九州大学 工学部地球環境工学科
船舶海洋システム工学コース

システム設計工学 （担当：木村）

(7) 組み合わせ最適化

計算量とは
組合せ最適化問題（と解法）

場所： 船1講義室

授業の資料等は

http://sysplan.nams.kyushu-u.ac.jp/gen/index.html

前回までの最適化：
最適化すべき変数が連続値

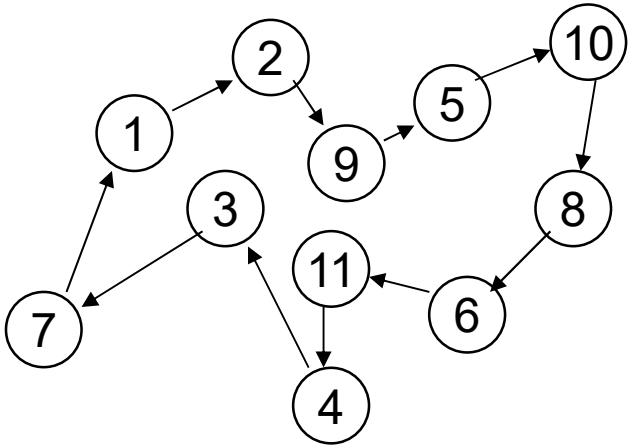
・線形計画問題：線形計画法
・非線形最適化問題：勾配法・シンプレックス法・SA・GA等

今回の講義の内容：
最適化すべき変数が離散値の場合

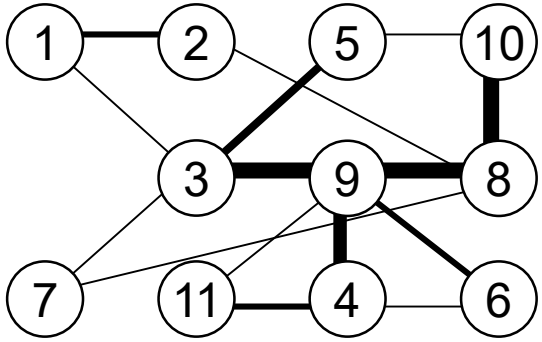
・整数計画問題／組合せ最適化問題

【組合せ最適化問題】

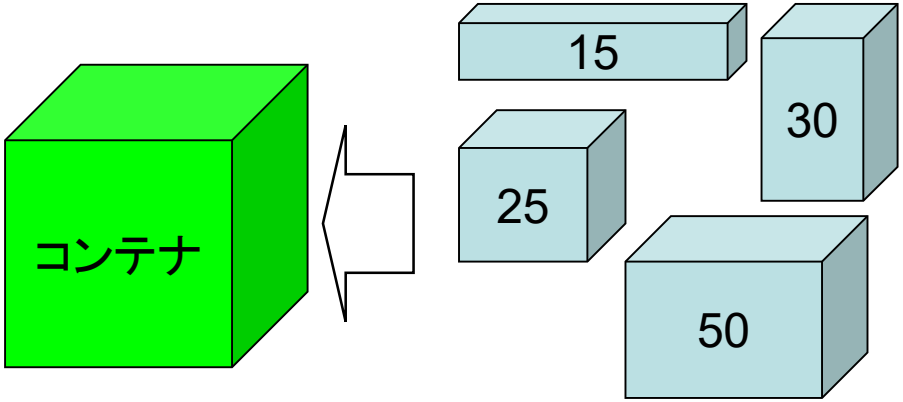
例1： 巡回セールスマン問題（TSP）
全都市を最小コストで巡回する経路は？



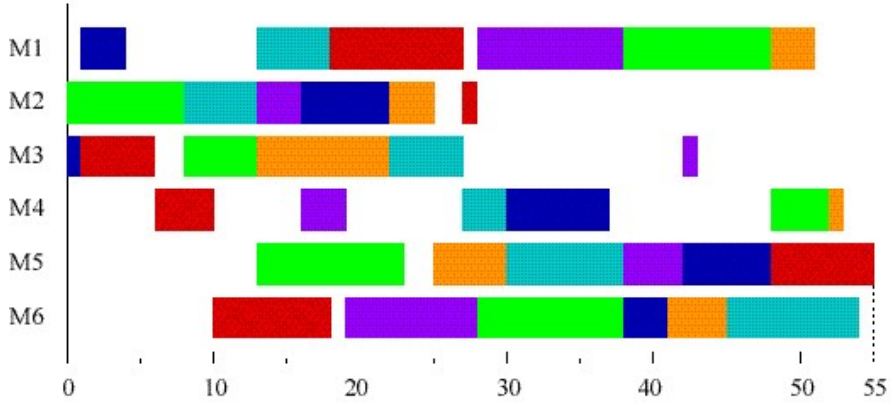
例2： 2次割当配置問題（QAP）
要素間の物流コストとグラフ構造はgiven:
コストが最小になる要素配置は？



例3： ナップサック問題（ビン詰め問題）
容量制限を越えない範囲で、
価値が最大になるようコンテナへ積込む



例4： ジョブショップスケジューリング問題
複数の機械による作業が必要なタスク
どういう順番で実行すると作業時間が
短縮できるか？



【組合せ最適化問題の難しさ】

問題を解く手数 ・計算時間 ・記憶容量

計算量の大きさの表現： 問題の難しさ・複雑さは、それを解くのに要する**計算量**で表す

問題の大きさ n（正確には、問題の記述に要するビット数）のとき、難しさを2段階に大別

1）**多項式時間問題**： 計算論的には「易しい」問題
　　計算量が $O(n)$, $O(n\log(n))$, $O(n^2)$ など一般に $O(n^{\alpha})$ である問題
　　αは定数
2）それ以外
　　計算量が $O(n!)$, $O(2^n)$ など指数的に増加する問題

問題解決法（アルゴリズム）に関して、以下の概念を導入

1）決定性（deterministic）：アルゴリズムの流れが確定していること
2）非決定性（non-deterministic）：アルゴリズムが分岐点にいるとき、常に正しい道を選ぶ能力があるものとすること。（これは架空の能力）

以上の概念を用いて、問題を3種類に分類：

多項式時間問題

1）P： 決定性アルゴリズムで多項式時間内で解決できる全ての問題。
2）NP： 非決定性アルゴリズムで、多項式時間内に解決できる全ての問題。
3）NP困難：NP以上の難しさをもつ問題

【組合せ最適化問題の難しさ】

問題を解く手数 ・計算時間 ・記憶容量

計算量の大きさの表現： 問題の難しさ・複雑さは、それを解くのに要する**計算量**で表す

問題の大きさ n（正確には、問題の記述に要するビット数）のとき、難しさを2段階に大別

1）**多項式時間問題**： 計算論的には「易しい」問題
　　計算量が $O(n)$, $O(n\log(n))$, $O(n^2)$ など一般に $O(n^{\alpha})$ である問題
　　αは定数
2）それ以外
　　計算量が $O(n!)$, $O(2^n)$ など指数的に増加する問題

問題解決法（アルゴリズム）に関して、以下の概念を導入

1）決定性（deterministic）：アルゴリズムの流れが確定していること
2）非決定性（non-deterministic）：アルゴリズムが分岐点にいるとき、常に正しい道を選ぶ能力があるものとすること。（これは架空の能力）

以上の概念を用いて、問題を3種類に分類：

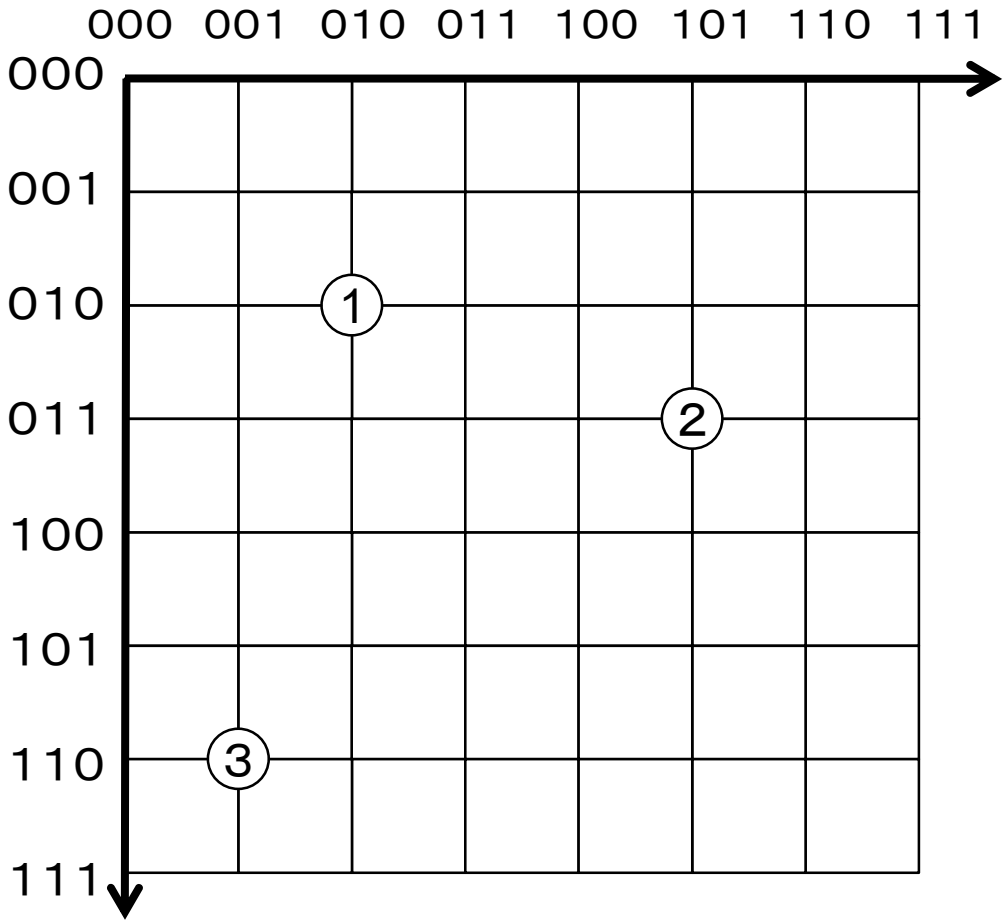
多項式時間問題

1）P： 決定性アルゴリズムで多項式時間内で解決できる全ての問題。
2）NP： 非決定性アルゴリズムで、多項式時間内に解決できる全ての問題。
3）NP困難：NP以上の難しさをもつ問題

【補足説明】 問題の記述に要するビット長（＝記述長）の計算例

例） 巡回セールスマン問題の場合： 都市間の距離をコストとすると、
問題の記述に必要なのは都市配置のみ

3都市が ８×８ の格子上のどこかに配置されている場合、



記述長＝18 bit

010 010　101 011　001 110

① ② ③

【補足説明】 big-O記法の定義

f と g を自然数の集合 N から正の実数の集合 R へ写像する単調増加関数とする。

すべての整数 $n \le n_0$ に対して、

$$f(n) \le c\, g(n)$$

であるような正の整数 c と n_0 が存在するならば、$f(n) = O(g(n))$ という。

このとき、定数係数を無視していることを強調するために、

$f(n)$ は $g(n)$ の**上界（upper bound）**であるといい、より厳密には

$f(n)$ は $g(n)$ の**漸近的上界（asymptotic upper bound）**であるという。

例1） $2n^3 + 5n^2 + 22n + 6 = O(n^3)$ ⟶ **多項式境界**

例2） $3n\log_2 n + 5n\log_2(\log_2 n) + 2 = O(n\log n)$

定数倍変わるだけなので底は省略

例3） $2^n + 10n^2 + 6 = O(2^n)$ ⟶ **指数境界**

【組合せ最適化問題の難しさ(続き1)】

●Pのオーダーで解ける:やさしい問題
多項式時間で解けるアルゴリズムが存在する問題群

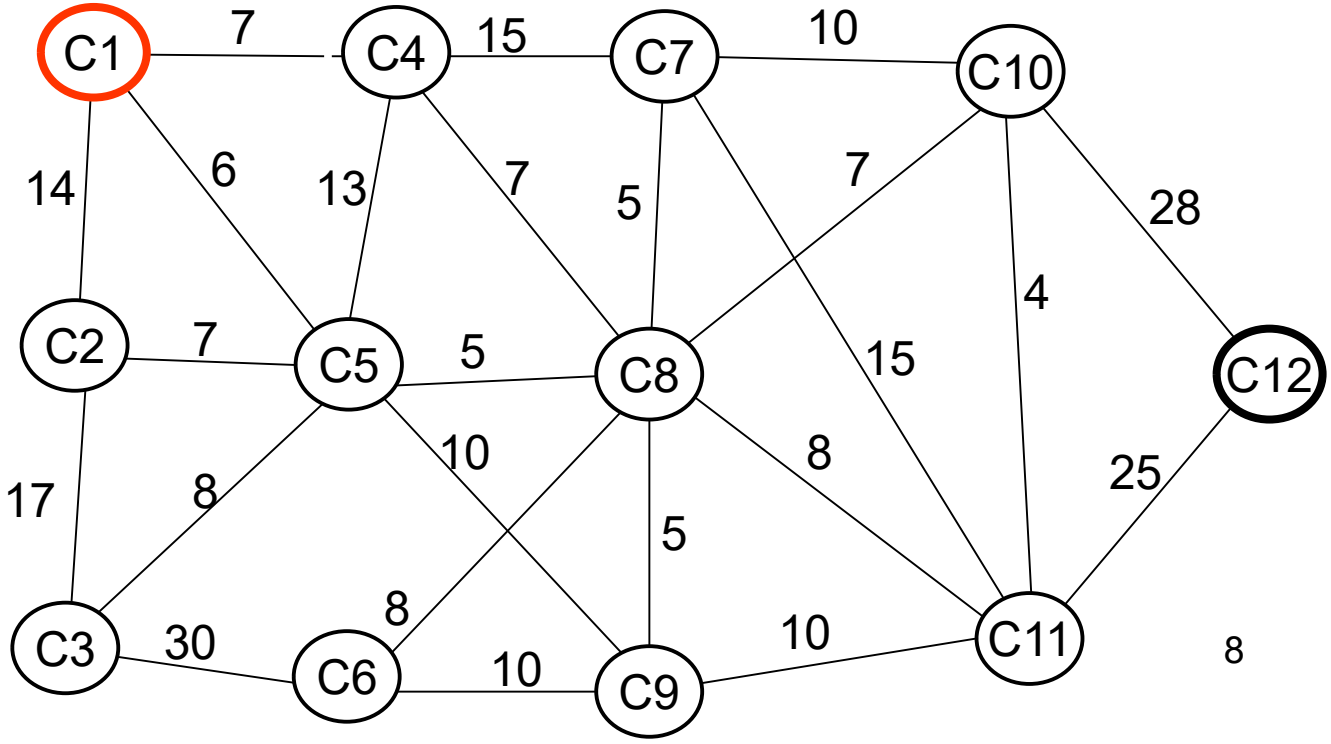
例1: 重み付きグラフにおける2点間の最短路探索問題 → ダイクストラの方法 $O(n^2)$

例2: 重み付きグラフにおける最大流問題 → フロー増加法 $O(n^2)$

例3: ソート・サーチ $O(n\log n)$

最大流問題:

