多次元0-1ナップサック問題に対する修復オペレーターを用いた局所探索法の提案

沼田研究室
4402066　秀方友一郎

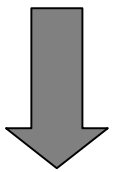
発表構成

1. 本研究の背景と目的
2. ナップサック問題
3. 代表的な解法
4. 0-1空間の局所探索法
5. 代理制約式と修復オペレーター
6. 有効勾配法を用いた初期解の生成
7. 実験
8. まとめ
9. 参考文献

1.1 研究背景

現実社会に現れる様々な最適化問題

ex)**資金計画、配送計画、生産計画** …etc.



多くの場合

組合せ最適化問題として定式化できる

ex)**巡回セールスマン問題、施設配置問題**

ナップサック問題 …etc.

1.1 研究背景

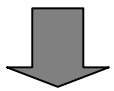
組合せ最適化問題の・・

問題規模が小さい場合

全数探索によって厳密解が求まる

問題規模が大きい場合

全数探索が困難となる



・近似解法が必要となる

・様々な解法が考えられている

本研究では，多次元0-1ナップサック問題に対する近似解法について研究する

1.2 研究目的

多次元0-1ナップサック問題に対する近似解法の提案

(1) 修復オペレーターを用いた0-1空間の局所探索法
→既存の方法と比較する

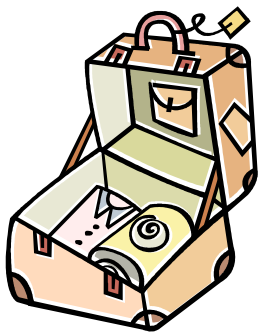
(2) メタ戦略を導入し、さらなる近似解の精度向上を目指す

2.1　0-1ナップサック問題

限られた容量のナップサックに品物を詰める問題

- **品物は複数ある**
- **品物はそれぞれ異なる重さと価値を持つ**

限られた容量の中でなるべく価値を最大にする品物の詰め方を決定する



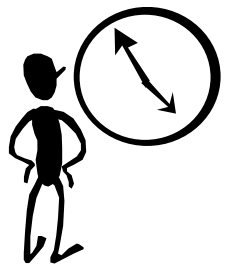
2.2 多次元0-1ナップサック問題

(MKP:Multidimensional Knapsack Problem)

- 0-1ナップサック問題の制約が2つ以上ある問題

  例：複数のプロジェクトを計画

  - 材料の制約
  - 時間の制約
  - 人の制約



全ての制約条件を満たし，期待利益を最大にするプロジェクトの組合せを決定する

2.2.1 MKP定式化(1) -記号の定義-

定式化の際に用いる記号

入力データ

- c_j ：プロジェクト j を実行することで得られる利益
- a_{ij} ：プロジェクト j が必要とする第 i 資源の量
- b_i ：第 i 資源の利用可能量

決定変数 x_j

- プロジェクト j を実行するとき、1
- プロジェクト j を実行しないとき、0

2.2.2 MKP定式化 (2)

多次元0－1ナップサック問題

$$
(\mathrm{P}) \quad
\begin{aligned}
&\text{maximize} && \sum_{j=1}^{n} c_j x_j \quad j = 1,2,\ldots,n && (2.1)\\
&\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \le b_{i,}\ i = 1,2,\ldots,m && (2.2)\\
& && x_j \in \{0,1\}
\end{aligned}
$$

目的関数と制約式の全ての係数 i, j に対して $c_j > 0, a_{ij} \ge 0, b_i > 0,$

$\sum_{j=1}^{n} a_{ij} > b_i$ となるMKPを考える (問題として意味を持つ仮定)

目的関数値をできるだけ大きくする x の0-1パターンを求める

3. ＭＫＰに対する代表的な解法

- **構築法**
  - **例：有効勾配法**
- **改善法**
  - **例：0-1空間の局所探索法，順列空間の局所探索法**
- **メタ戦略**
  - **例：遺伝的アルゴリズム，タブー探索**

