メタヒューリスティクスアルゴリズムの性質を用いた実務的な 0-1 整数計画問題の高速な解法

(株)数理システム
佐藤　誠
田辺　隆人

概要

- NUOPT のメタヒューリスティクスアルゴリズム WCSP （タブーサーチ）
- min max 関数の実装
- 実務問題への応用
- 今後の発展

WCSP（重み付き制約充足問題）

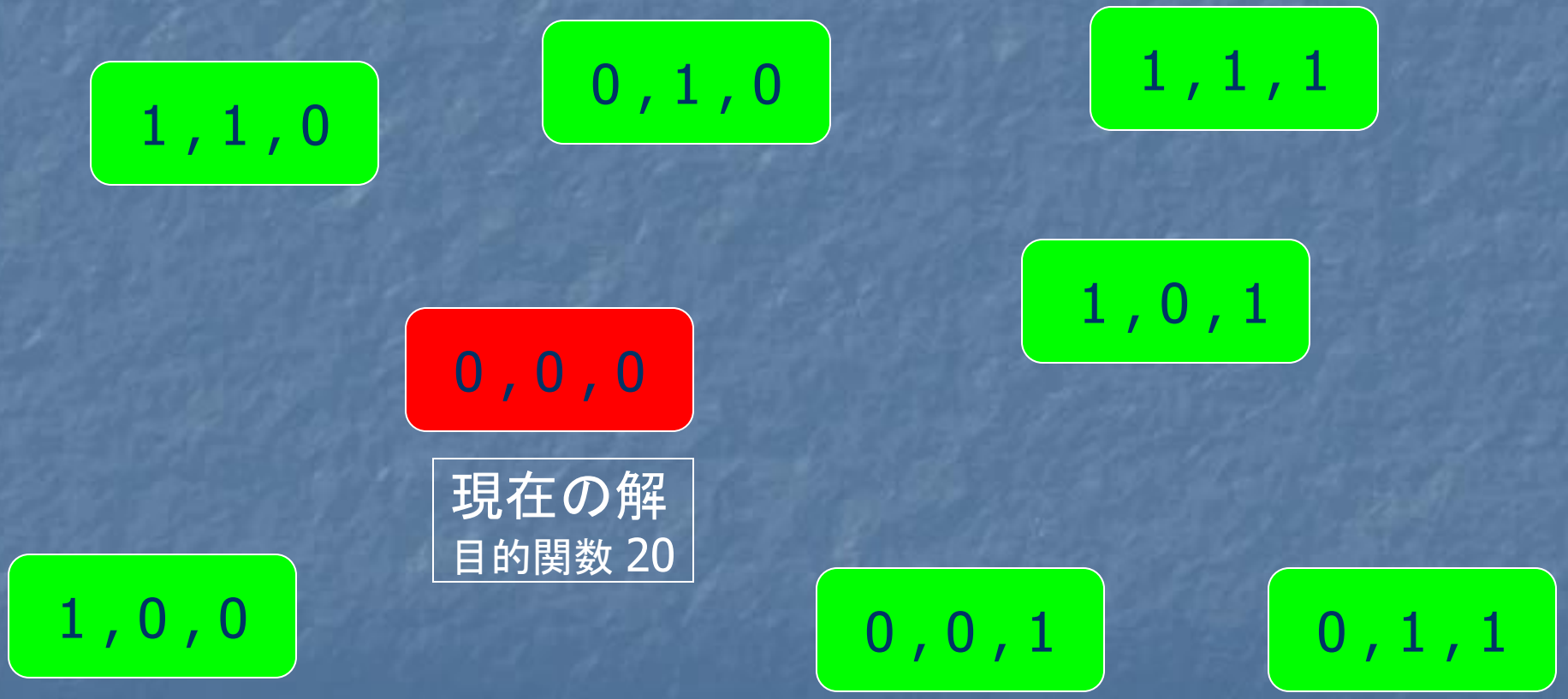
- タブーサーチ
- 0-1 整数変数のみで記述された問題が対象
- 近傍は，現在の解ベクトルからどれか1つの成分を変化させたもの