各節点(C1～C12)間の
最大流量が与えられた
もとで、C1からC12への
最大流量とそのときの
各枝の流量を求める問題

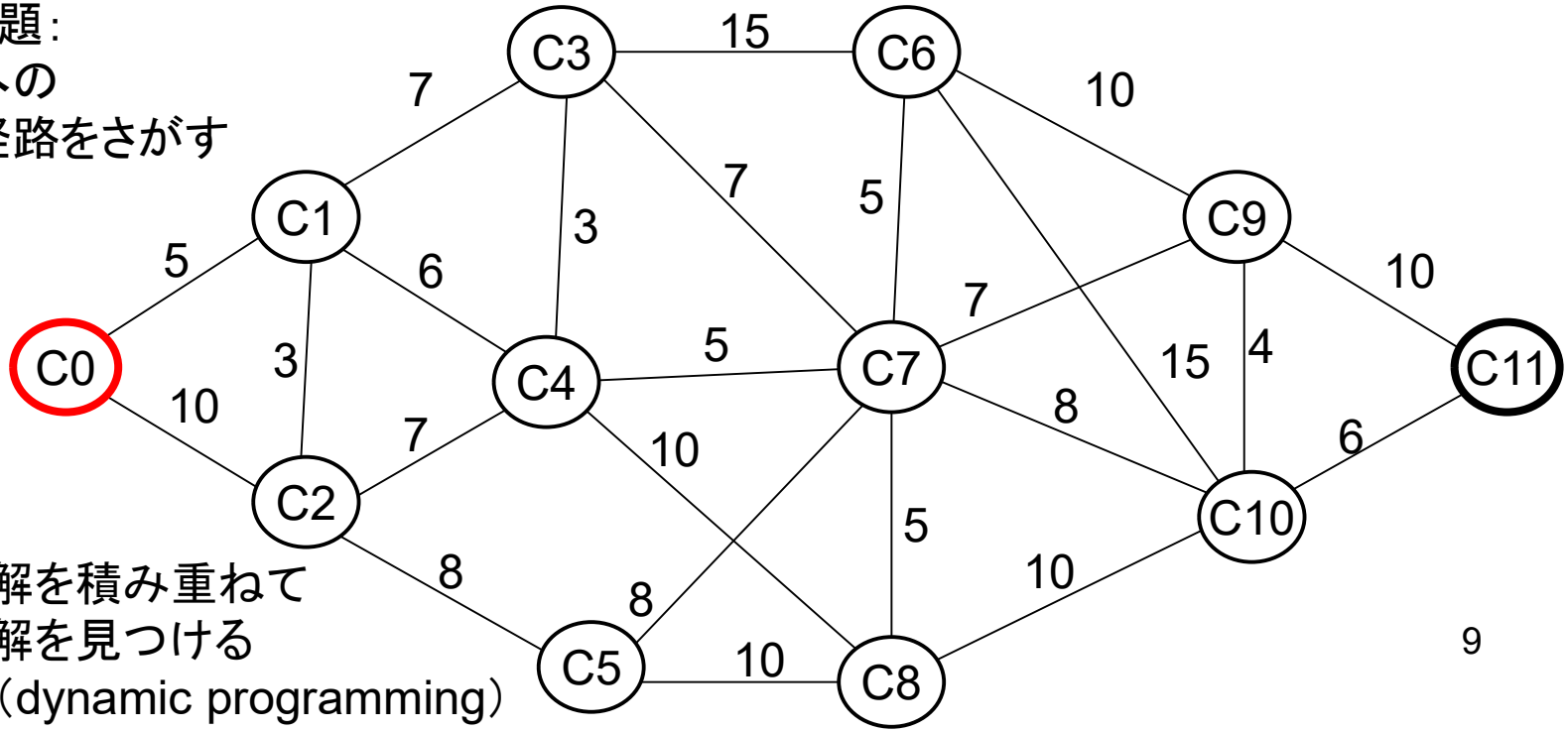


【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく

最短路探索問題：
C1からC12への
最小コストの経路をさがす



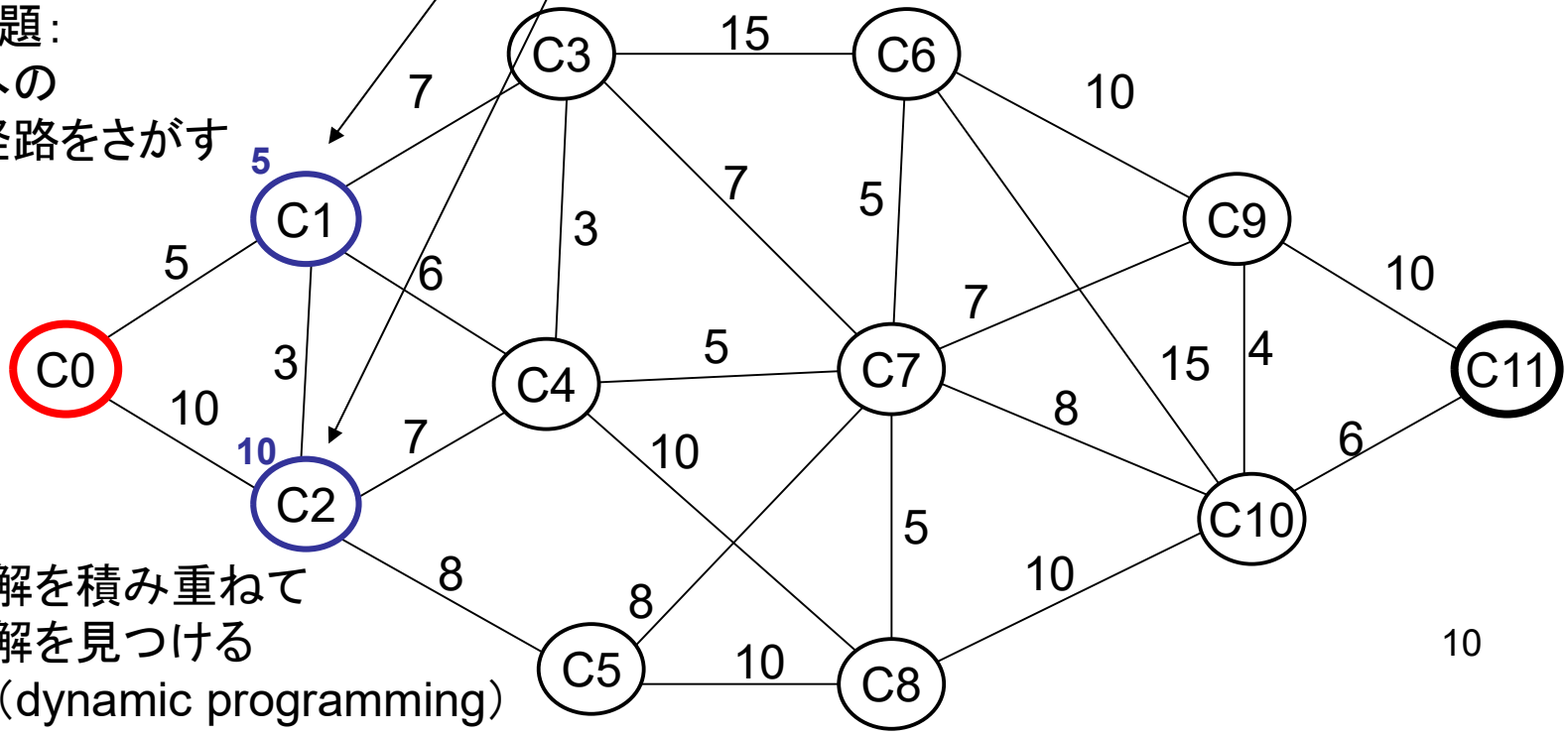
部分的な最適解を積み重ねて
全体的な最適解を見つける
→動的計画法（dynamic programming）

【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく

最短路探索問題：
C1からC12への
最小コストの経路をさがす

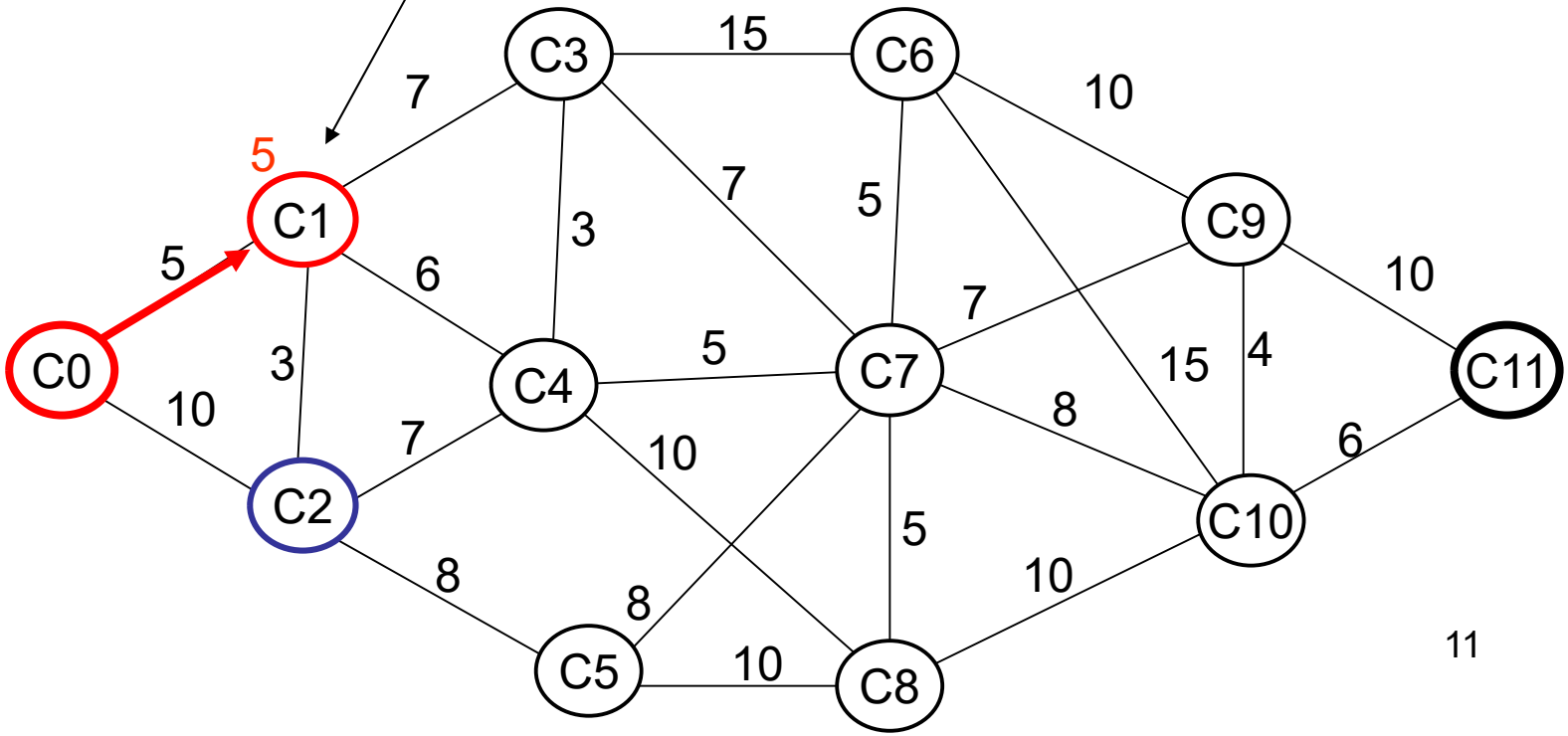


部分的な最適解を積み重ねて
全体的な最適解を見つける
→動的計画法（dynamic programming）

【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

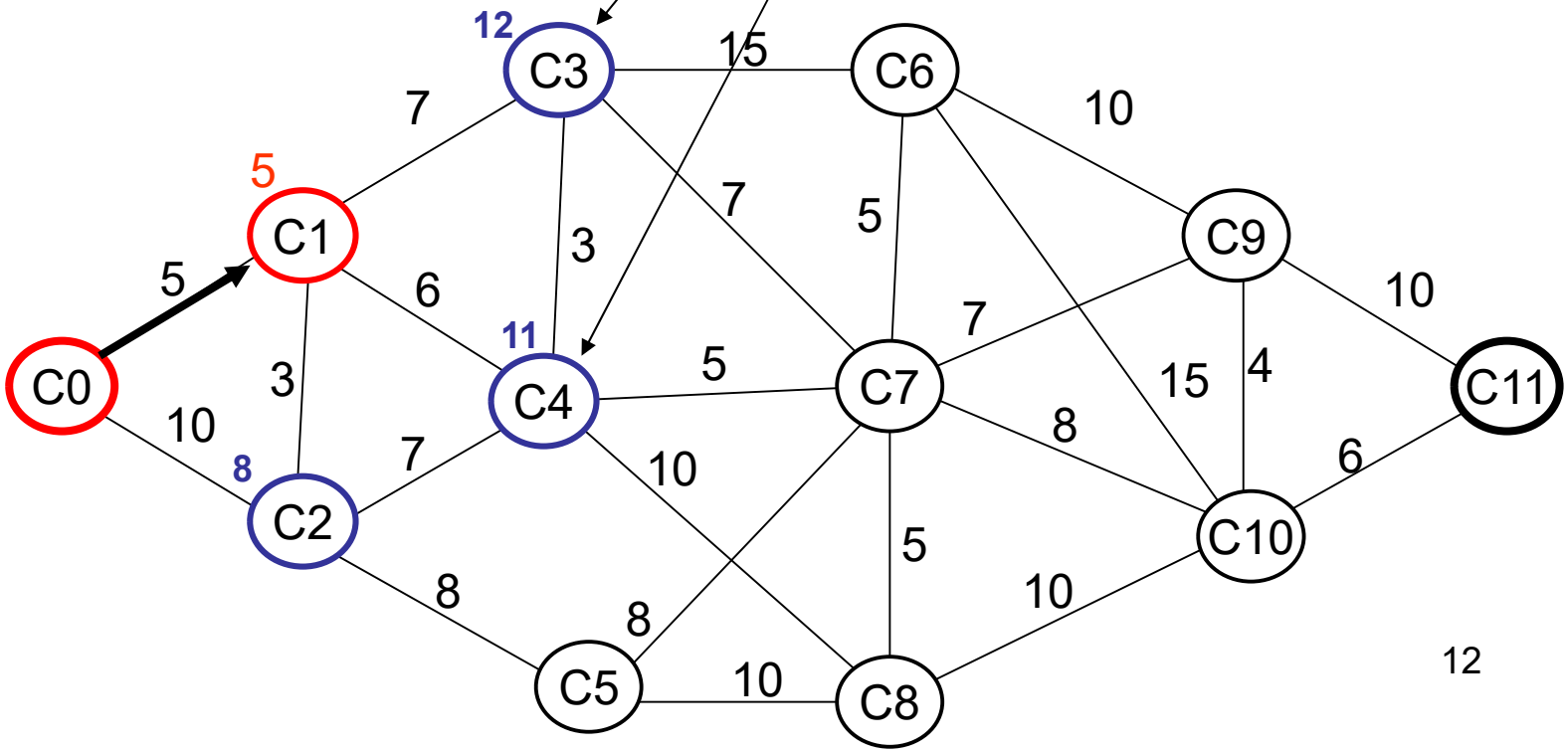
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

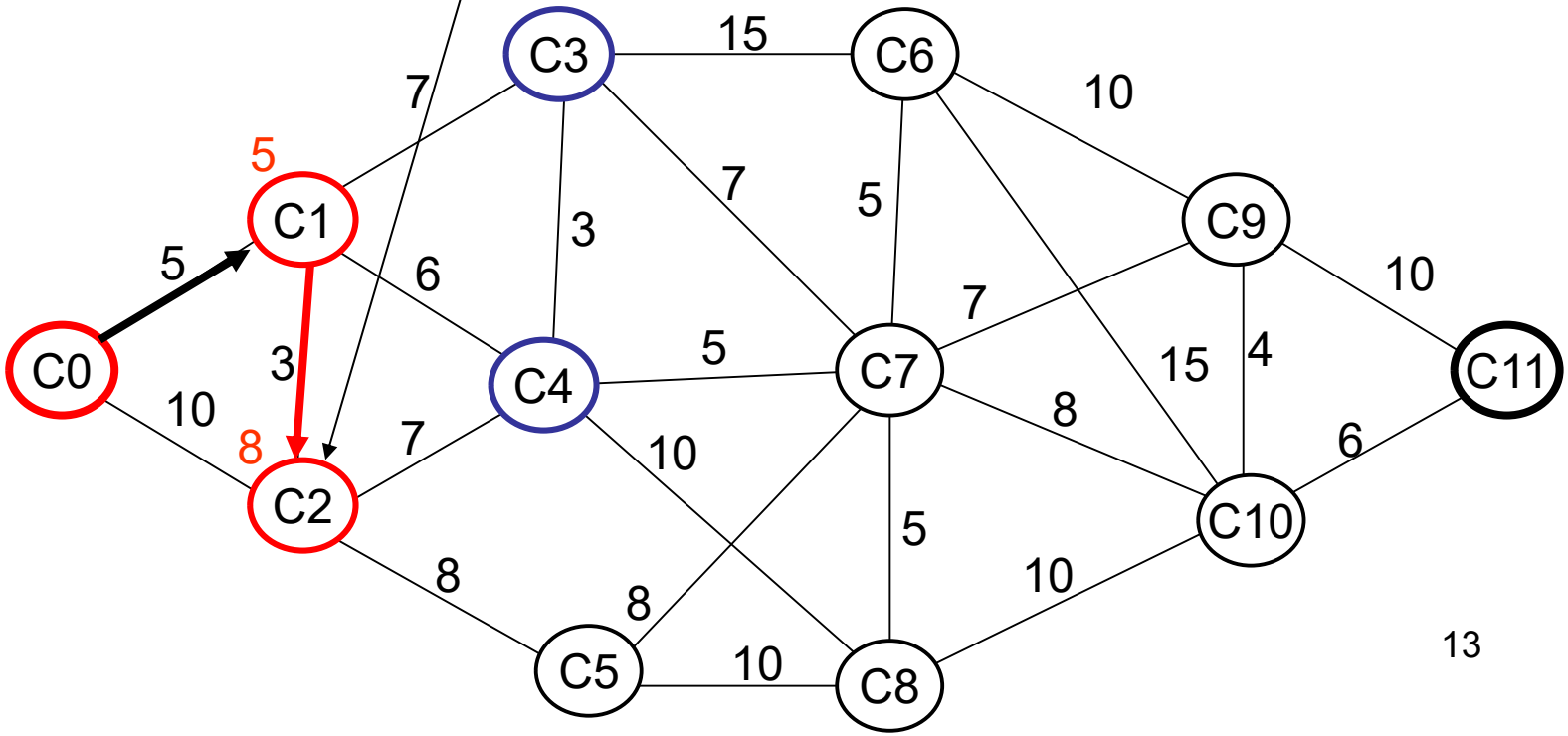
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