比較的に短時間で良い解を算出する0-1空間の局所探索法に注目

4. 局所探索法

初期解を少し変更してできる解集合（近傍）を探索し，改善解があれば更新

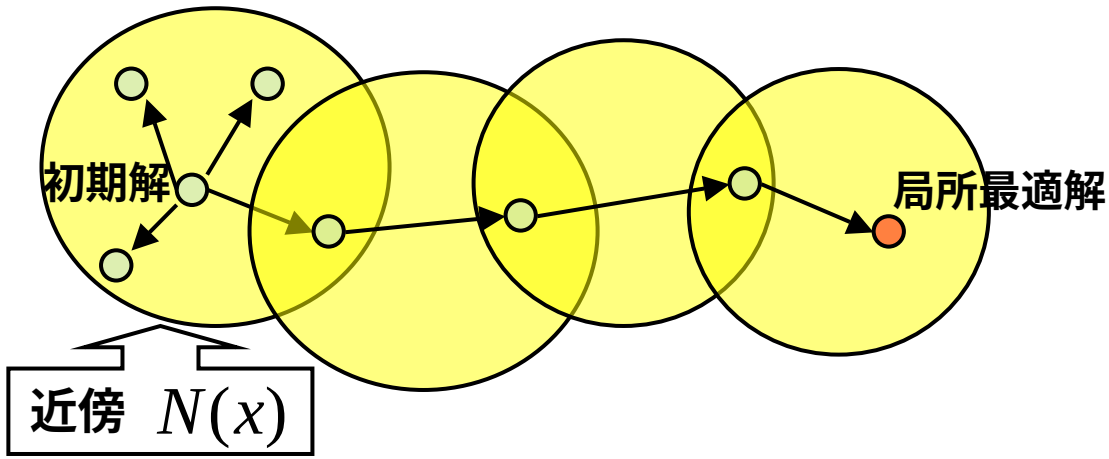


図4.1：近傍と局所探索法

- 近傍をどうするか？
- 初期解をどうするか？

4.1 局所探索法の近傍

0-1空間の局所探索法の近傍

現在の解 x **の**0-1**ベクトルを・・**

1ヶ所反転した解集合 $N_1(x)$

2ヶ所反転した解集合 $N_2(x)$

$$N_1(x) \cup N_2(x)$$

近傍 $N(x)$

例：現在解 $(1,0,0,1)$**の近傍** $N(x)$

$$N(x)\begin{cases} N_1(x) \cdots (0,0,0,1),(1,1,0,1),(1,0,1,1),(1,0,0,0) \\ N_2(x) \cdots (0,1,0,1),(0,0,1,1),(0,0,0,0),(1,1,1,1),(1,1,0,0),(1,0,1,0) \end{cases}$$

