分がまばらになっているもの (D) を比較した．インスタンス E から G については，セル数が 25 の設定で，一部のグループのみが均等に含まれているもの (E)，全てのグループが含まれておりグループ毎のコンテナ配分に偏りがあるもの (F)，グループ毎のコンテナの配分がまばらになっているもの (G) を比較した．インスタンス H から K については，セル数が 30 の設定で，一部のグループのみが均等に含まれているもの (H, I)，一部のグループのみがまばらに含まれているもの (J)，全てのグループが含まれておりコンテナ数の配分がまばらになっているもの (K) を比較した．

表 3: 実験 2 のインスタンス

インスタンス	T	V	b	M	n	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7
A	5	4	3	20	60	5	5	5	5	0	0	0	0
B	5	4	3	20	60	4	4	4	4	4	0	0	0
C	5	4	3	20	60	3	3	3	3	2	2	2	2
D	5	4	3	20	60	1	3	5	2	1	1	5	2
E	5	5	3	25	75	5	5	5	5	5	0	0	0
F	5	5	3	25	75	10	5	5	1	1	1	1	1
G	5	5	3	25	75	3	2	1	5	3	4	0	7
H	5	6	3	30	90	6	6	6	6	6	0	0	0
I	5	6	3	30	90	0	0	0	6	6	6	6	6
J	5	6	3	30	90	6	8	5	7	4	0	0	0
K	5	6	3	30	90	5	8	4	7	3	1	1	1

5.2.2 結果

表 3 で示すインスタンスに対して，求解 (I) から (VI) を適用した結果を表 4 および表 5 に示す．

表 4 における getMinimum(I)，getMinimum(III)，getMinimum(V) はそれぞれ求解 (I)，(III)，(V) において getMinimum 関数を適用した結果を意味し，関数の実行にかかった時間を示している．

表 5 における targetZDD は制約条件を満たす解集合を表現する ZDD を意味し，この ZDD の構築にかかった時間およびこの ZDD の節点数とこの ZDD が表現する集合数 (解の数) を示している．getKLightest(II)，getKLightest(IV)，getKLightest(VI) は，それぞれ求解 (II)，(IV)，(VI) において getKLightest 関数を適用して得られた ZDD を意味し，また weightGE(III) は求解 (III) において weightGE 関数を適用して得られた ZDD を意味し，そして weightRange(V) は求解 (V) において weightRange 関数を適用して得られた ZDD を意味する．それぞれの ZDD の構築にかかった時間およびこの ZDD の節点数とこの ZDD が表現する集合数 (解の数) を示している．

表 4: 実験 2 の結果 (getMinimum 演算にかかる時間)

	time (μs)		
インスタンス	getMinimum(I)	getMinimum(III)	getMinimum(V)
A	29,366	31,705	31,991
B	116,583	144,203	134,525
C	855,183	2,185,709	1,896,322
D	355,147	535,266	688,213
E	485,439	992,894	1,023,174
F	249,196	581,228	496,420
G	1,529,394	4,661,351	4,168,104
H	1,032,662	2,488,406	2,698,113
I	1,642,553	3,811,678	3,923,955
J	594,585	1,507,056	1,504,209
K	1,598,877	6,811,277	5,900,422

表 5: 実験 2 の結果 (getMinimum 演算以外の演算にかかる時間と得られた ZDD の節点数とその ZDD が表現する解の数 (集合の数))

	time (μs)					
インスタンス	targetZDD	getKLightest(II)	weightGE(III)	getKLightest(IV)	weightRange(V)	getKLightest(VI)
A	1,740,163	35,905	36,663	427,428	77,208	63,481
B	2,647,817	182,156	149,083	655,006	231,765	274,281
C	3,740,251	1,343,751	2,210,580	11,214,609	5,380,181	4,953,174
D	1,921,681	368,507	467,321	2,869,082	1,183,148	1,049,071
E	7,772,198	634,527	1,796,720	5,638,473	1,515,693	1,993,866
F	4,991,729	404,469	622,448	3,706,055	1,166,104	1,062,361
G	6,538,422	2,815,424	4,708,259	27,367,443	5,558,461	11,429,919
H	16,615,839	1,600,819	2,380,680	12,448,688	3,300,340	6,645,646
I	2,928,225	3,290,597	2,638,914	19,994,964	4,627,546	7,991,247
J	11,187,052	852,168	1,397,046	8,968,171	2,096,659	2,983,312
K	14,086,785	2,579,061	7,570,239	47,178,350	7,551,941	25,263,133