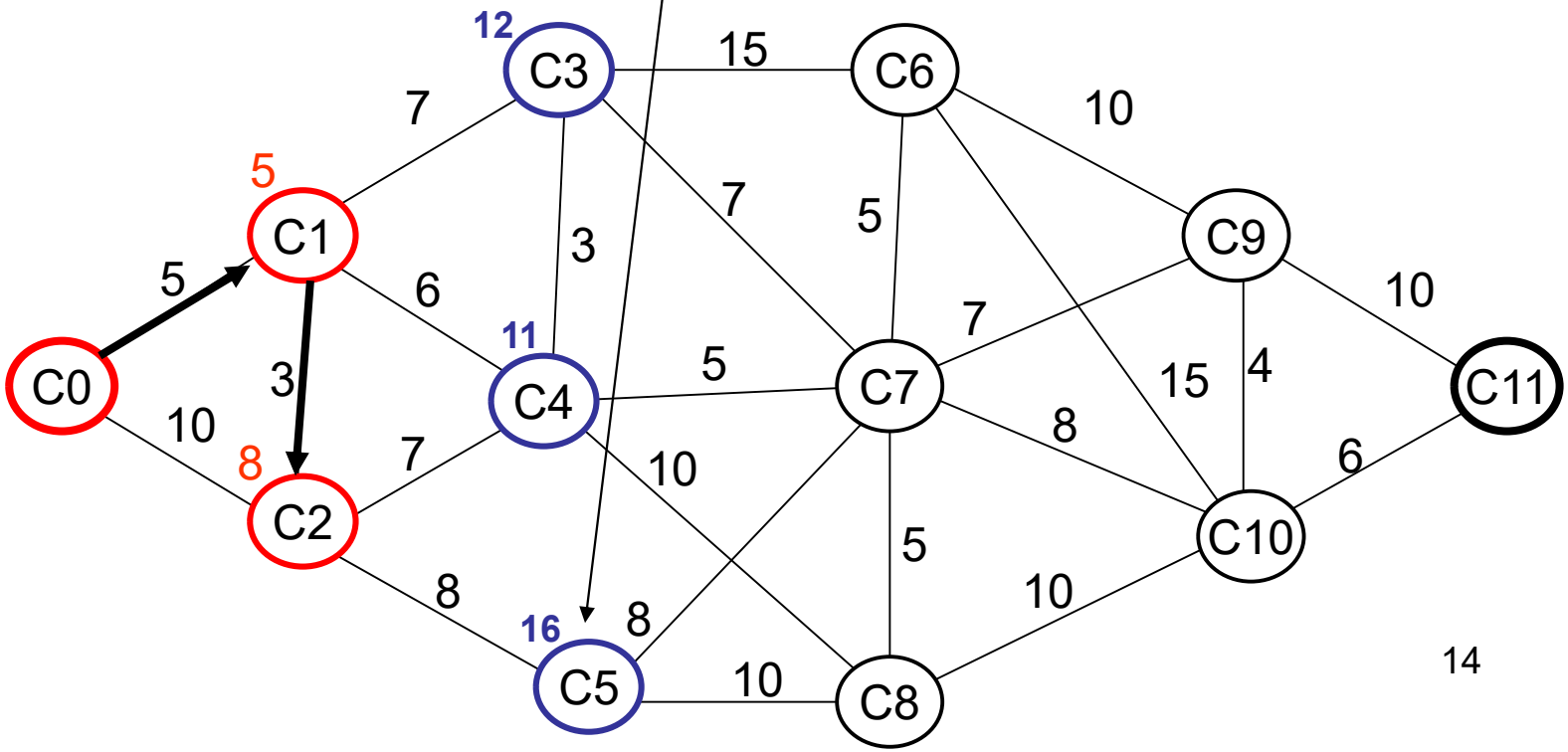
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

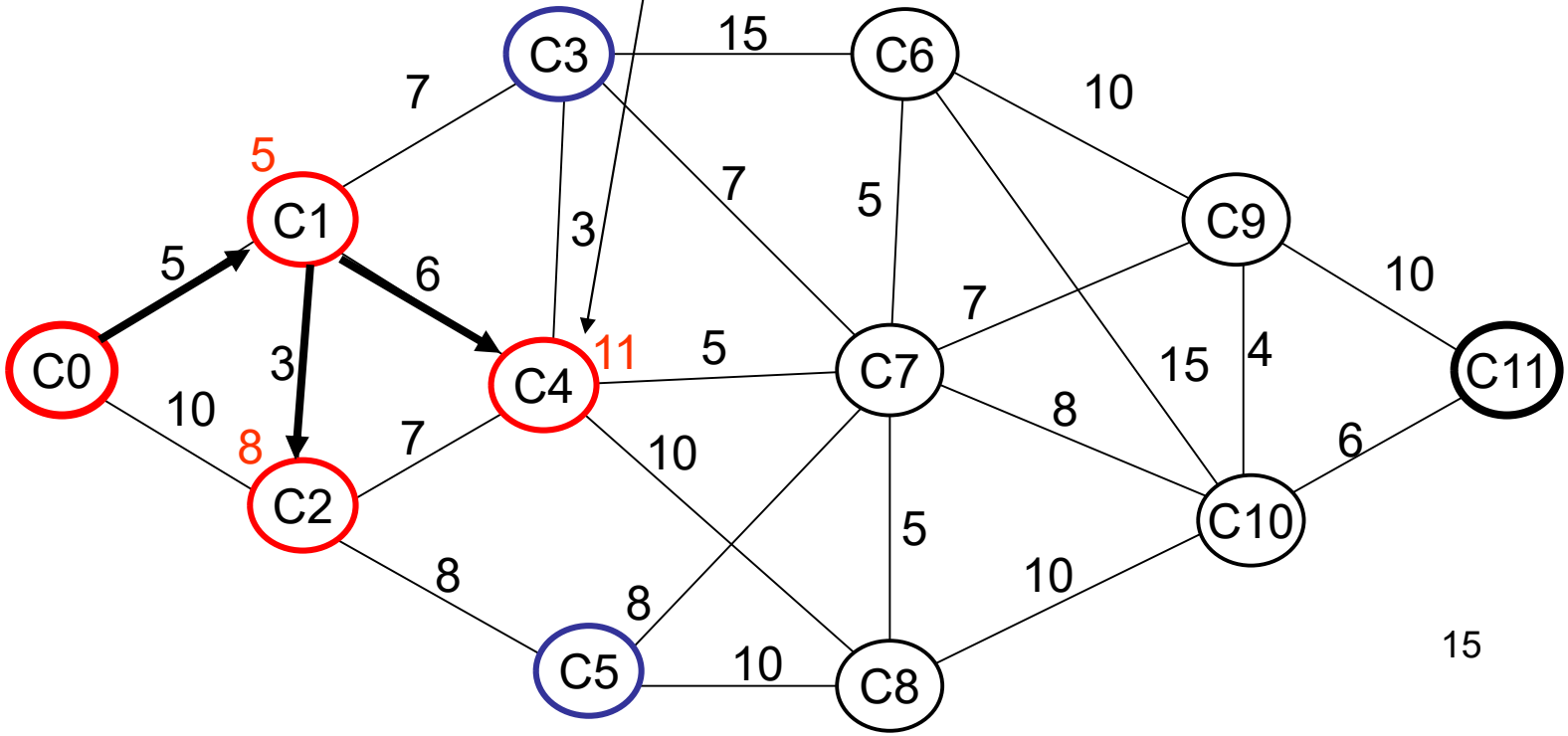
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

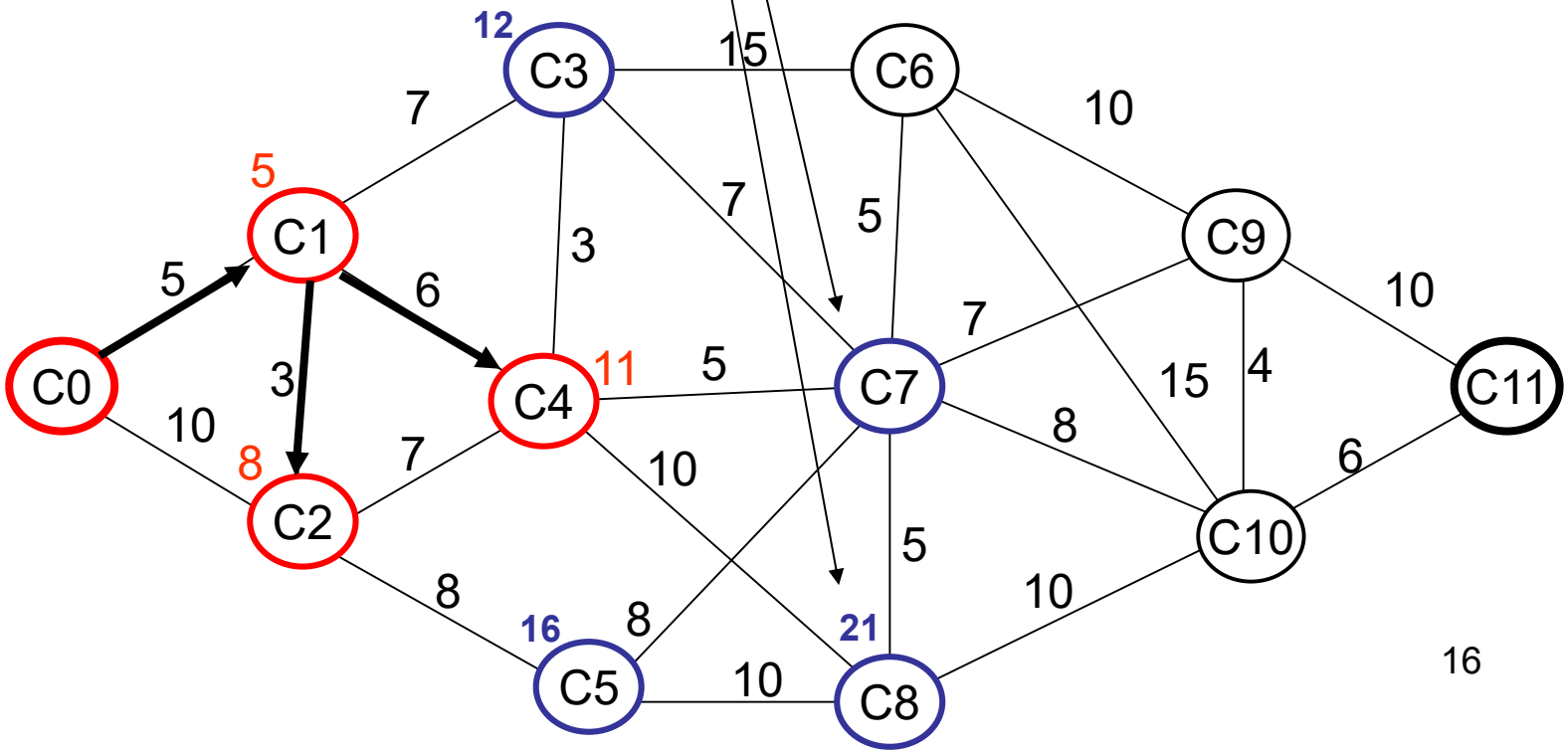
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

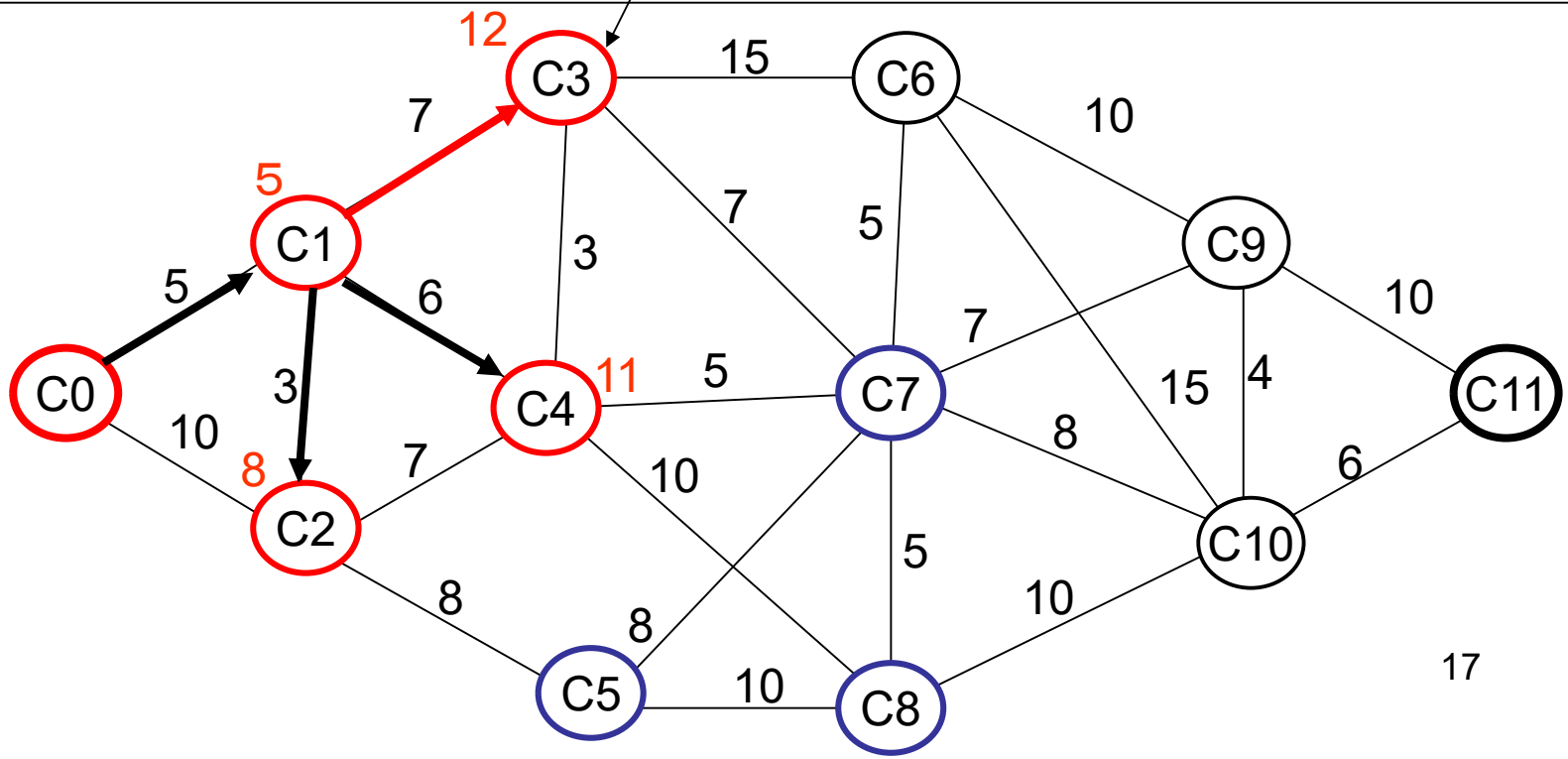
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

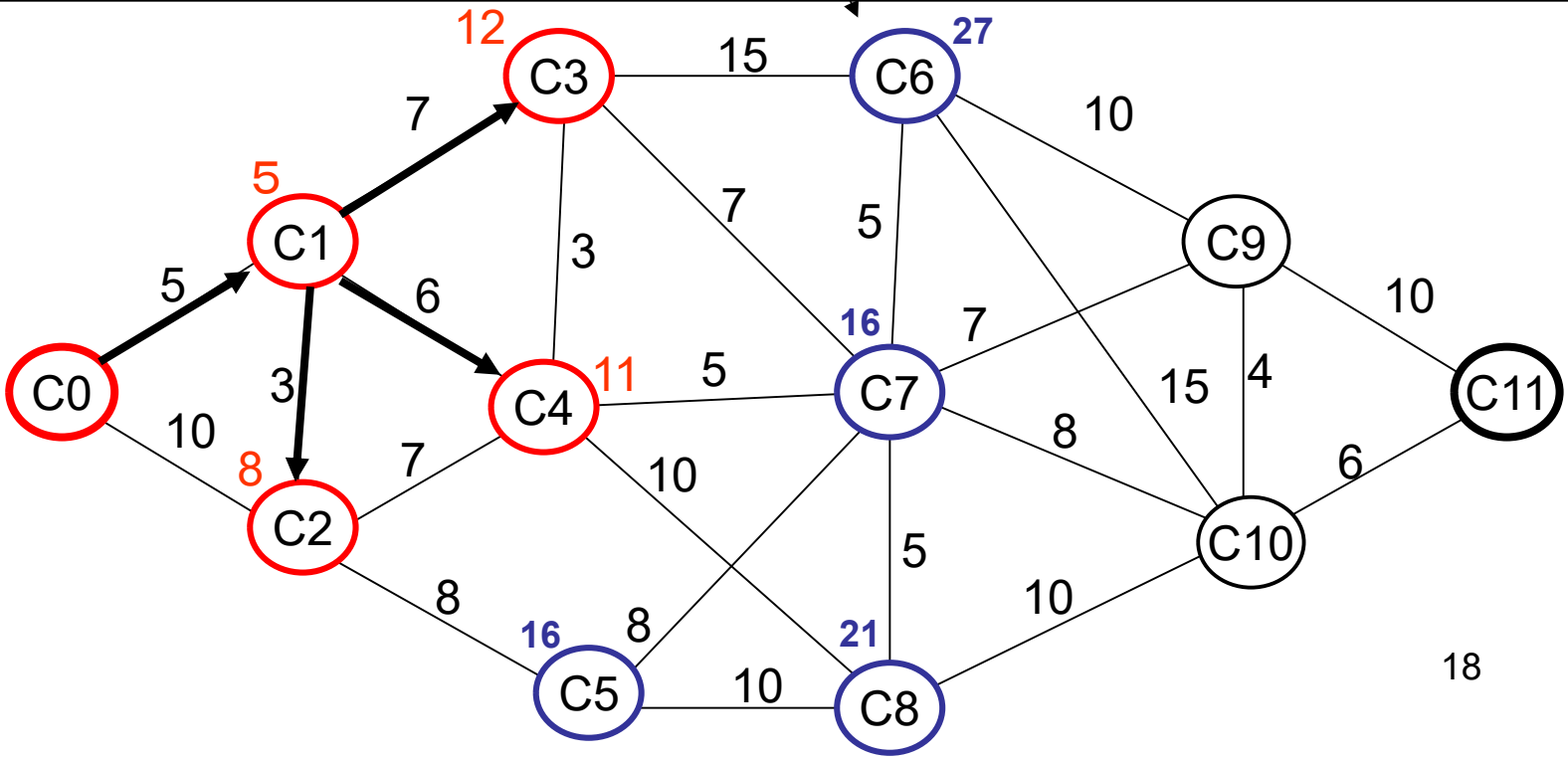
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