実行可能な解もあれば，実行不可能な解もある

4.2 従来の0-1空間の局所探索法

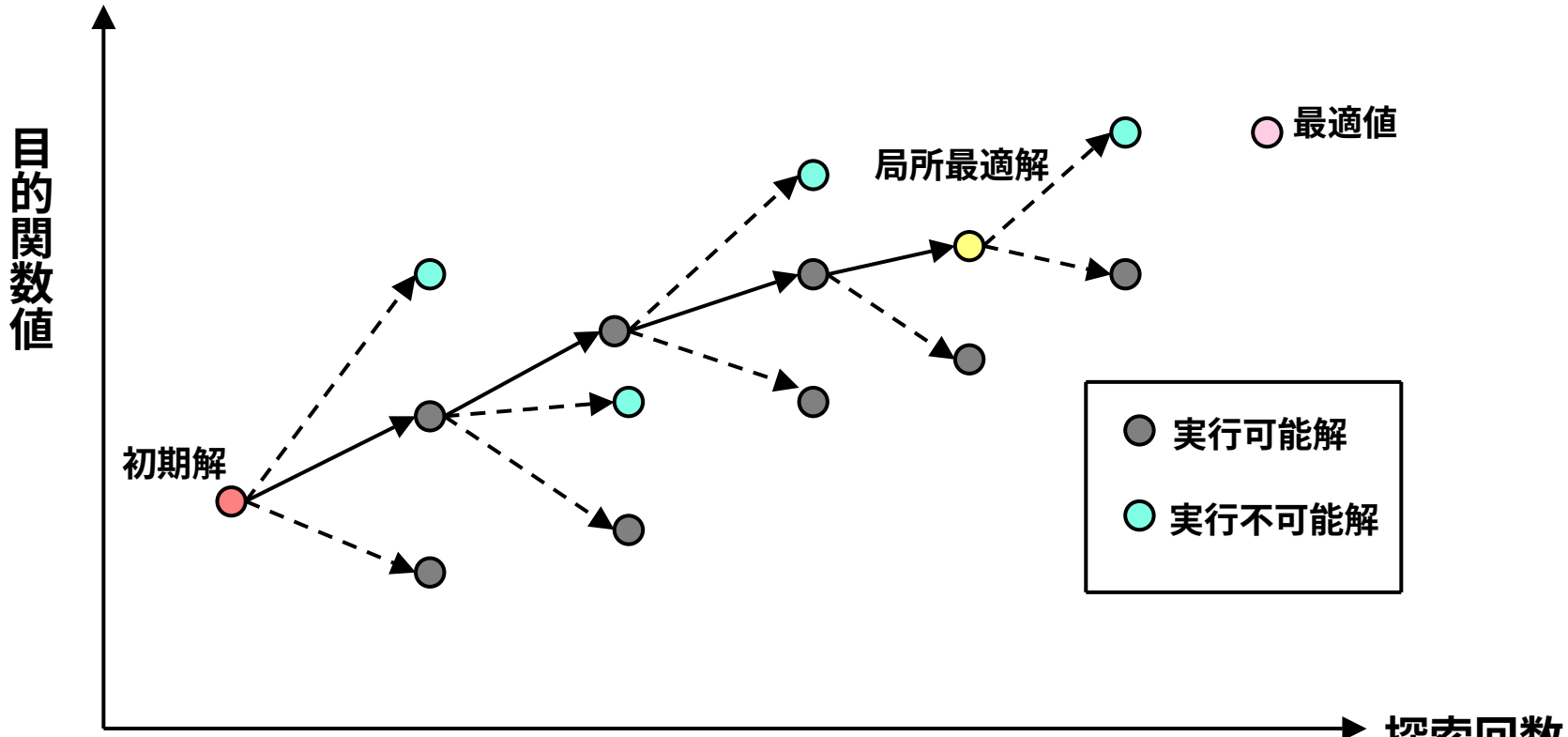


図4.2：従来の0-1空間の局所探索法

実行可能解の中で最大の目的関数値となるものに移動していく

4.3 本研究での0-1空間の局所探索法

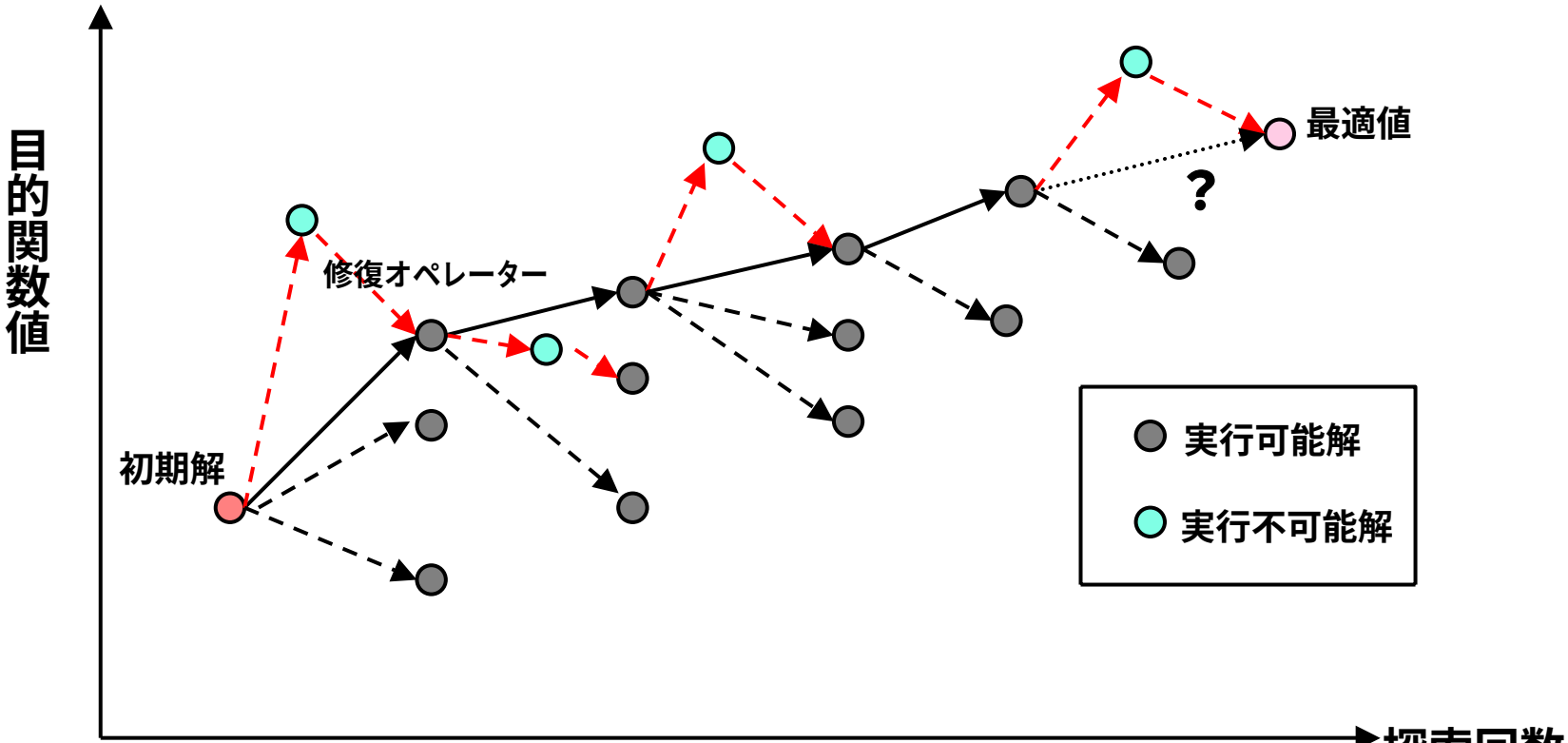


図4.3：本研究で提案する修復オペレーターを導入した0-1空間の局所探索法

実行不可能解に移動した時は，修復オペレーターを用いる

修復した実行可能解を含め局所探索を行うことによる精度の上昇

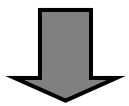
5. 代理制約式と修復オペレーター

修復オペレーター

[2]の中で使用されたもの

特徴

- 非常に単純なアルゴリズムなので計算時間をあまり要しない
- 実行不可能解を実行可能解に対応させることができる



- 何らかの基準で採択優先順序を決めておく必要がある
- 本研究では代理制約式を用いて順序を付ける

5.1 代理制約式

代理制約式

連続緩和問題の双対変数、shadow price($sp(i)$)**を用いて**
複数の制約を1制約に変換する

効率値

$$u_j = c_j / d_j$$

$$sp(1)\{a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n\} \le sp(1)\cdot b_1$$
$$\vdots$$
$$+)\,sp(m)\{a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n\} \le sp(m)\cdot b_m$$

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m} sp(i)\cdot a_{ij}\right)x_{ij} \le \sum_{i=1}^{m} sp(i)\cdot b_i$$

$$\left(d_j = \sum_{i=1}^{m} sp(i)\cdot a_{ij} \text{ とする}\right)$$