	節点数					
インスタンス	targetZDD	getKLightest(I)	weightGE(III)	getKLightest(IV)	weightRange(V)	getKLightest(VI)
A	23,436	135	28,832	14,505	26,266	128
B	94,045	60	124,309	60,547	105,911	323
C	609,576	135	1,246,442	690,021	1,069,865	1,128
D	157,077	135	349,557	210,319	318,061	1,034
E	279,270	199	469,622	262,541	420,671	190
F	197,713	118	395,222	193,333	316,972	102
G	991,225	107	2,253,771	1,282,983	2,126,842	688
H	668,860	70	1,397,451	876,677	1,293,905	348
I	1,075,195	114	2,132,007	1,383,263	2,012,347	498
J	372,251	63	890,304	566,413	849,843	323
K	751,044	127	2,687,896	1,859,772	2,497,219	809

	集合数					
インスタンス	targetZDD	getKLightest(I)	weightGE(III)	getKLightest(IV)	weightRange(V)	getKLightest(VI)
A	261,331	61	135,919	10,507	52,091	43
B	2,224,955	2,500	1,149,986	75,017	371,975	1,402
C	146,092,390	180,000	74,762,462	3,432,534	17,107,536	71,072
D	23,024,040	60,000	11,770,354	516,668	2,576,746	20,924
E	22,069,251	81	11,351,980	634,709	3,161,127	57
F	25,393,800	120	12,998,801	603,802	3,011,284	38
G	203,629,040	10,000	103,606,398	3,583,756	17,903,960	3,924
H	164,176,640	2,500	84,157,678	4,138,716	20,609,930	1,402
I	164,176,640	2,500	84,157,678	4,138,716	20,609,930	1,402
J	112,807,815	625	57,934,051	3,060,287	15,232,319	381
K	2,726,183,870	30,000	1,395,200,861	64,217,852	320,036,192	10,748

図 10 から図 15 に，例としてインスタンス O で得られた解を示す．なお，求解 (II)，(IV)，(VI) においては複数の上位解を列挙しているが，以下に示す解はそれらの中からランダムに 1 つを選んだものである．セル内の数字は，セル毎