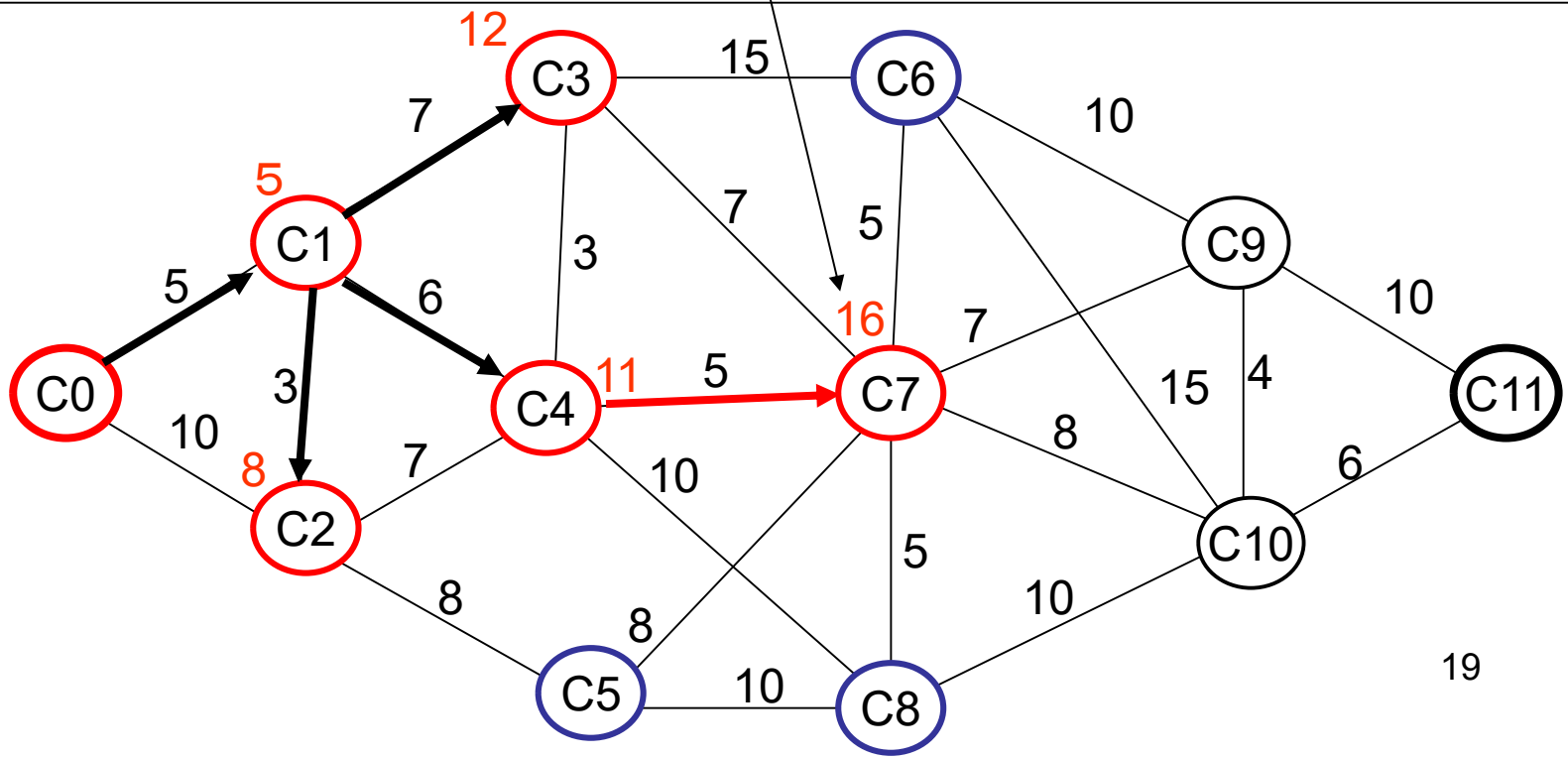
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

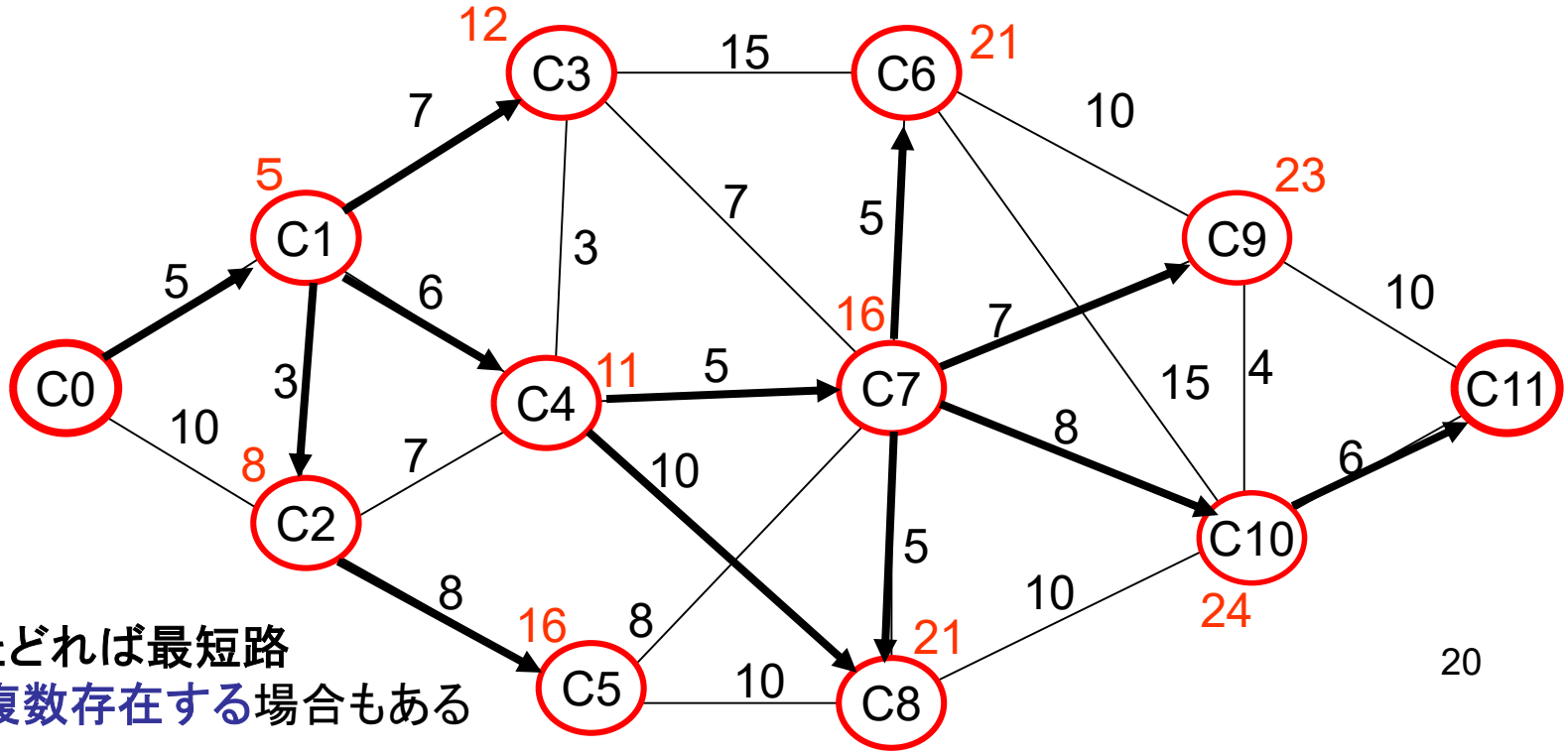
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【最短経路探索：ダイクストラの方法】

1） 全ての節点をつぎの3種類に分類する：
   1．始点からその節点へ行く最短の経路が分かった節点の集合　T
   2．集合Tに属する節点から直接訪れることのできる節点の集合　F
   3．TとF以外の節点の集合　　U

2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



・矢印を逆にたどれば最短路
・最短経路は複数存在する場合もある

【Pythonによる ダイクストラ法の実行】

```
import numpy as np
from scipy.sparse.csgraph import shortest_path

#To:          0  1  2  3  4  5  6  7  8  9 10 11
graph = [[ 0,  5, 10,  0,  0,  0,  0,  0,  0,  0,  0,  0],
         [ 5,  0,  3,  7,  6,  0,  0,  0,  0,  0,  0,  0],
         [10,  3,  0,  0,  7,  8,  0,  0,  0,  0,  0,  0],
         [ 0,  7,  0,  0,  3,  0, 15,  7,  0,  0,  0,  0],
         [ 0,  6,  7,  3,  0,  0,  0,  5, 10,  0,  0,  0],
         [ 0,  0,  8,  0,  0,  0,  0,  8, 10,  0,  0,  0],
         [ 0,  0,  0, 15,  0,  0,  0,  5,  0, 10, 15,  0],
         [ 0,  0,  0,  7,  5,  8,  5,  0,  5,  7,  8,  0],
         [ 0,  0,  0,  0, 10, 10,  0,  5,  0,  0, 10,  0],
         [ 0,  0,  0,  0,  0,  0, 10,  7,  0,  0,  4, 10],
         [ 0,  0,  0,  0,  0,  0, 15,  8, 10,  4,  0,  6],
         [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 10,  6,  0]]

a = np.array( graph )
(cost, path) = shortest_path( a, return_predecessors=True)
print("cost matrix-------------")
print( cost )
print("predecessors------------")
print( path )
```

ノードjへ

ノードiから

前スライドに示した例題の有向グラフの隣接行列

ノード0→1のコスト5

ノード1→2のコスト3

…

グラフ最短経路探索のソルバ

最短経路における1つ前の頂点

【実行結果】

```
>python graph.py
cost matrix------------
[[ 0.  5.  8. 12. 11. 16. 21. 16. 21. 23. 24. 30.]
 [ 5.  0.  3.  7.  6. 11. 16. 11. 16. 18. 19. 25.]
 [ 8.  3.  0. 10.  7.  8. 17. 12. 17. 19. 20. 26.]
 [12.  7. 10.  0.  3. 15. 12.  7. 12. 14. 15. 21.]
 [11.  6.  7.  3.  0. 13. 10.  5. 10. 12. 13. 19.]
 [16. 11.  8. 15. 13.  0. 13.  8. 10. 15. 16. 22.]
 [21. 16. 17. 12. 10. 13.  0.  5. 10. 10. 13. 19.]
 [16. 11. 12.  7.  5.  8.  5.  0.  5.  7.  8. 14.]
 [21. 16. 17. 12. 10. 10. 10.  5.  0. 12. 10. 16.]
 [23. 18. 19. 14. 12. 15. 10.  7. 12.  0.  4. 10.]
 [24. 19. 20. 15. 13. 16. 13.  8. 10.  4.  0.  6.]
 [30. 25. 26. 21. 19. 22. 19. 14. 16. 10.  6.  0.]]
predecessors-----------
[[-9999     0     1     1     1     2     7     4     4     7     7    10]
 [    1 -9999     1     1     1     2     7     4     4     7     7    10]
 [    1     2 -9999     1     2     2     7     4     4     7     7    10]
 [    1     3     1 -9999     3     7     7     3     7     7     7    10]
 [    1     4     4     4 -9999     7     7     4     4     7     7    10]
 [    1     2     5     7     7 -9999     7     5     5     7     7    10]
 [    1     4     4     7     7     7 -9999     6     7     6     7    10]
 [    1     4     4     7     7     7     7 -9999     7     7     7    10]
 [    1     4     4     7     8     8     7     8 -9999     7     8    10]
 [    1     4     4     7     7     7     9     9     7 -9999     9     9]
 [    1     4     4     7     7     7     7    10    10    10 -9999    10]
 [    1     4     4     7     7     7     7    10    10    11    11 -9999]]
```

ノードｉからスタートして
ノードｊをゴールとした
最短経路のコスト

ノードｉからスタートして
ノードｊへ至る最短経路
のｊの1つ前のノード

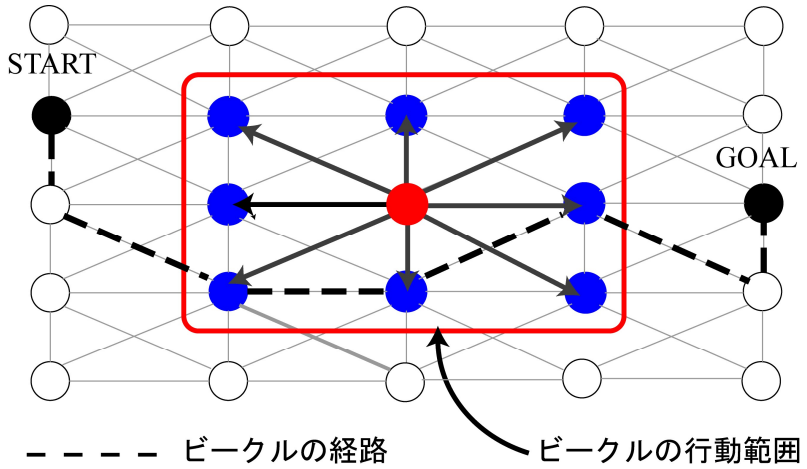
ノード0→11
への経路を得る

ノード8へつな
がる経路は4
と7の2つだが
4のみ表示

AUVの経路計画問題

経路計画問題における仮定

- 定常速度 c で航行
- AUV の経由点は格子点空間で構成
- 運動学的・力学的特性は考えない
- 到着日時は指定しない
- 経路 $\boldsymbol{P}$ は出発点 p_0 から目的点 p_m までの経由点を示す



$$(\boldsymbol{P} = \{p_0, p_1, \ldots, p_i, \ldots, p_m\})$$

エネルギー消費のコスト関数

$$Cost(p_i, p_{i+1}) = \iint_{p_i}^{p_{i+1}} \frac{\rho}{c} \|\boldsymbol{V}_i(x,y)\|^3 dxdy$$

$\boldsymbol{V}_i(x,y) = c\boldsymbol{e}_i - \boldsymbol{v}_c(x,y)$

$:AUV$ と潮流の相対速度

$\boldsymbol{e}_i$ $:AUV$ の速度の単位ベクトル

$\boldsymbol{v}_c(x,y)$: 潮流の速度ベクトル

経路計画問題の定式化

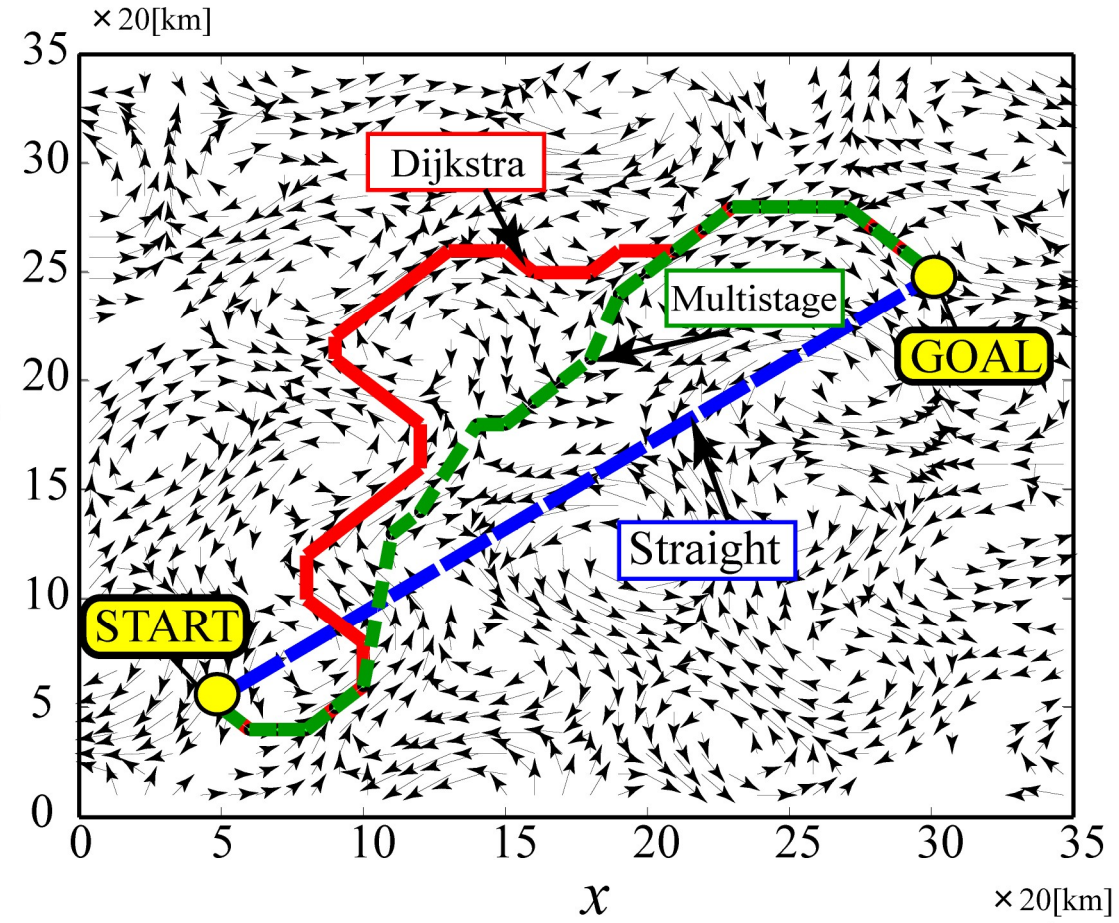
Minimize

$$Total\ cost(\boldsymbol{P}) = \sum_{i=0}^{m-1} Cost(p_i, p_{i+1})$$

（Case 1）複数の渦が存在する潮流場

探索手法

- Dijkstra
  (最短経路問題,**Dijkstra**法)
- Multistage
  ゴールから遠ざかる方向のエッジが選択できない
  (多段決定問題,動的計画法)
- Straight
  (直進した場合)