5.2　修復オペレーターのアルゴリズム

Step1　**代理制約式で求めた効率値** u_j **の大きな順に，プロジェクトを並び替える．これをプロジェクトの採択優先順序とする**

Step2　**採択優先順序の小さな方からみて、制約条件を満たすまで1→0に変更**(drop phase)

Step3　**採択優先順序の大きな方からみて、制約条件を満たす限り0→1に変更し，終了する**(add phase)

(例)

$$\text{maximize} \quad 100x_1 + 600x_2 + 1200x_3 + 2400x_4 + 500x_5 + 2000x_6 \qquad (5.1)$$

$$\text{subject to} \quad 8x_1 + 12x_2 + 13x_3 + 64x_4 + 22x_5 + 41x_6 \le 80 \qquad (5.2)$$

$$3x_1 + 6x_2 + 4x_3 + 18x_4 + 6x_5 + 4x_6 \le 20 \qquad (5.3)$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \in \{0,1\}$$

局所探索で実行不可能解 $(x_1, x_2, x_3, x_4, x_5, x_6) = (1,1,1,1,0,0)$ **を評価するときの制約の左辺は**(5.2),(5.3)**の順に** $97,31$

代理制約式で求めたプロジェクト採択優先順序 $(3,2,6,4,5,1)$

		制約の左辺 (5.2)	制約の左辺 (5.3)	解の変化	
drop phase	$x_1 : 1 \to 0$ **に変更**	89	28	(0,1,1,1,0,0)	
	$x_4 : 1 \to 0$ **に変更**	25	10	(0,1,1,0,0,0)	
add phase	$x_6 : 0 \to 1$ **に変更**	66	14	(0,1,1,0,0,1)	
	$x_1 : 0 \to 1$ **に変更**	74	17	(1,1,1,0,0,1)	← **実行可能解**