に割り当てられたグループが持つ重量を表現する．

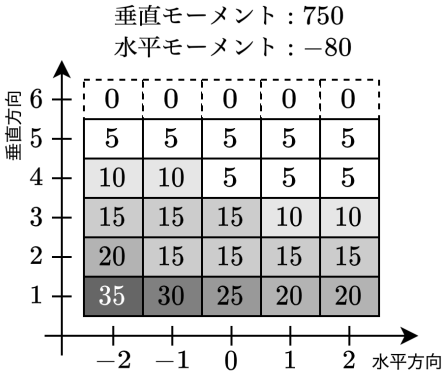

図 10: 求解 (I) で得られた解

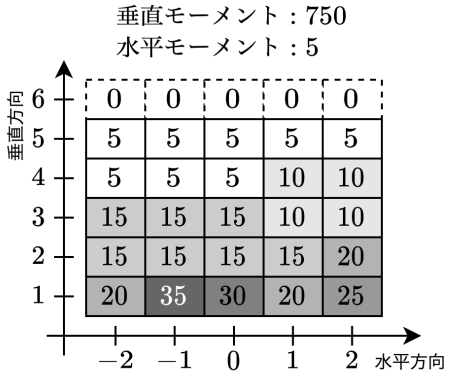

図 11: 求解 (II) で得られた解

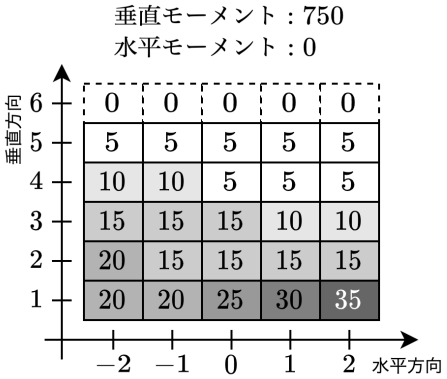

図 12: 求解 (III) で得られた解

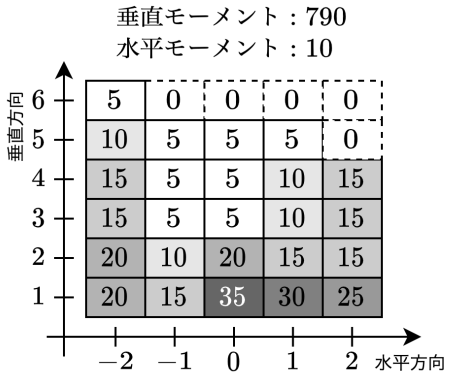

図 13: 求解 (IV) で得られた解

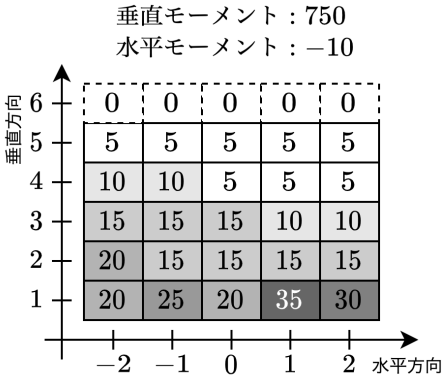

図 14: 求解 (V) で得られた解

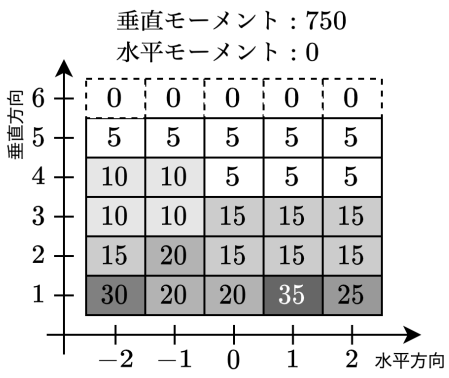

図 15: 求解 (VI) で得られた解

5.2.3 考察

getMinimum や getKLightest，weightGE，weightRange の実行にかかる時間については，おおよそ引数に取る ZDD の節点数に比例した時間がかかる．これは，これらの関数の実行時に探索する節点数が，引数に取る ZDD の節点数に比例することによるものと考えられる．

getKLightest によって得られた ZDD の節点数は，引数に取る ZDD の節点数に依存していない．これは，引数に取る ZDD の内部的な構造と，$strict$ を 1 としたことによるものと考えられる．すなわち，$strict$ を 1 としていることによって，返り値の ZDD が表現する集合数は $K = 10$ 個以上であるものになっているが，この集合数の判定を行った対象の ZDD が表現する集合数が 10 以上でありさえすれば良いので，返り値の ZDD の節点数は引数に取る ZDD の元々の節点数には依存しないものと考えられる．

weightGE(III) で得られた ZDD が表現する集合数は，おおよそ targetZDD が表現する集合数の半分強 (targetZDD が表現する集合数の 50.9%から 51.7%) となっている．ある解が表現するコンテナの配置について，水平方向に反転させた配置 (水平座標を -1 倍した解) は元の配置の水平モーメントの -1 倍の値を水平モーメントとして持つ配置であって，これも解集合に含まれる．従って，水平モーメントが 0 よりも大きい解の数を N_{GT0} とし，水平モーメントが 0 である解の数を N_0 とすると，targetZDD が表現する解の数は $2 \cdot N_{GT0} + N_0$ となる．weightGE(III) で得られた ZDD が表現する集合数は $N_{GT0} + N_0$ であり，このような値となっていると考えられる．

解集合を表現する ZDD の節点数について，全体として，グループ毎のコンテナ数の設定に伴う，制約条件を満たすコンテナ配置の組合せ数に依存して増減する傾向があることが分かる．