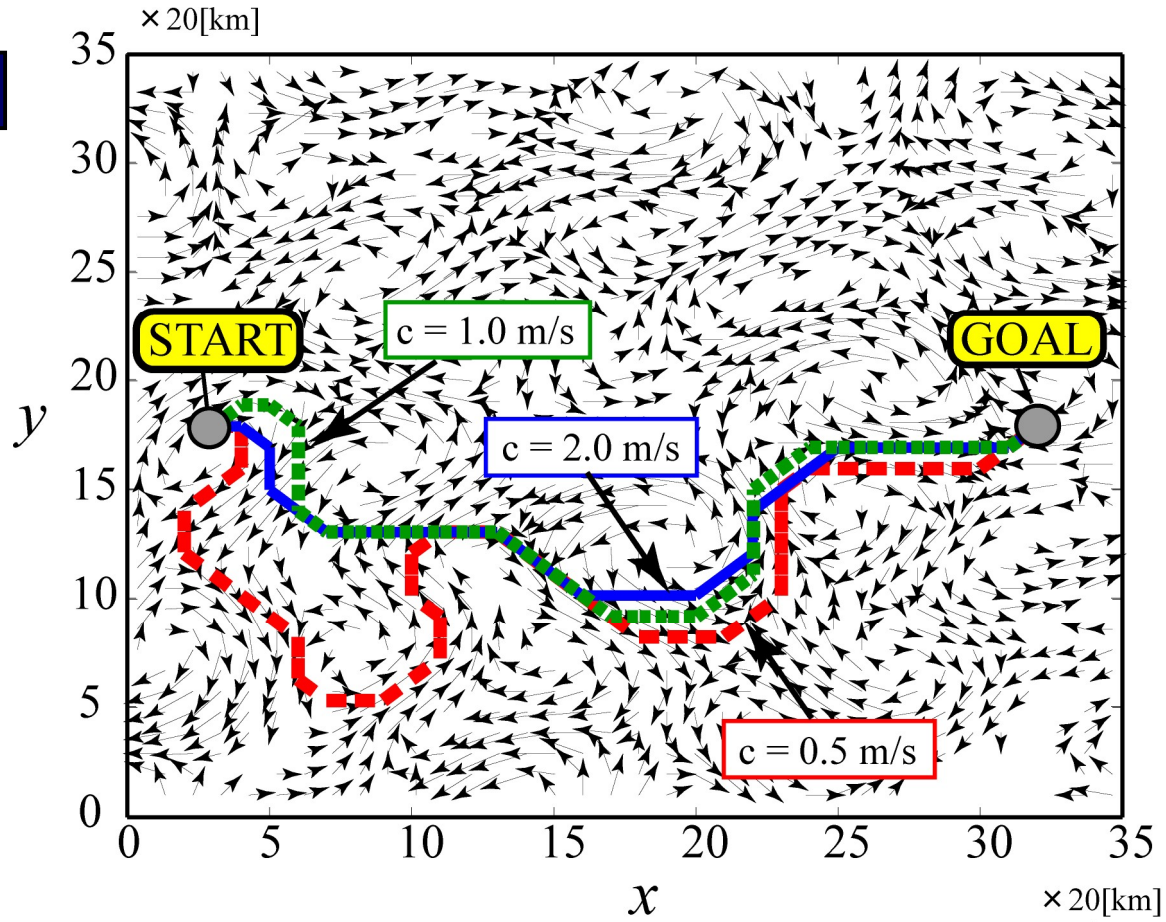
	Cost function $[J]$	Computation time $[ms]$
Straight	42531	13708
Multistage	1553	17219
Dijkstra	965	55360

（Case 2）定常速度を変化させた場合

探索手法

- Dijkstra法を適用
(最短経路問題,Dijkstra法)

3種類の定常速度で探索

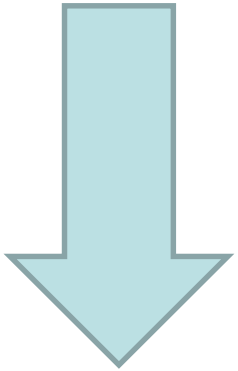


	Cost function $[J]$	Computation time $[ms]$
c = 0.5 m/s	2725	102484
c = 1.0 m/s	13714	58453
c = 2.0 m/s	87007	41488

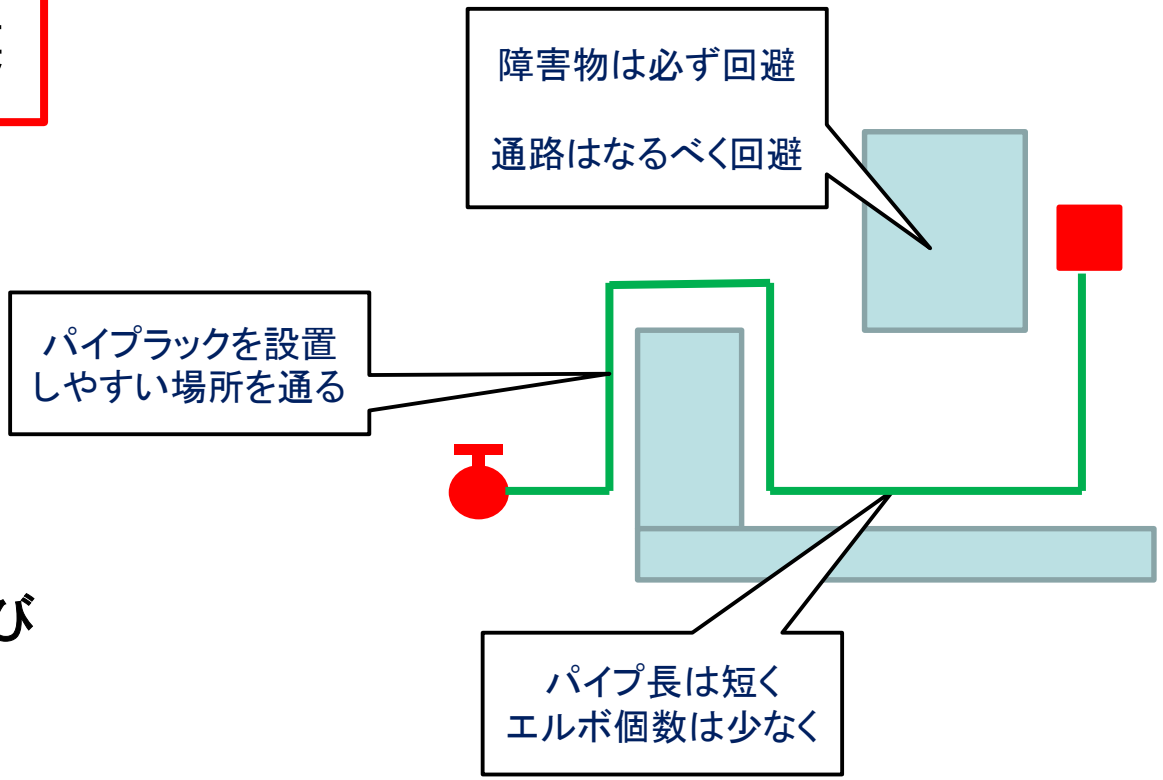
【ダイクストラ法による配管設計】

・知識と経験を必要とする配管設計作業の自動化・省力化

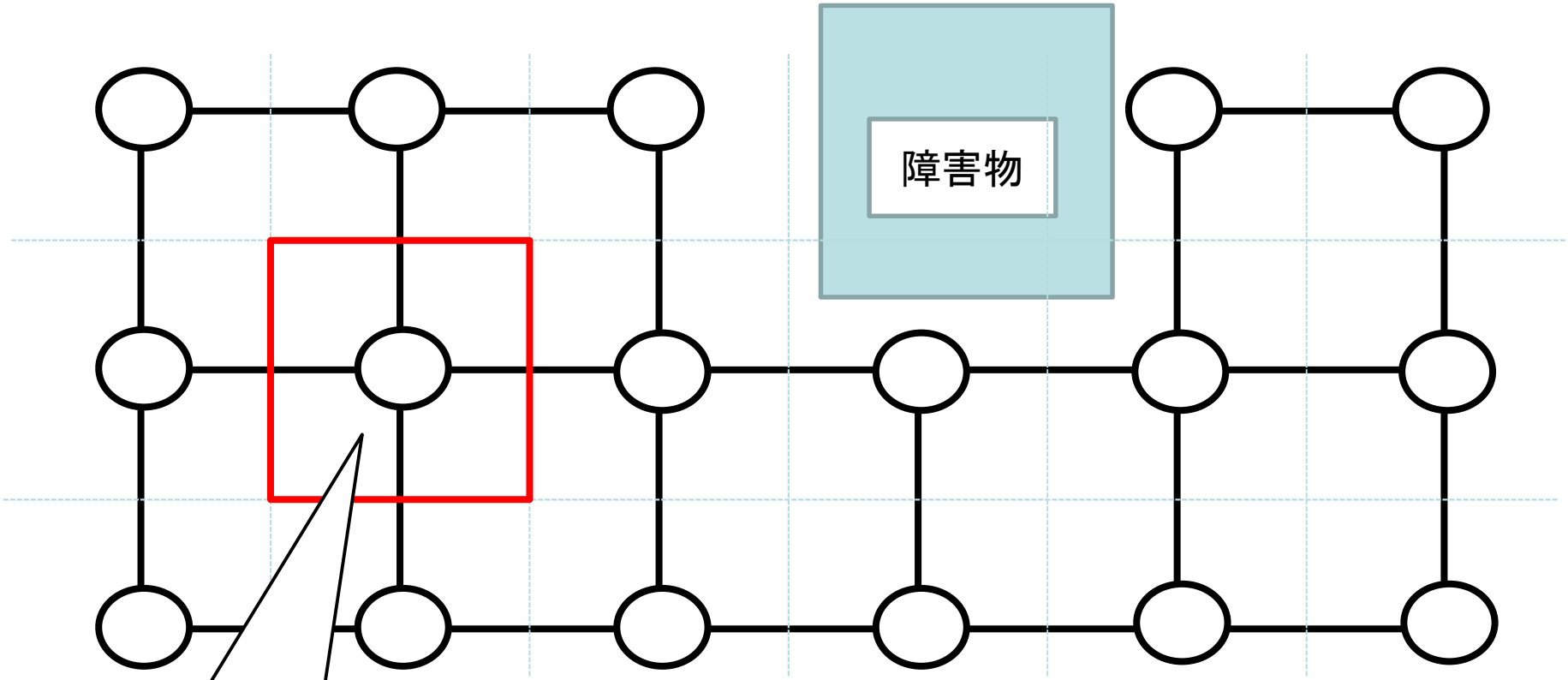
2つの機器間を結ぶ
パイプルーティング作業



障害物の個数や形状、および
パイプ直径に左右されない
強力なアルゴリズムを構築



ダイクストラ法を用いた先行研究における設計対象空間のネットワーク表現方法



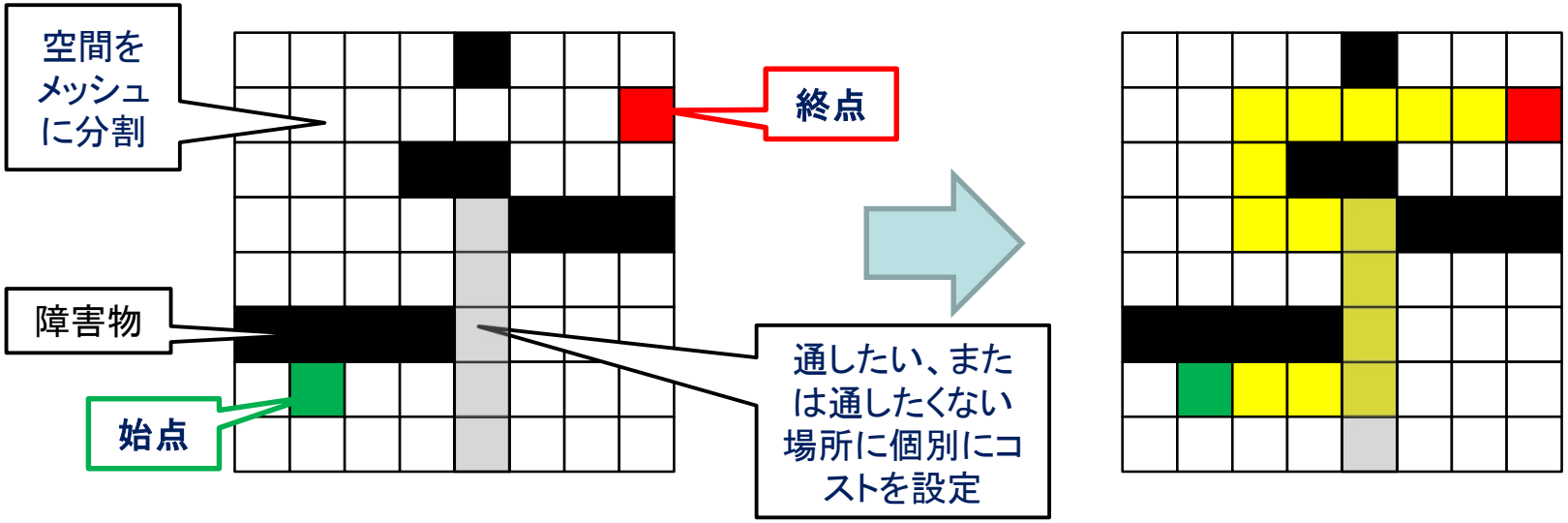
あるセルから、6方向の
隣接するセルへ移動可能

・セルはパイプ直径より
大きくなければならない
・エルボのコストが表現できない

ダイクストラ法による新しいパイプ経路探索法

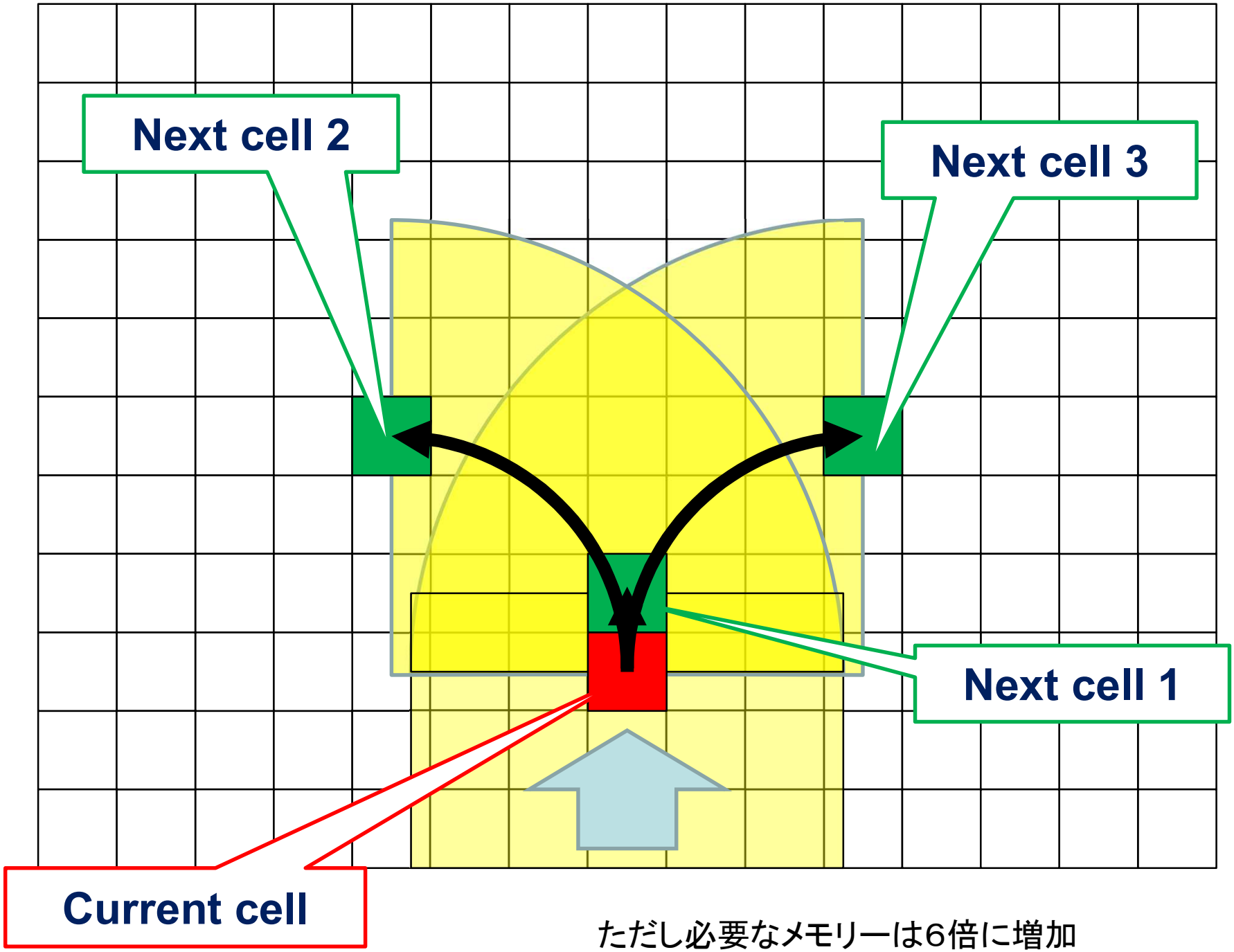
・ダイクストラ法とは？
　重み付きグラフ（ネットワーク）上の２点間の最短経路を求めるアルゴリズム
　最適解が保証される