6. 有効勾配法を用いた初期解の生成

局所探索法の初期解

なるべく精度の良い初期解を採用すべき

良い解の周辺により良い解がある可能性が高いから

本研究では**有効勾配法**によって求めた解を初期解として用いる

6. 有効勾配法を用いた初期解の生成

有効勾配法

全てのプロジェクトを実行する状態からスタート

↓

有効勾配 c_j / h_j の値が小さなプロジェクトを逐次除外していく

※有効勾配とは目的関数値をなるべく減らさず、実行可能領域により近づけるための基準となるもの

↓

できるだけ目的関数値が大きくなる解を求める

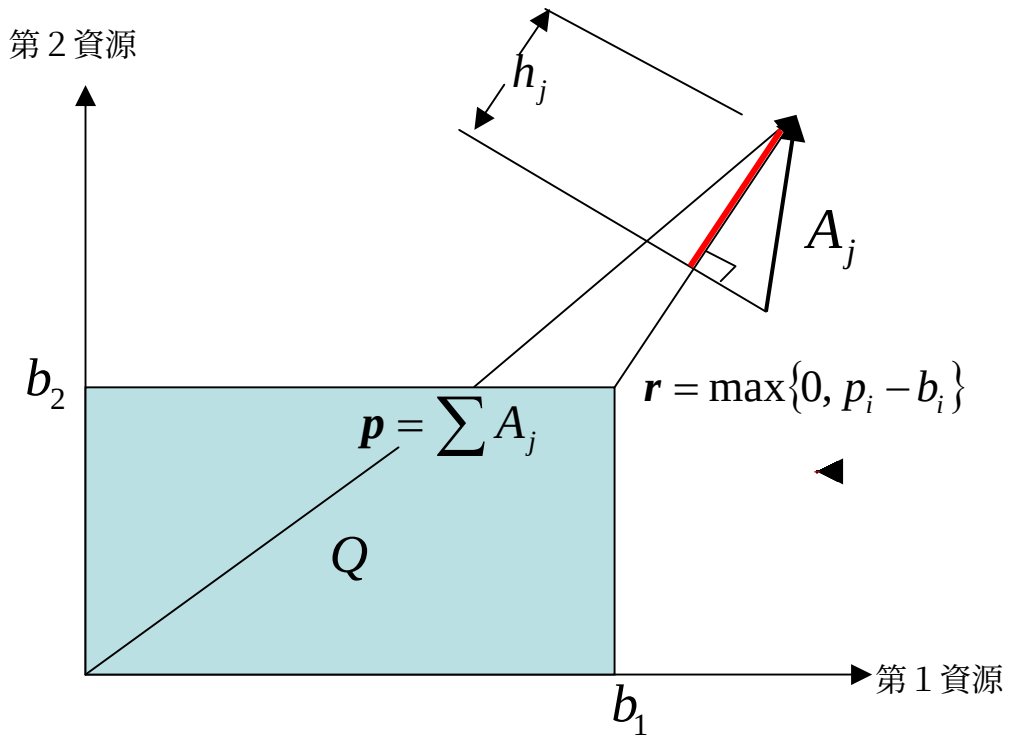


図6.1: A_jのr方向への正射影 h_j (2次元の場合)

7. 実験

7.1. 実験概要

・入力データにはOR－Libraryにある**ＭＫＰ**のデータを使用
・解の精度を最適値からの乖離率 (Gap) で判断

Gap＝((最適値－目的関数値)/最適値)×１００　(7.1)

7.2. 実験1

解法①：従来の0-1空間の局所探索法
初期解：有効勾配法の解

精度の比較をする

解法②：修復オペレーターを用いた0-1空間の局所探索法
初期解：有効勾配法の解

7.2 実験1 (結果)