インスタンス H と I の結果について，targetZDD や getKLightest，weightGE，weightRange によって得られた ZDD の節点数は異なるが，表現する集合数は等しくなっている．これは，この 2 つのインスタンスにおける，グループ毎のコンテナの数の設定が，「(グループ番号は異なるものの，) 5 つのグループのコンテナがそれぞれ 6 つずつある」という点で等しく，配置の組合せ数が等しいことによるものと考えられる．インスタンス H の節点数が少ないことは，実験 1 の場合と同様，グループ番号の違いにより，解となる変数への値の割り当てにおいて，0 を割り当てられる変数の数が多いことによるものと考えらえる．

6　おわりに

本研究では，CSPP の解列挙の問題に ZDD を用いて，BDD を用いた場合と比較して実験を行った全てのインスタンスで節点数が少なくなることを確認した．また，適切に重みを設定し，ZDD を用いた線形重み最適化の手法を用いることで，垂直方向のモーメント，水平方向のモーメント，あるいはその両方を考慮したときの最適解および上位解を取得できることを確認した．

今後の課題としては，本研究で考慮できなかった制約条件の考慮や問題設定の拡張が挙げられる．具体的には，3 次元座標の導入のほか，位置ごとの重量上限の考慮，コンテナのサイズ (20ft, 40ft, 45ft) の区別，コンテナの種類 (DRY, REEFER, OOG, etc.) の区別，コンテナの内容物の IMDG に基づく区別と，それらの区別に基づく制約条件，などが挙げられる．

これらの条件を考慮する拡張を行えば，現実のコンテナ積載計画を考える業務の支援を行うシステムの構築が検討できる．

謝辞

本論文の執筆にあたりご指導頂きました指導教員の川原純先生に深く感謝いたします．また，研究を進める上でご助言を頂いた湊真一先生をはじめ，研究会を通して様々な意見を下さり，議論を共にさせていただきました湊研究室の皆様にも心よりお礼申し上げます．

参考文献

[1] Shields, J. J.: Containership stowage: A computer-aided preplanning system, *Marine Technology and SNAME News*, Vol. 21.04, pp. 370–383 (1984).

[2] International Maritime Organization: IMDG code : International maritime dangerous goods code, Vol. 41-22 (2023).

[3] Avriel, M., Penn, M., Shiprer, N. and Witteboon, S.: Stowage planning for container ships to reduce the number of shifts, *Annals of Operations Research*, Vol. 76, pp. 55–71 (1998).

[4] Liu, F.: Multi‐objective optimization for stowage planning of large

containership (2012).

[5] Ambrosino, D., Paolucci, M. and Sciomachen, A.: Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem, *Flexible Services and Manufacturing Journal*, Vol. 27, pp. 263–284 (2013).

[6] Ambrosino, D., Anghinolfi, D., paolucci, M. and Sciomachen, A.: A new three-step heuristic for the Master Bay Plan Problem, *Maritime Economics Logistics*, Vol. 11, pp. 98–120 (2009).

[7] Pacino, D.: An LNS Approach for Container Stowage Multi-port Master Planning, *International Conference on Computational Logistics* (2013).

[8] Dubrovsky, O., Levitin, G. and Penn, M.: A Genetic Algorithm with a Compact Solution Encoding for the Container Ship Stowage Problem, *Journal of Heuristics*, Vol. 8, pp. 585–599 (2002).

[9] Zhang, C., Guan, H., Yuan, Y., Chen, W. and Wu, T.: Machine learning-driven algorithms for the container relocation problem, *Transportation Research Part B: Methodological*, Vol. 139, pp. 102–131 (2020).

[10] Akers, S. B.: Binary Decision Diagrams, *IEEE Transactions on Computers*, Vol. C-27, pp. 509–516 (1978).

[11] Jensen, R. M., Leknes, E. and Bebbington, T.: Fast Interactive Decision Support for Modifying Stowage Plans Using Binary Decision Diagrams, *Prceedings of the International MultiConference of Engineers and Computer Scientists 2012* (2012).

[12] ichi Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems, *DAC '93: Proceedings of the 30th international Design Automation Conference*, pp. 272–277 (1993).

[13] ichi Minato, S., Banbara, M. et al.: Interval-Memoized Backtracking on ZDDs for Fast Enumeration of All Lower Cost Solutions, *arXiv:2201.08118* (2022).

[14] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, no. 8, pp. 677–691 (1986).