・多くの先行研究において、配管経路探索に利用されている



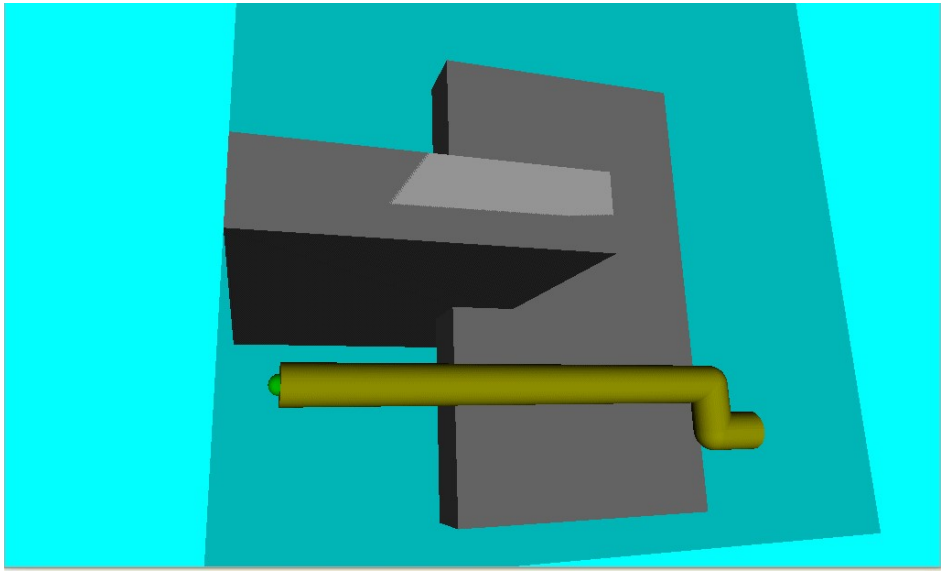
【問題点】 デザイン対象空間を**パイプ径以上の幅を持つ格子で分割**
　→　設計上の制約が大きく、**非実用的**

そこで…　パイプの「向き」を状態量として持たせる
→　**パイプ径より小さな格子分割で経路探索**

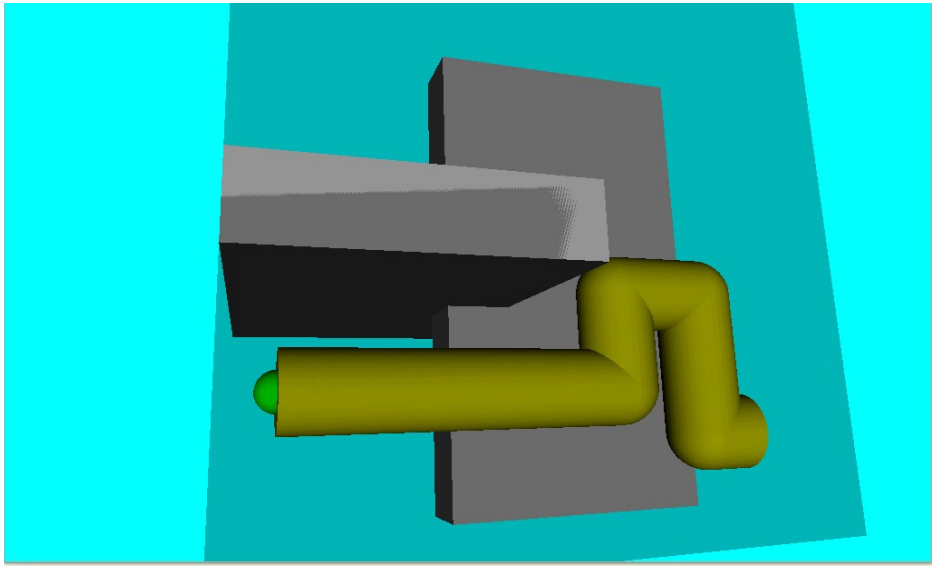


ただし必要なメモリーは6倍に増加

計算機シミュレーションによるルーティング例



パイプ径：200mm

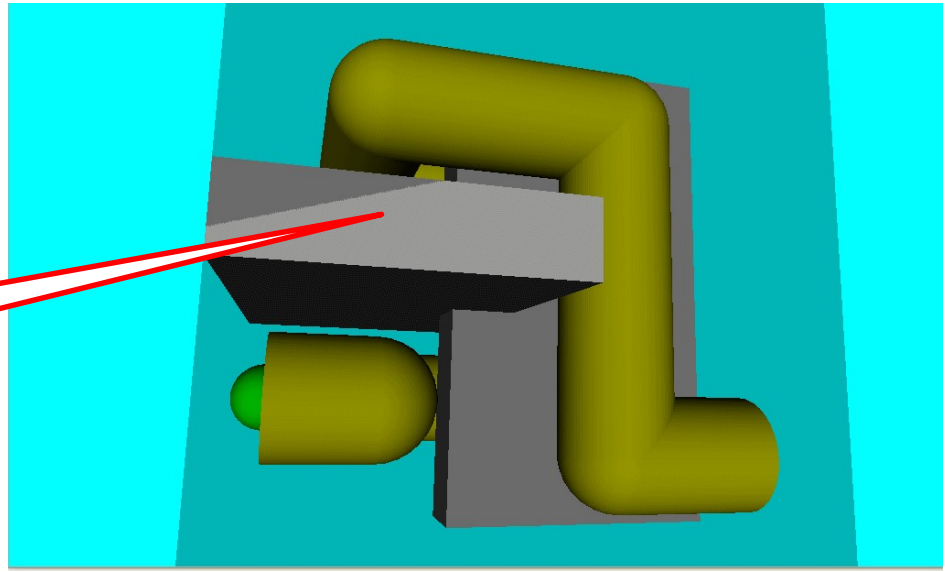


パイプ径：400mm

格子の幅：全て250mm

始点・終点の座標と方向は同じ

径が増加してエルボが曲がりきれなくなり、他のルートをとる



パイプ径：600mm

空間を離散化するための格子の幅は、
場所に応じて変えることも可能

【組合せ最適化問題の難しさ（続き2）】

●**NP完全問題（NP-complete problems）**：
有限時間で解けるが、
多項式時間で解けるようなアルゴリズムがない（見つかっていない）問題群

NP完全問題の例：　巡回セールスマン問題（Traveling Salesman Problem: TSP）、
2次割り当て問題（Quadratic Assignment Problem: QAP）
ナップサック問題（knapsack problem）、クリーク問題（creek）
~~ジョブショップスケジューリング問題（job shop scheduling）~~

NP完全問題の正準問題：**SAT（Boolean satisfiability problem）**

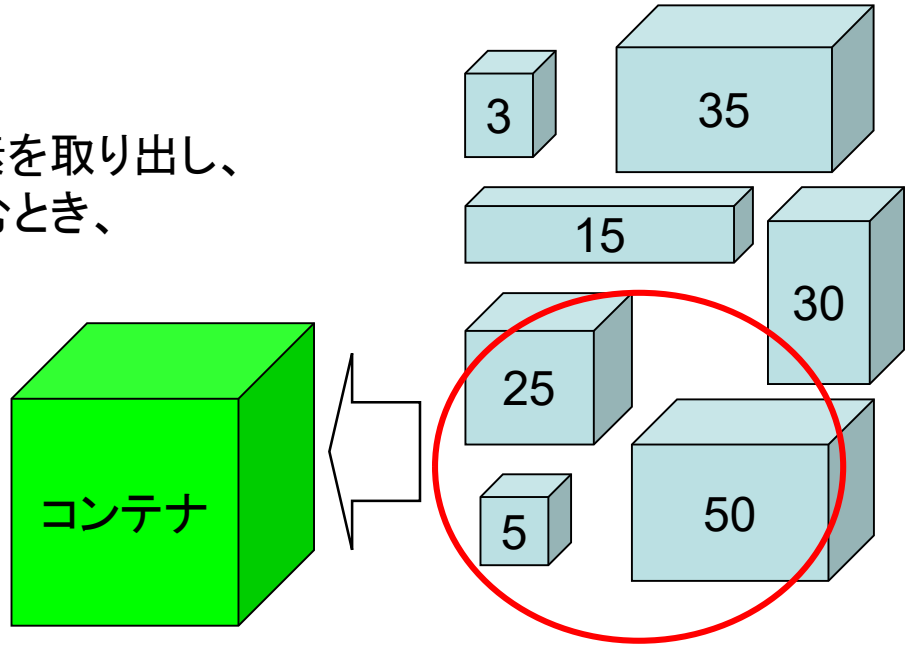
クックの定理（Cooke's Theorem）：
全てのNP完全問題は多項式時間内にSATへ変換できる

●**SAT（Boolean satisfiability problem）とは？**
論理式を真にする論理変数の組合せを判定する問題

例）　論理変数　a,b,c　のとき、
論理式　$(a \cup b) \cap c$　が true になるような変数の組合せは存在するか？

【ナップサック問題】

異なる大きさと価値を有する要素の集合から要素を取り出し、
容量に制限のある容器（ナップサック）に詰め込むとき、
価値の合計が最大になるように詰め込む問題。



ナップサックの容量 W

w_i 要素 i の重さ $[w_1, w_2, \cdots w_n]$

p_i 要素 i の価値 $[p_1, p_2, \cdots p_n]$

選択行列 $[x_1, x_2, \cdots x_n]$

$$x_i = \begin{cases} 1 & \text{要素 i をナップサックへ入れる場合} \\ 0 & \text{それ以外} \end{cases}$$

探索空間： 2^n

ナップサックに詰め込んだ価値の合計 $f = \sum_{i=1}^{n} x_i \, p_i$

ただし制約条件 $\sum_{i=1}^{n} x_i \, w_i \leq W$

整数計画問題

重さ合計がナップサック容量を超えない範囲内で、
価値の合計 f を最大化する選択行列を求める

コスト関数も制約条件も1次式なので
線形計画問題に見えるが、変数が離散値なので線形計画法は使えない

【ナップサック問題の例】

ナップサックの
容量： 8

要素	1	2	3	4	5
重さ	4	5	2	1	6
価値	45	55	20	10	60

ナップサックの
容量： 112

この程度の規模なら全探索で簡単に求められる

要素	1	2	3	4	5	6	7	8
重さ	6	20	25	30	40	30	60	10
価値	15	100	90	60	40	15	10	3

TSPやQAPについては、ベンチマーク問題を集めたWebサイトがあり、
探索アルゴリズムの性能評価のために利用できる

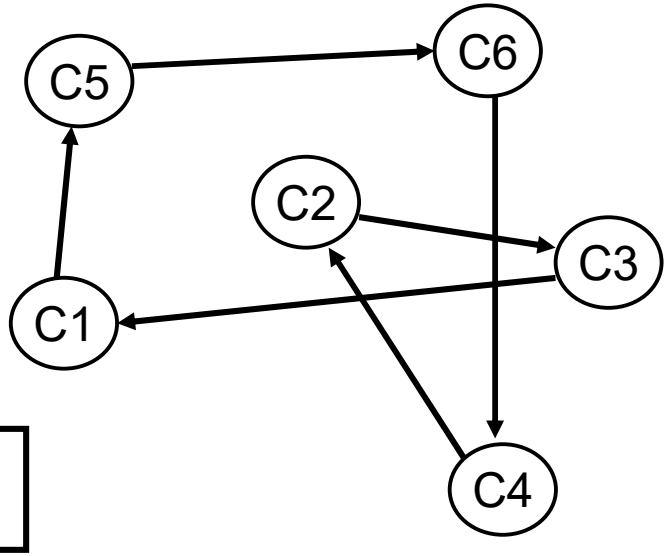
TSPのベンチマークを集めたサイト：TSPLIB
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

QAPのベンチマークを集めたサイト：QAPLIB
http://www.opt.math.tu-graz.ac.at/qaplib/

その他　東京海洋大学流通情報工学科　久保先生のサイト：
http://www.ipc.tosho-u.ac.jp/~kubo/

【巡回セールスマン問題：TSP】

始点にいるセールスマンが、n 個の町を訪問してもとの始点に帰ってくるのに、どのルートがコスト最小か？



コスト行列 C_{ij} 都市 i から j へ移動するコスト

$$\mathbf{C}=\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & & \\ \vdots & & \ddots & \vdots \\ c_{n1} & & \cdots & c_{nn} \end{bmatrix}$$

都市 i → j へ直接行けない場合、コスト無限大

巡回路のコスト：$f=\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\,x_{ij}$

巡回路行列 x_{ij}

$$\mathbf{X}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & & \\ \vdots & & \ddots & \vdots \\ x_{n1} & & \cdots & x_{nn} \end{bmatrix}$$

都市 i から j へ移動するとき　1　それ以外　0

ただし

$\sum_{i=1}^{n} x_{ij}=1$　町 j に入る矢印は一つ

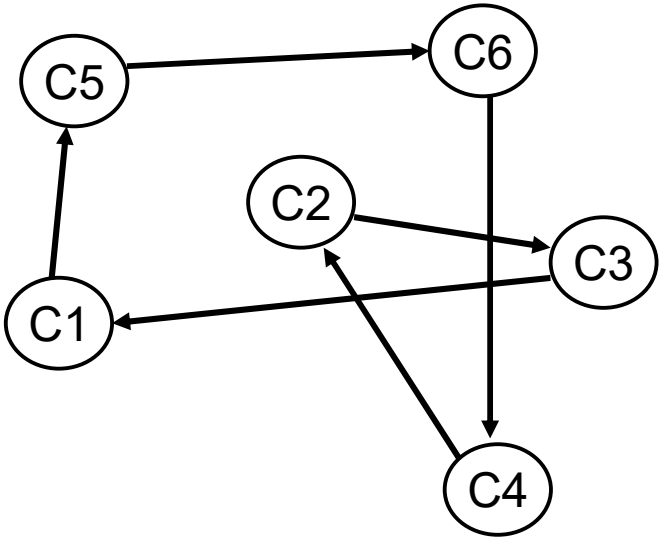
$\sum_{j=1}^{n} x_{ij}=1$　町 i から出る矢印は一つ

$x_{ii}=0$　町から出た矢印は、同じ町に入らない

コスト最小になる巡回路行列を見つける

【巡回セールスマン問題：TSP】

巡回路行列
都市 i から j へ移動するとき　1　それ以外　0



$$x_{ij}$$

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & & \\ \vdots & & \ddots & \vdots \\ x_{n1} & & \cdots & x_{nn} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