  → 近傍の値を取得する部分がプログラム全体の 8 ～9 割を占める

  → 線形・非線形問わず解くことができる

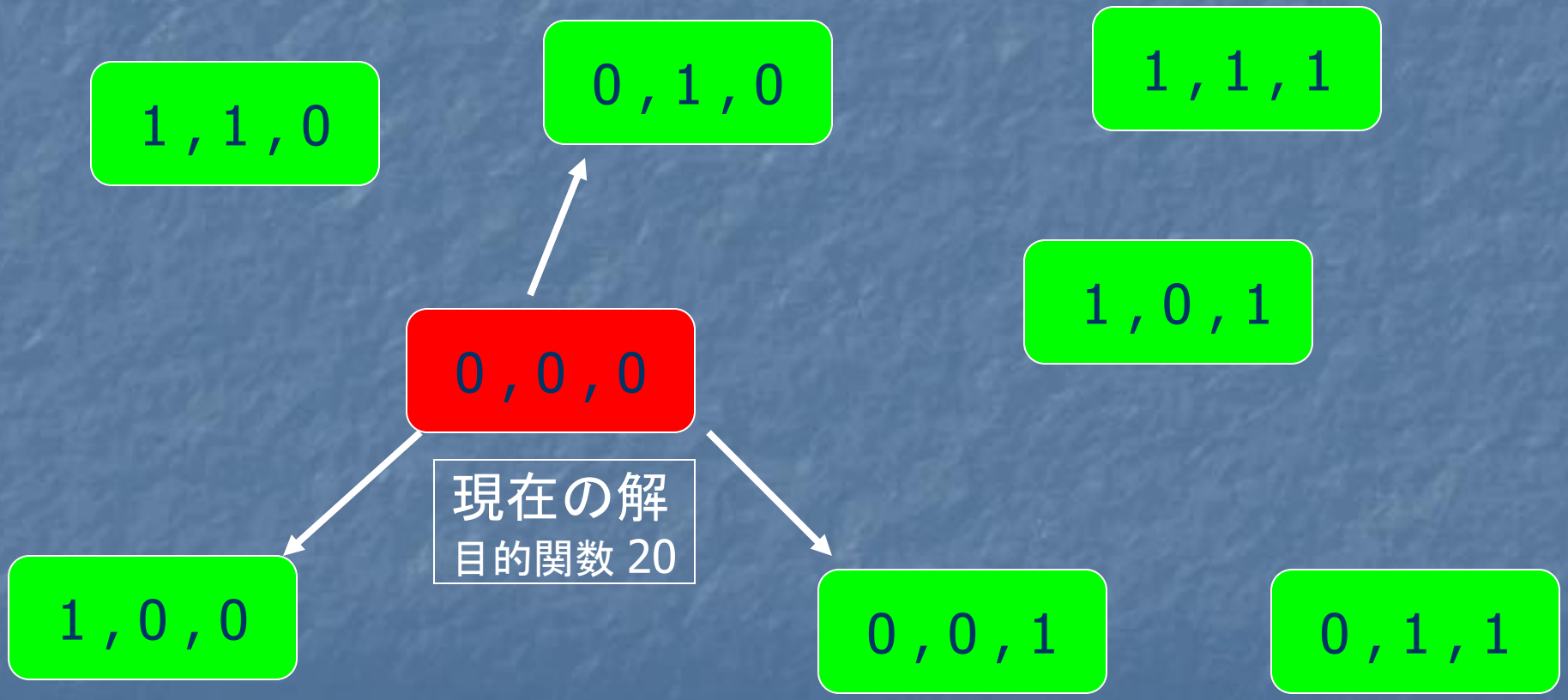
近傍探索のイメージ

- 3 変数の最大化問題の場合



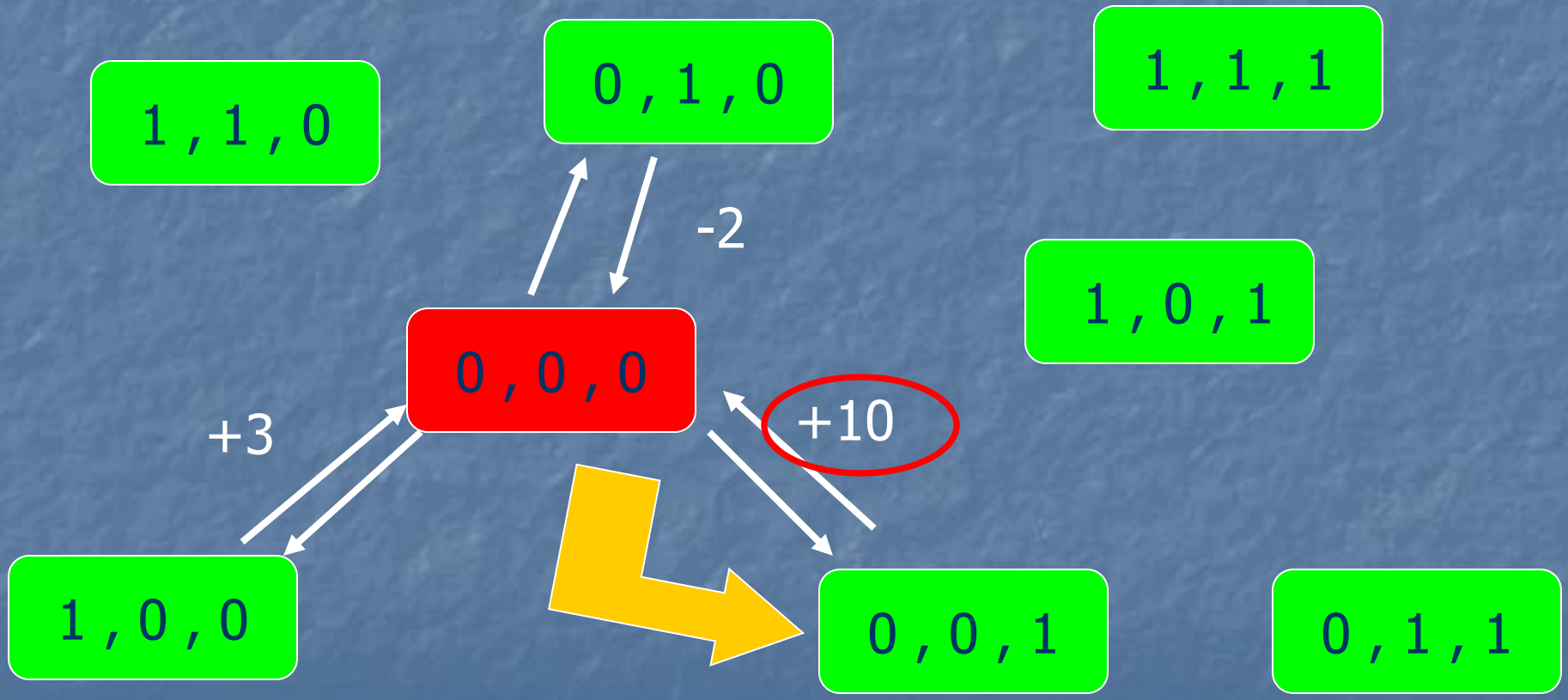
近傍探索のイメージ

- 3 変数の最大化問題の場合



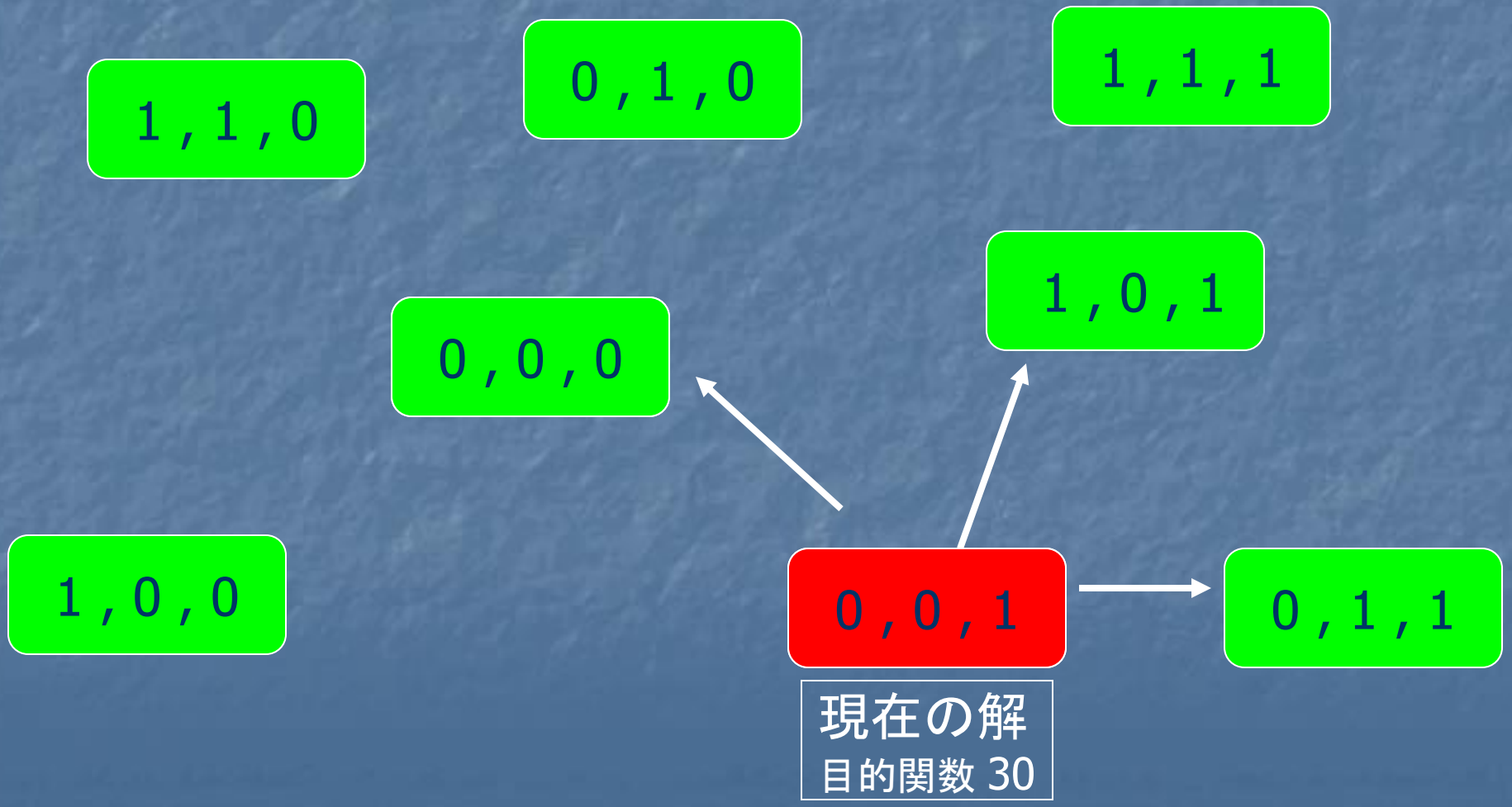
近傍探索のイメージ

- 3 変数の最大化問題の場合



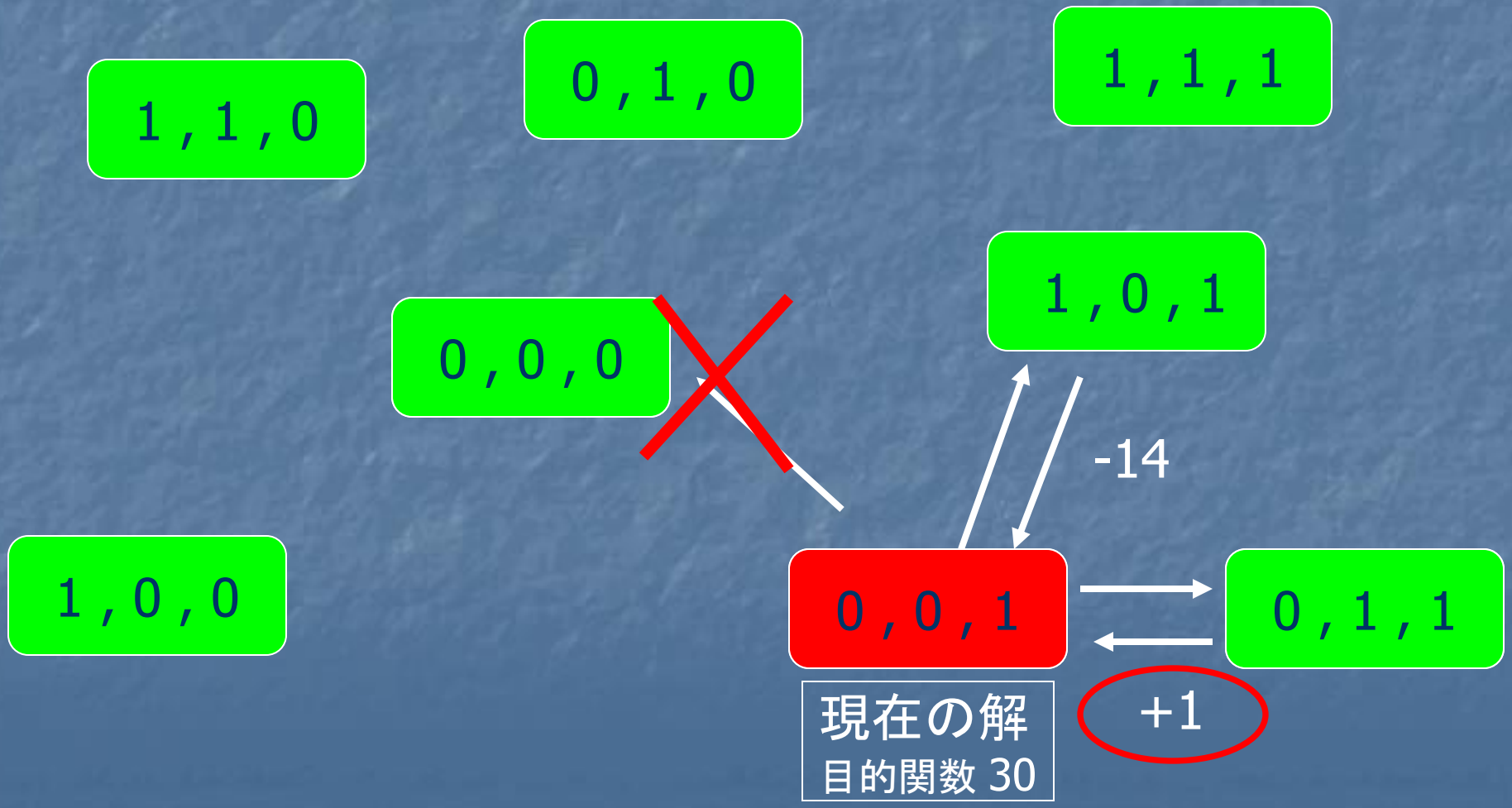
近傍探索のイメージ

- 3 変数の最大化問題の場合



近傍探索のイメージ

- 3 変数の最大化問題の場合



近傍探索のイメージ

- 3 変数の最大化問題の場合