表7.1：実験1結果

解法 / Problem Name	有効勾配法(初期解)		解法①		解法②		最適値
	目的関数値	Gap(%)	目的関数値	Gap(%)	目的関数値	Gap(%)	
Weing1(28変数2制約)	140477	0.5669	140477	0.5669	141278	0	141278
Weing3 (28変数2制約)	95627	0.0522	95627	0.0522	95627	0.0522	95677
Weing6 (28変数2制約)	130213	0.3139	130233	0.2986	130233	0.2986	130623
Weing7 (105変数2制約)	1095112	0.0304	1095112	0.0304	1095382	0.006	1095445
Weing8 (105変数2制約)	620060	0.6822	620060	0.6822	621086	0.5178	624319
Sento1 (60変数30制約)	7648	1.605	7648	1.605	7758	0.1801	7772
Sento2 (60変数30制約)	8697	0.2866	8697	0.2866	8722	0	8722
Petersen2 (10変数10制約)	8337	4.240	8337	4.240	8706.1	0	8706.1
Petersen5 (28変数10制約)	11810	4.758	11890	4.113	12400	0	12400
Petersen6 (39変数5制約)	10506	1.055	10570	0.4521	10570	0.4521	10618
100-30-1 (100変数30制約)	21375	2.602	21557	1.773	21835	0.5058	21946
100-30-2 (100変数30制約)	40058	1.739	40318	1.101	40458	0.7580	40767
100-30-3 (100変数30制約)	56677	1.421	56955	0.937	57494	0	57494

Gap＝((最適値－目的関数)/最適値)×１００

7.2 実験1 (考察)

・従来の0-1空間の局所探索法よりも良い精度の近似解を得ることができた

⇒ 従来の方法よりも高精度な解法

・いくつかの問題を見てみると最適解に達していない

↓ さらに解の改善を図るために

メタ戦略を組み込む・・・局所最適解からの脱出

7.3 反復局所探索法

過去の探索で得られた良い解にランダムな変形を加えたものを初期解とし、局所探索を反復する方法

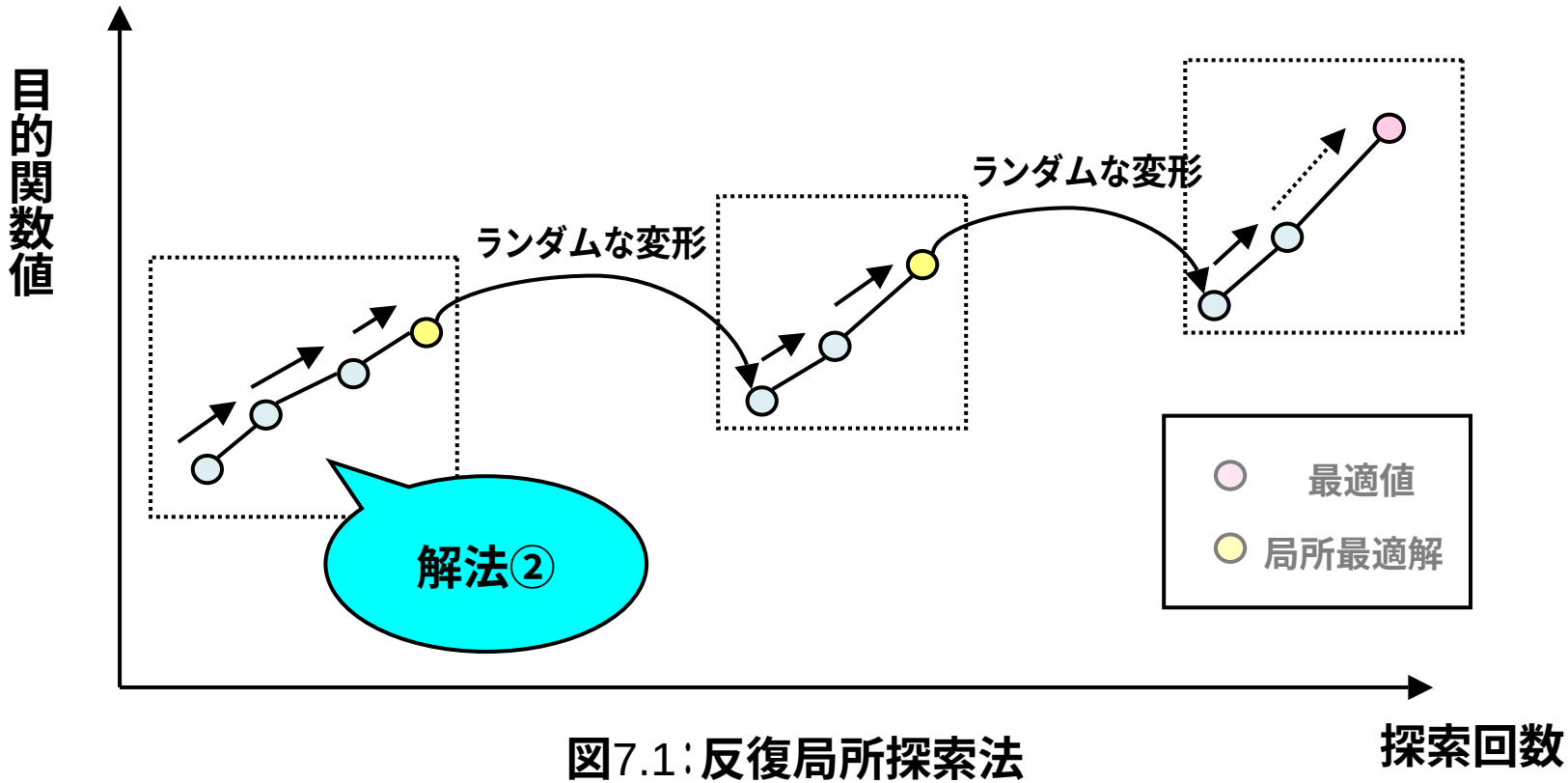
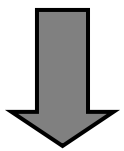