ただし、右式の条件を満たすことは
一筆書きの巡回路を形成する必要条件
ではあるが十分条件ではない

また、コスト関数も制約条件も1次式なので
線形計画問題に見えるが、変数が離散値なので線形計画法は使えない

整数計画問題

$$\sum_{i=1}^{n} x_{ij} = 1$$

町 j に入る矢印は一つ

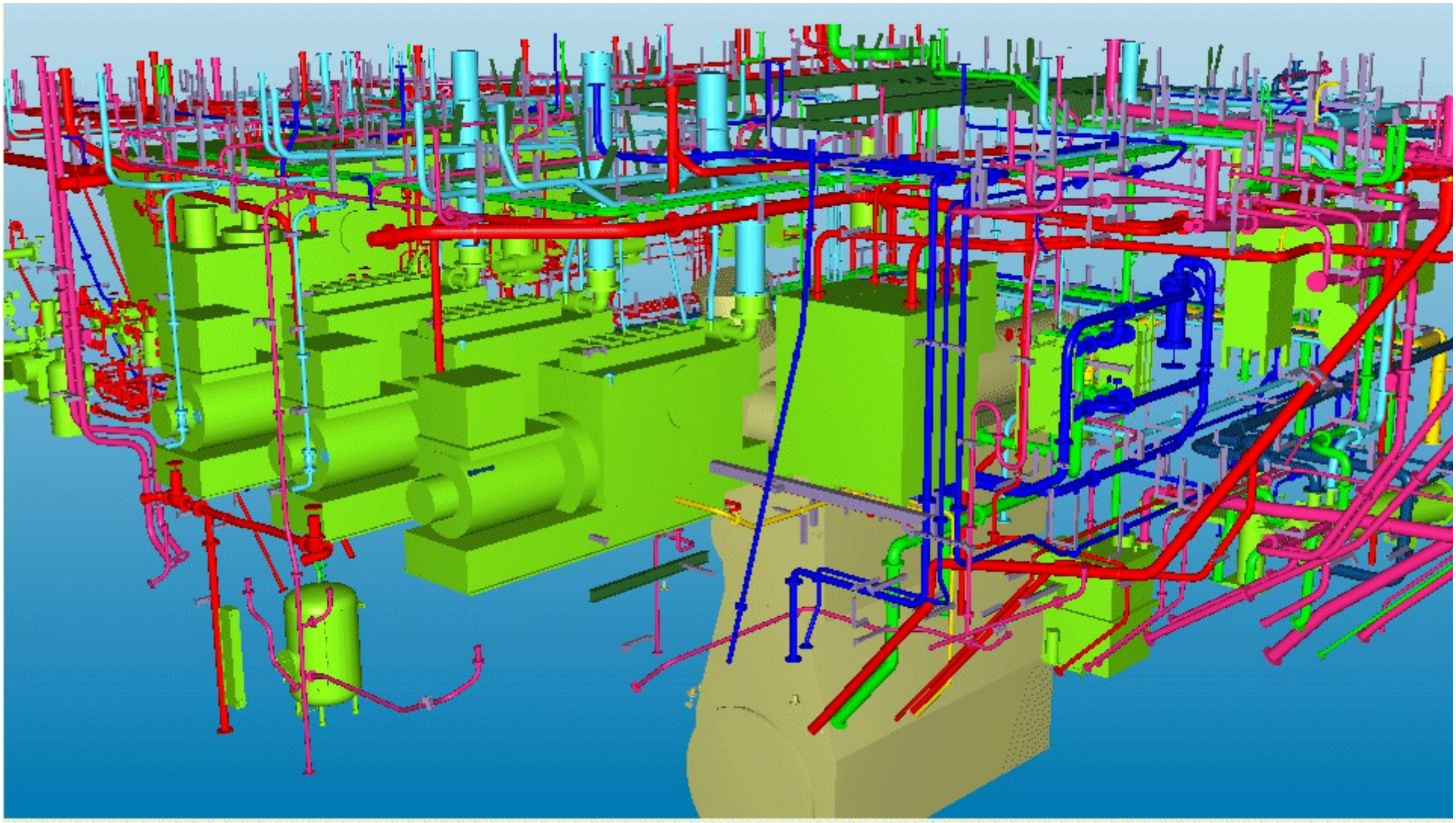
$$\sum_{j=1}^{n} x_{ij} = 1$$

町 i から出る矢印は一つ

$$x_{ii} = 0$$

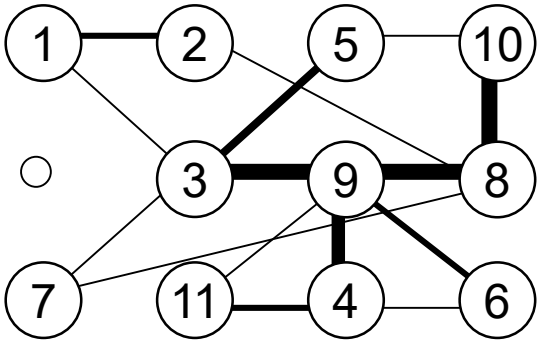
町から出た矢印は、
同じ町に入らない

船のエンジンルーム周辺はパイプだらけ　パイプは船価の大きな比率を占める



パイプ量を減らすには、**機器配置設計**が重要

【2次割当問題：QAP】



場所がn箇所あり、そこにm個の要素を配置する。
ただし、各要素間を流れる物流の単位長さあたりのコストと、
各場所の間の距離が与えられているとき、
コストが最小になる要素の配置は？

整数計画問題

配置行列 $x_{ij} = \begin{cases} 1 & \text{要素 i が場所 j に配置} \\ 0 & \text{それ以外} \end{cases}$

ただし $1 \le i \le m \quad 1 \le j \le n \quad m \le n$

ただし

$\sum_{i=1}^{m} x_{ij} \le 1$ 各場所に置かれる要素は1つ以下

$\sum_{j=1}^{n} x_{ij} = 1$ 各要素はどこか一箇所に必ず置く

コスト行列 a_{ik} 要素 i と要素 k 間の単位距離あたりのコスト

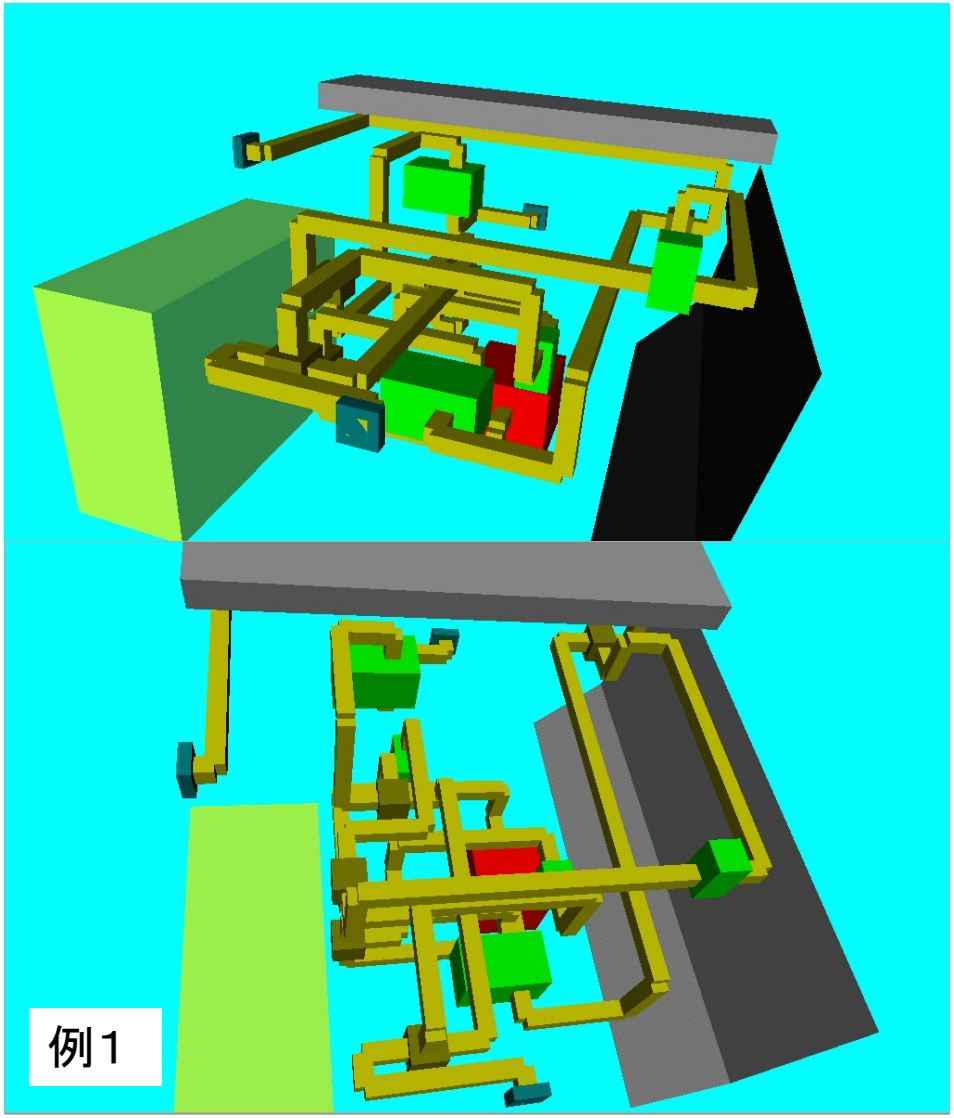
距離行列 $b_{j\ell}$ 場所 j と場所 ℓ 間の距離

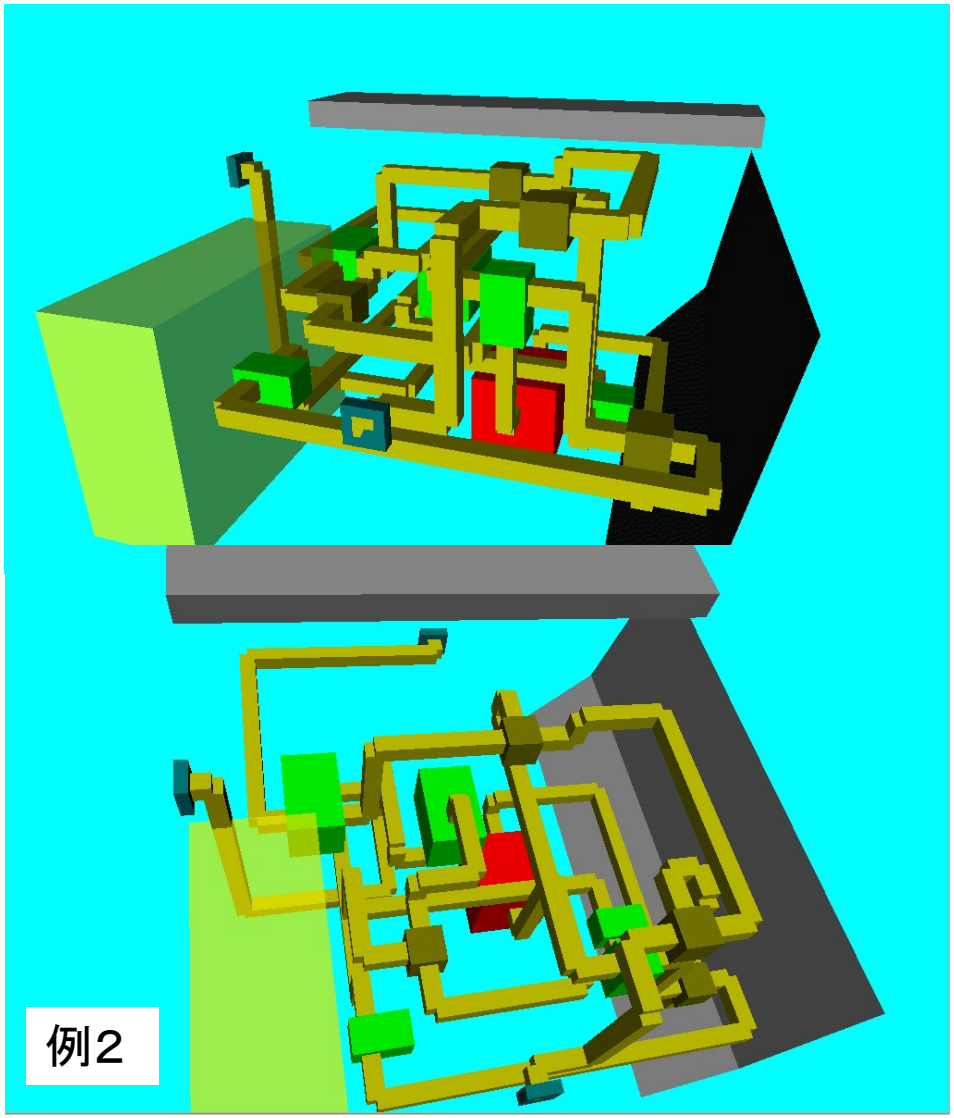
最適化すべき変数 x の2次式

総コスト： $f = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{\ell=1}^{n} a_{ik} b_{j\ell} x_{ij} x_{k\ell}$