近傍探索のイメージ

- 3 変数の最大化問題の場合

近傍を探索するスピードが肝

1

現在の解
目的関数 31

min max 関数

- 以下の min max 関数を実装

$$g_i = f_i(x)$$

$$\min_i(g_i) \quad \max_i(g_i)$$

min max 関数

- 特徴
  - ○ g が x の線形式の場合に高速化
  - ○ x は必ず 1 成分のみしか一度に変化しないという性質を用いて高速化
  - ○ x に対する係数が 0 のケースが多いという性質を用いて高速化

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

高速化の詳細

今一番大きい関数を g(0) とする

$$g(0) = 2x_1 - 3x_2 + 0x_3 + 0x_4 + \dots$$
$$g(1) = 1x_1 + 0x_2 + 3x_3 + 0x_4 + \dots$$
$$g(2) = 0x_1 - 2x_2 + 0x_3 - 1x_4 + \dots$$
$$g(3) = -5x_1 + 3x_2 - 2x_3 + 3x_4 + \dots$$

x1 が 0 → 1 になった場合 どの g が max になるか？

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

x1 が 0 → 1 になった場合 どの g が max になるか？

→ g(0) は “2” 増加するので, x1 の係数が 0 以下の g については検討する必要がない！

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

x2 が 1 → 0 になった場合 どの g が max になるか？