図7.1：反復局所探索法

7.4 実験2

実験1で最適解を見出せなかった問題について

解法③：反復局所探索戦略を適用した修復オペレーターを用いた0-1空間の局所探索法



最適値に達するか実験

7.4. 実験2 (結果・考察)

表7.2：実験2結果

解法 / Problem Name	解法③			最適値
	目的関数値	Gap(%)	計算時間(ミリ秒)	
Weing3 (28変数2制約)	95677	0	100	95677
Weing6 (28変数2制約)	130623	0	200	130623
Weing7 (105変数2制約)	1095445	0	181	1095445
Weing8 (105変数2制約)	624319	0	861	624319
Sento1 (60変数30制約)	7772	0	981	7772
Petersen6 (39変数5制約)	10618	0	901	10618
100-30-1 (100変数30制約)	21946	0	246214	21946
100-30-2 (100変数30制約)	40687	0.196	620032	40767

Gap＝((最適値－目的関数)/最適値)×１００

・小規模な問題に対して短時間で最適値
・100変数30制約式の難しい問題に対しても、高精度な解を発見

8. まとめ･今後の展開

⑴ 修復オペレーターを用いて0-1空間を局所探索するアルゴリズムを提案

➡ 従来の0-1空間の局所探索法より性能がよい

⑵ メタ戦略を組込んだところ精度がさらに上昇

➡ 近似解法として有用

大規模な問題に対する計算時間が今後の課題

9. 参考文献

[1]柳浦睦憲　茨木俊秀：「組合せ最適化-メタ戦略を中心として-」,朝倉書店，2001

[2] P.C.CHU AND J.E.BEASLEY：“A Genetic Algorithm for the Multidimensional Knapsack Problem”, Journal of Heuristics,4:pp.63-86,1998

[3] J.E.Beasley:OR-Library
http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html, 2006/1/29

[4] 沼田一道　小沢和俊:“解の順列表現と探索空間縮小戦略による多次元ナップサック問題の発見的解法”, システム制御情報学会論文誌, vol17, No.5, pp218-220, 2004