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

x2 が 1 → 0 になった場合 どの g が max になるか？

→ g(0) は “3” 増加するので, x2 の係数が 0 以下の g については検討する必要がない！

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

x3 が 0 → 1 になった場合 どの g が max になるか？

高速化の詳細

今一番大きい関数を g(0) とする

g(0) = 2 x1 - 3 x2 + 0 x3 + 0 x4 + …

g(1) = 1 x1 + 0 x2 + 3 x3 + 0 x4 + …

g(2) = 0 x1 - 2 x2 + 0 x3 - 1 x4 + …

g(3) = -5 x1 + 3 x2 - 2 x3 + 3 x4 + …

x3 が 0 → 1 になった場合 どの g が max になるか？

→ g(0) は変化しないので, x3 の係数が 0 以下の g については検討する必要がない！

実務問題への応用

- min min 問題，max max 問題 が以前より高速に解けるようになった

$$g_{ij} = x_i d_{ij}$$

$$(\text{minimize}) \sum_i \min_j \left(g_{ij}, g_{ij} \neq 0 \right)$$

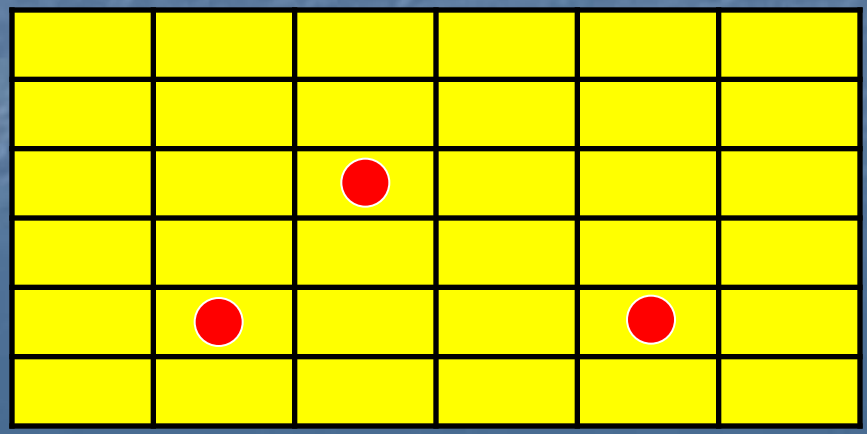
minmin 問題応用

- 施設配置問題
  - ○ ｎ × ｎ のマス上に　Ｎ 個の施設を建設する
  - ○ 各マス ｉ は最寄の施設に収容される
  - ○ 各マス ｉ から施設までの距離の総和を最小化したい

0-1 整数変数

$$x_i$$

マス ｉ に施設を建てるならば 1
そうでないならば 0



minmin 問題応用

- 従来の定式化＜中間変数の導入＞

  0-1整数変数 y を導入する

y_{ij} { マス j がマス i の施設に収容されるならば 1
そうでないならば 0

$x_i \geq y_{ij}$ 施設がないマスには収容されることができない

$\sum_i y_{ij} = 1$ 各マスは必ず1つの施設に収容される

minmin 問題応用

- 従来の定式化＜最寄の表現＞

あるマスから見て，全ての施設の中で一番近いところ以外の施設に収容されることはできない．

$$y_{ij} dist_{ij} \leq dist_{i'j} + \mathrm{M}\left(1 - x_{i'}\right)$$

目的関数は以下のように表現される

$$(\text{minimize}) \sum_{ij} y_{ij} dist_{ij}$$

M：十分大きい数

問題が単純であればこの制約はいらないが，他の目的関数・制約等の兼ね合いで“総距離”よりも優先したい事柄がある場合には，この制約がないと最寄に収容されない恐れがある.

あるマスが[illegible]この施設の中で一番近いところ以外の施設に収容さ[illegible]にはできない.

$$y_{ij} dist_{ij} \leq dist_{i'j} + \mathrm{M}\left(1 - x_{i'}\right)$$

目的関数は以下のように表現される

$$(\mathrm{minimize}) \sum_{ij} y_{ij} dist_{ij}$$

M:十分大きい数

minmin 問題応用

- 従来の定式化＜特徴＞
  - ○ 0-1 整数変数のみを用いて線形で定式化
  - ○ 中間変数を置くことによって変数の数は マスの数の二乗に比例する
  - ○ 最寄を表現するために，制約式を設けたが，この数はマスの数の三乗に比例する

マスの数の増加に伴い，問題規模は急激に爆発する．

例題では 10 × 10 の際にメモリーオーバでパンク

minmin 問題応用

- min 関数を用いた定式化

$$(\text{minimize}) \sum_i \min_j \left(dist_{ij} + \mathrm{M}\left(1 - x_i\right) \right)$$

- 特徴
  - ○ 中間変数を用いる必要がないため，マスの数の増加に対して線形に問題規模は増加する

厳密解法と比較すると，8 × 8 の規模の問題では 同一の解が 10 倍程度の速さで出すことができた．

厳密解法では 10 × 10 の規模がメモリオーバで解けなかったが，
20 × 20 の規模の問題が 100 秒程の探索で解の更新が落ち着いた．

今後の発展

- min max 以外にも関数を追加することによって様々な問題に適用することができる.

実用例:

Count 関数

Count 関数

- 0-1 整数変数を複数個持つ シナリオ が複数存在する

S1: x1 x2 x5 x8
S2: x2 x7 x10
S3: x3
S4: x6 x8 x9
S5: x1 x4 x8

- シナリオの中で，1 になっている変数の数が特定の範囲内にあるシナリオの数に対して，制約をかけたい．

Count 関数

- 従来の方法では，各シナリオ毎に中間変数 0-1 整数変数を用意して定式化を行った.

→ 実務では 100 万シナリオを超える問題を解きたい

→ 従来の方法では 1 万シナリオが限界

この問題に特化した Count 関数を実装したことにより
- ○ 100 万シナリオの問題も数分で解けるようになった
- ○ 従来まで取り扱っていた 1 万シナリオのケースでも，数 100 倍の速度で解けるようになった

まとめ

- メタヒューリスティクスは微係数を必要としないため，線形・非線形にこだわる必要はない．
- 中間変数を増やすよりも，直接値を参照できる関数を用意することによって高速化できる．
- 今後 min max 関数以外にも汎用的な関数を実装し 0-1 整数計画問題を幅広くカバーしたい．