[5]今野浩, 鈴木久敏：「整数計画法と組合せ最適化」, 日科技連, 1999

抄録訂正

P127　**図8．1**

(誤)　$$\boldsymbol{p} = \sum \boldsymbol{a}_j$$

(正)　$$\boldsymbol{p} = \sum \boldsymbol{A}_j$$

付録

MKPの代表的な解法

- **構築法**

  何も無いところから，目的関数値に対する局所的な評価値に基づいて解を構成する

- **改善法**

  与えられた解を簡単な操作によって逐次改善する方法

- **メタ戦略**

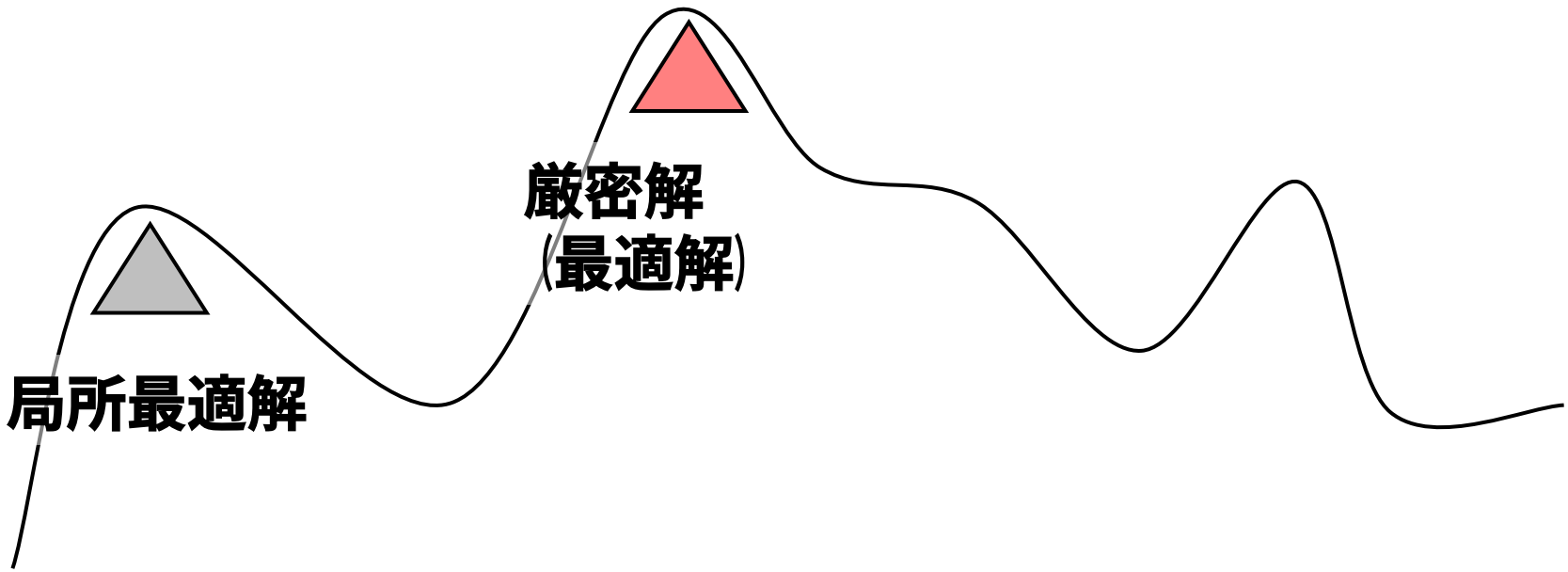
  局所探索などで得られた解の周辺の探索を丹念に行う方法

連続緩和問題

MKPの制約条件 $x_j \in \{0,1\}$ を $0 \le x_j \le 1$ に緩和したもの

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \quad j = 1,2,\ldots,n \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_{i,} \; i = 1,2,\ldots,m \\
& 0 \le x_j \le 1
\end{aligned}
$$

局所最適解



有効勾配法-記号の定義-

用いる記号

$N = \{1,2,...,n\}$: プロジェクト j を全て実行した場合の 集合

$N_1 = \{j \mid x_j = 1\}$: $x_j = 1$ となるプロジェクトの 集合

$\boldsymbol{A}_j = (a_{1j}, a_{2j}, ..., a_{mj})^t$: 第 j プロジェクトの資源ベ クトル

$\boldsymbol{b} = (b_1, b_2, ..., b_m)^t$: 利用可能量ベクトル

$\boldsymbol{p} = (\sum_{j \in N_1} a_{1j}, ... \sum_{j \in N_1} a_{mj})^t$: 要求量ベクトル

有効勾配法-アルゴリズム-

Step1 $N_1 := N$, $p := \sum_{j \in N_1} A_j$ とし、リストLを空とする

Step2 $\mathbf{p} \leq \mathbf{b}$ となるまで、以下を繰り返す

(1) ベクトル r の各成分 r_i を $r_i = \max\{0, p_i - b_i\}$ とする

(2) 全ての $j \in N_1$ につき、 $h_j := (A_j^T * r) / \|r\|$, $g_j := c_j / h_j$

(3) $g_k := \min g_j$ となる $k \in N_1$ を求め、 $N_1 := N_1 \setminus \{k\}$,

$p := p - a_k$ とし、リスト L の最後尾に k を加える

Step3 リストLの後ろから順にみて、 その要素jに対して $p + A_j \leq b$ であれば、 $p = p + A_j$, $N_1 := N_1 \cup \{j\}$ とする操作を繰り返す