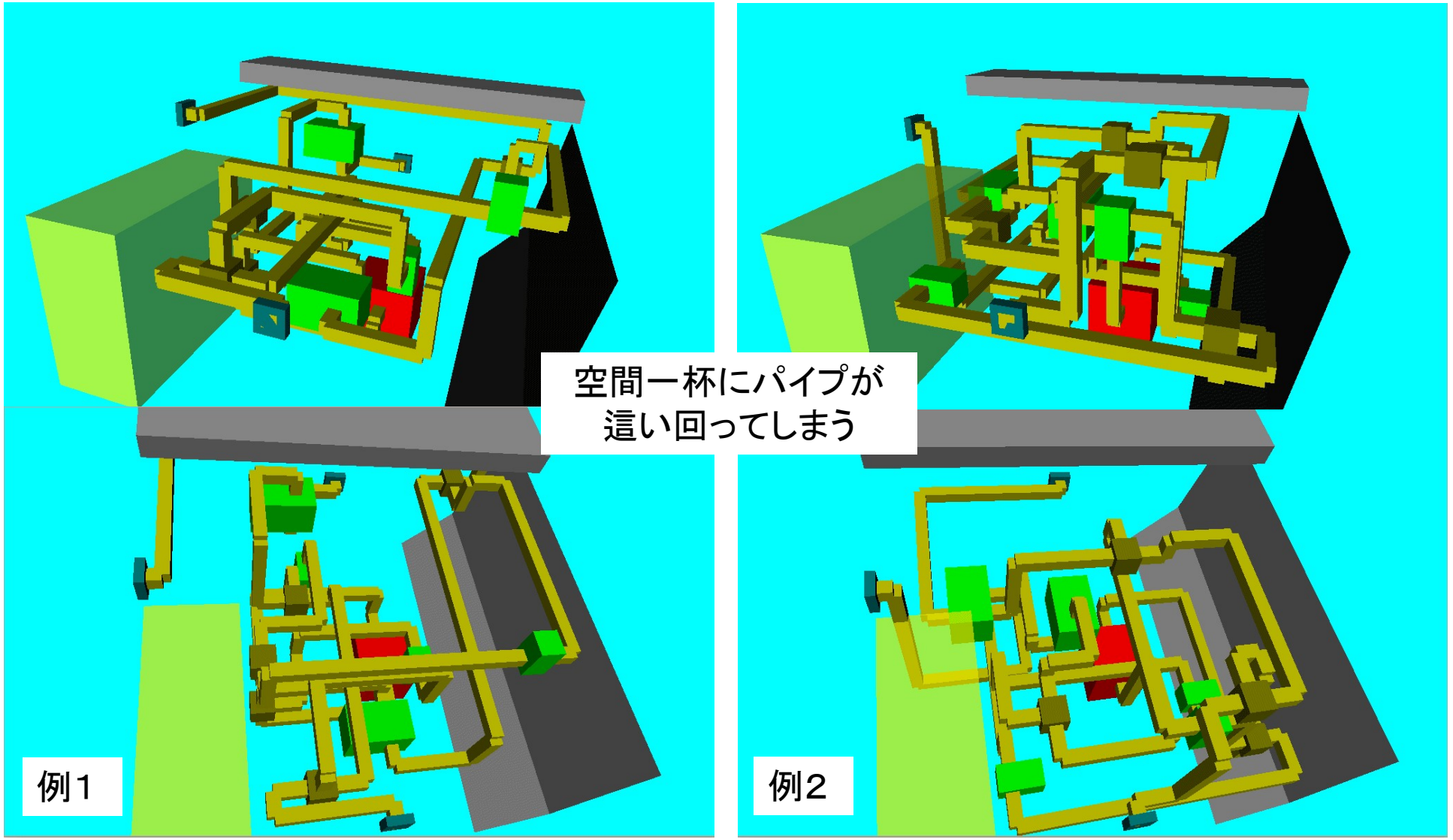
総コストが最小になる配置行列を見つける

ランダムな機器配置でパイプを接続すると…



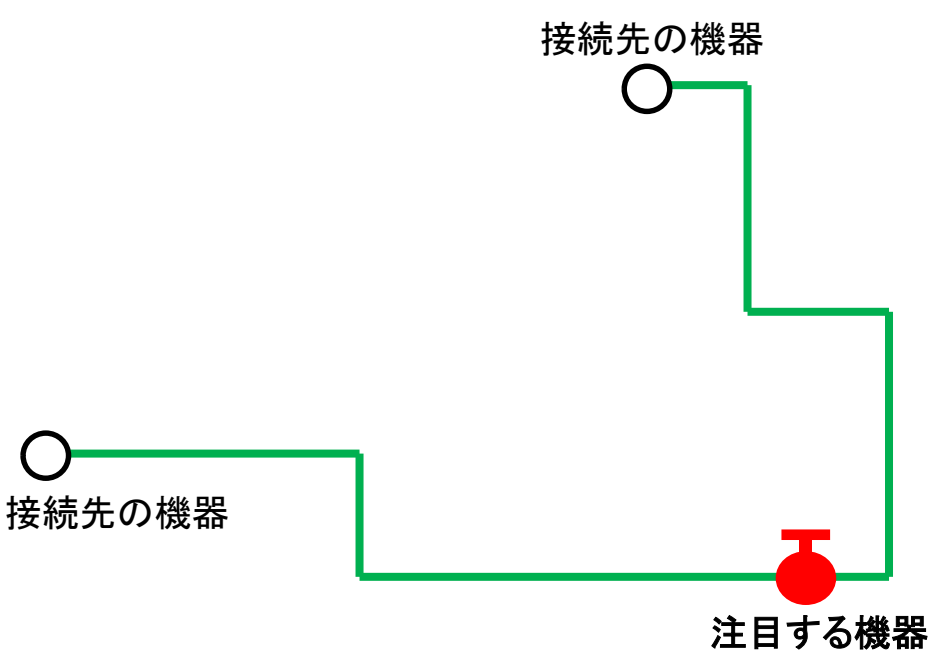


ランダムな機器配置でパイプを接続すると…

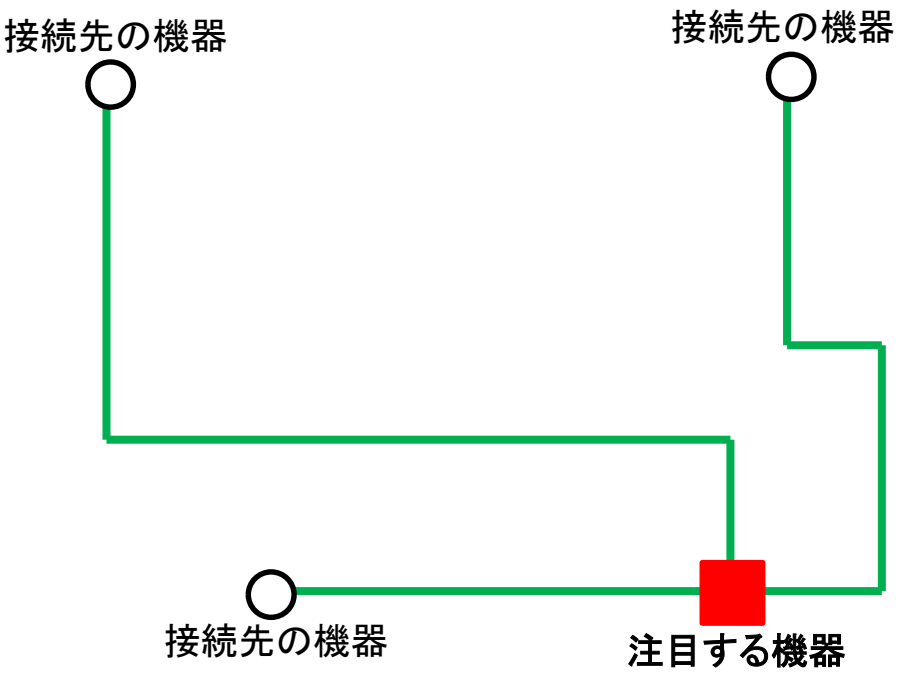


機器配置アルゴリズムの一例：

自己組織化機器配置法（一般的な手法ではない・ヒューリスティクス）



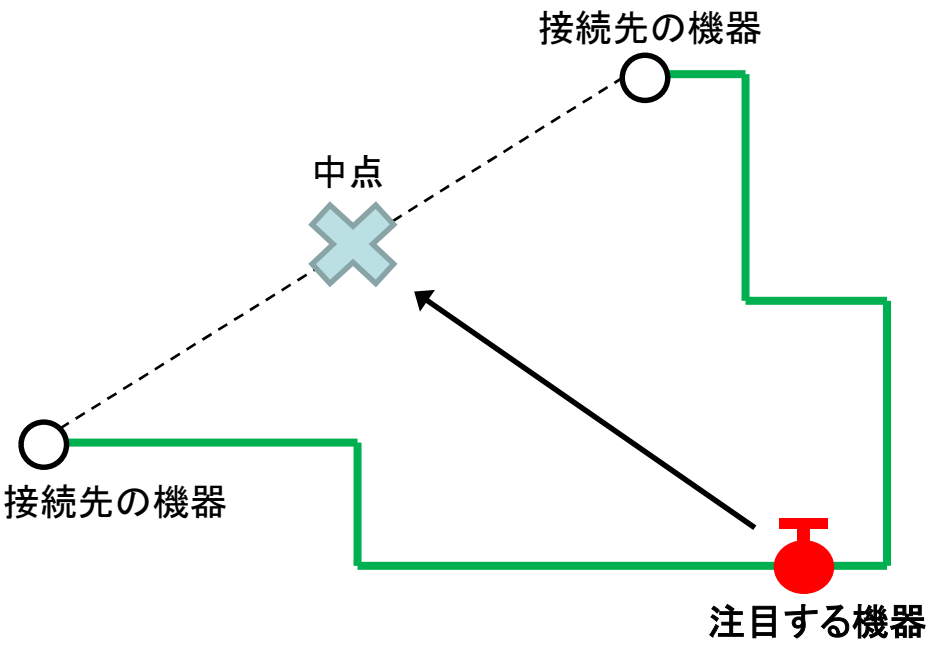
(a) 注目する機器の接続先が2箇所の場合



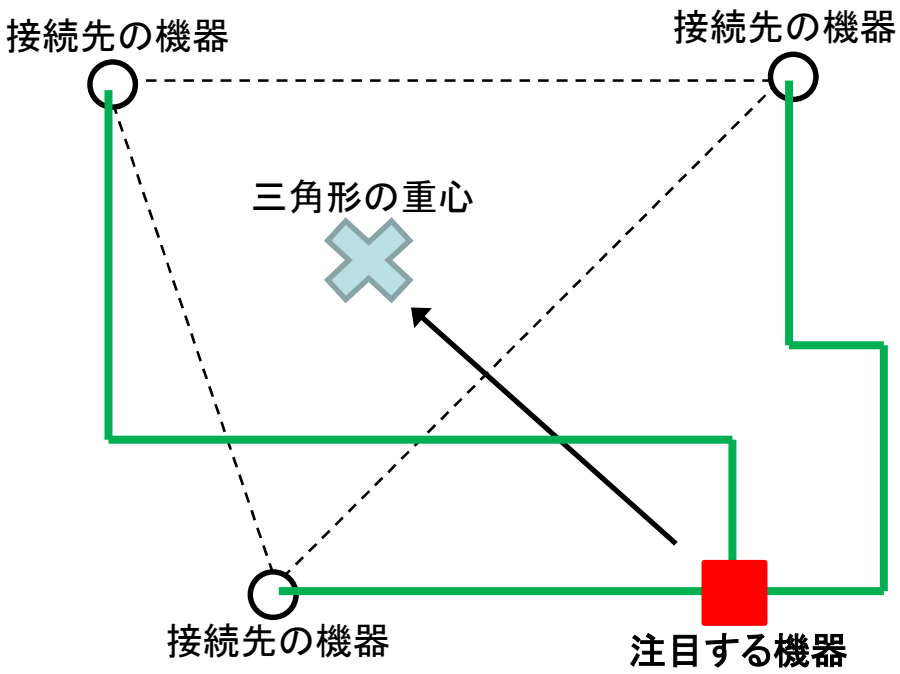
(b) 注目する機器の接続先が3箇所の場合

機器配置アルゴリズムの一例：

自己組織化機器配置法（一般的な手法ではない・ヒューリスティクス）



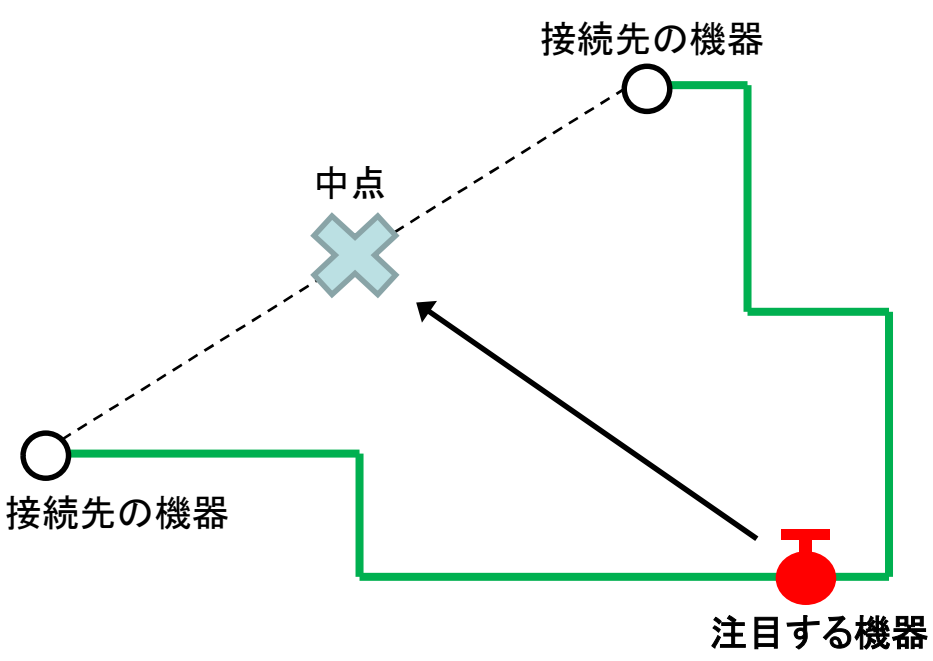
(a) 注目する機器の接続先が2箇所の場合



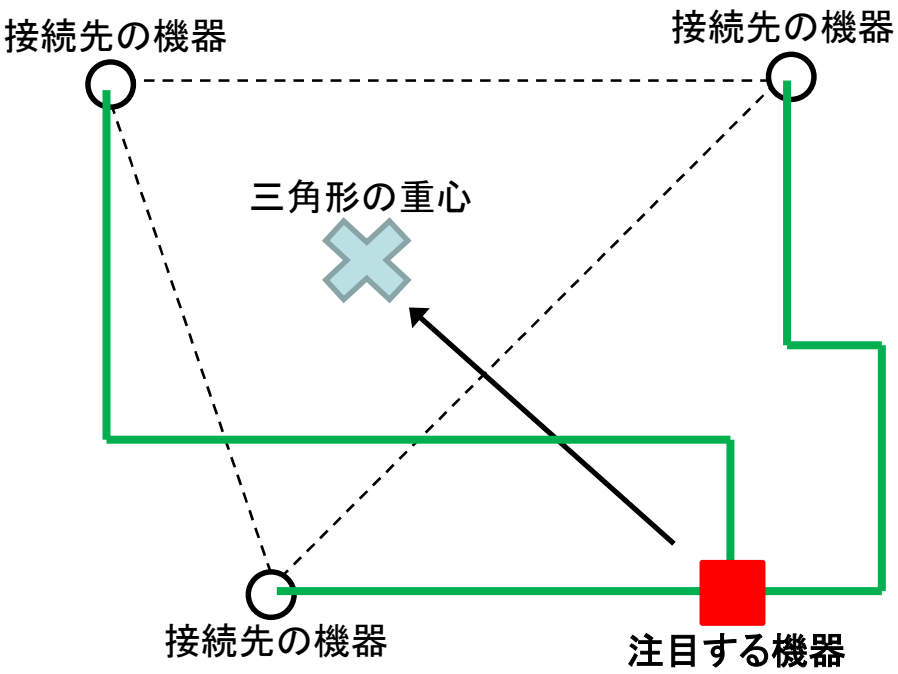
(b) 注目する機器の接続先が3箇所の場合

機器配置アルゴリズムの一例：

自己組織化機器配置法（一般的な手法ではない・ヒューリスティクス）



(a) 注目する機器の接続先が2箇所の場合

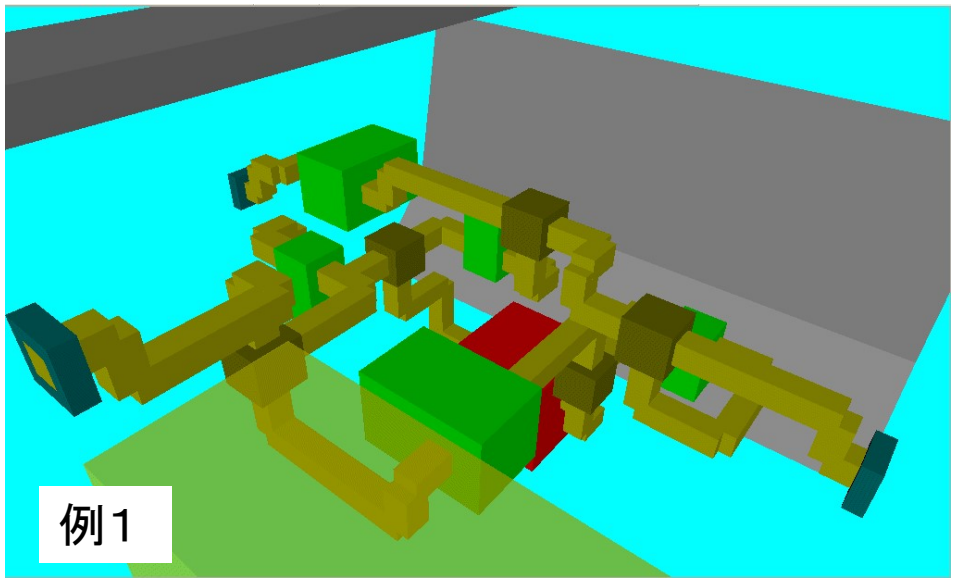


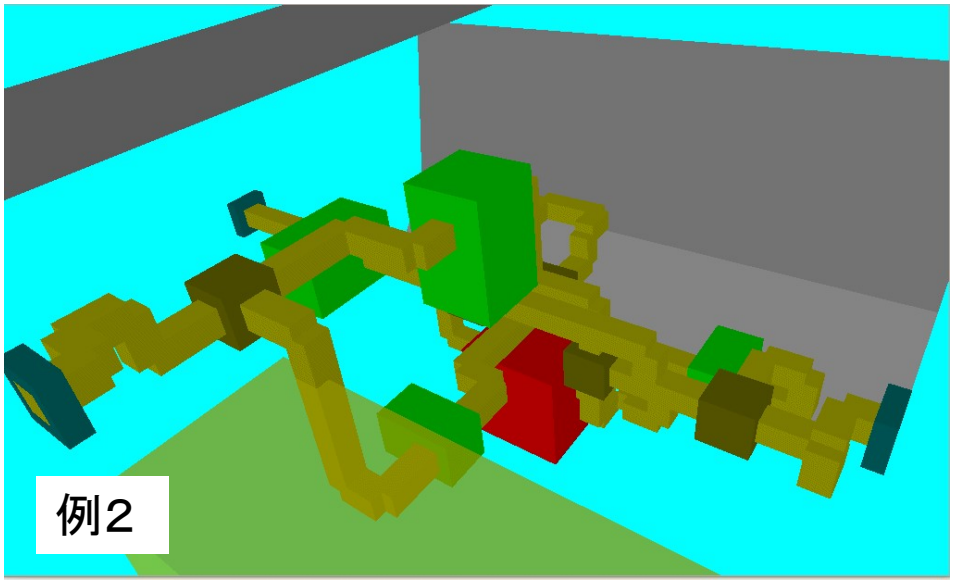
(b) 注目する機器の接続先が3箇所の場合

上記の操作を全ての機器についてランダムな順序で繰り返す

→ 機器がパイプの接続順に整列

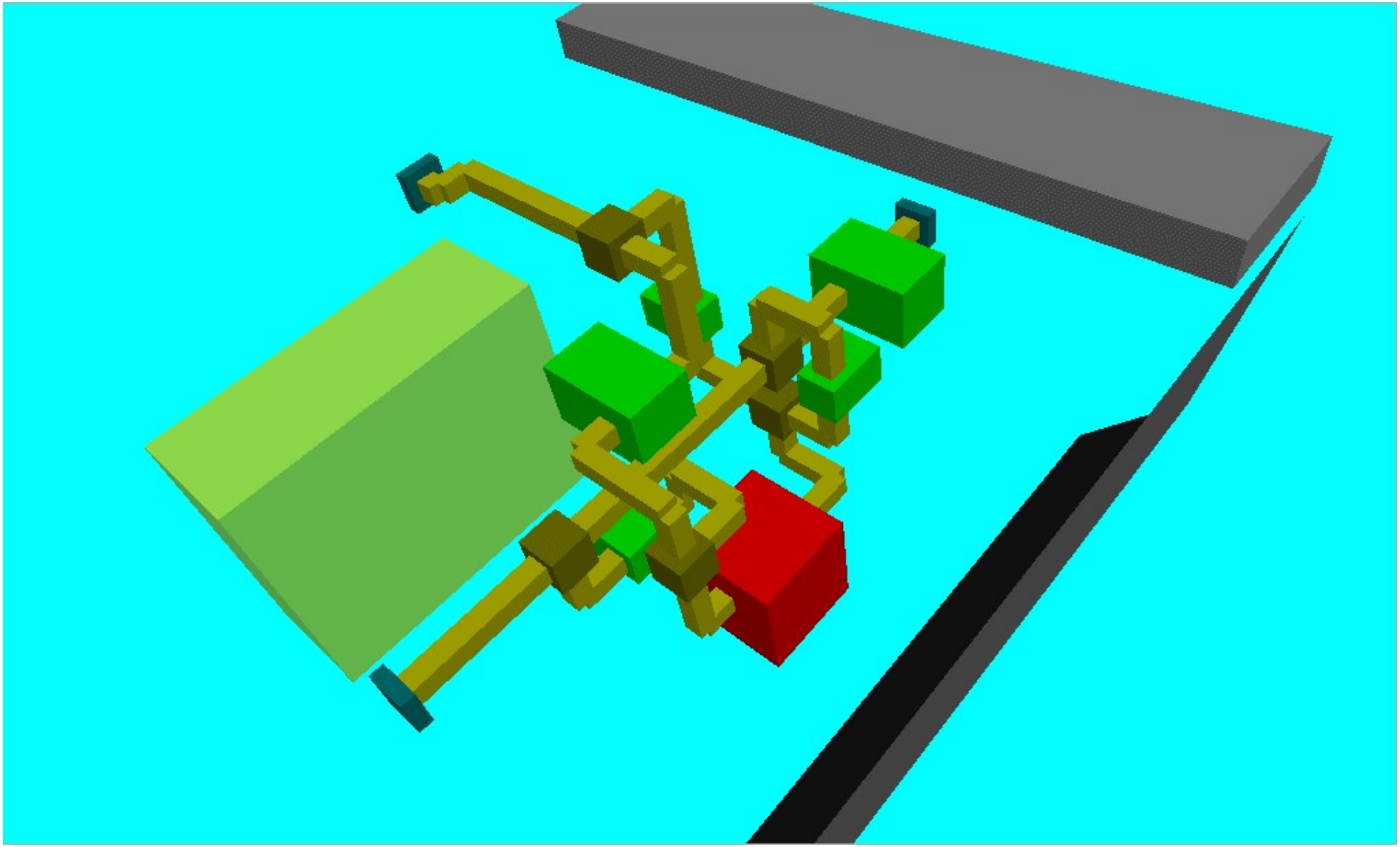
パイプ経路が短くなる効果的な機器配置アルゴリズム：
自己組織化機器配置法の適用例





【特徴】　・操作を行う機器の順序によって異なる解を得る

【問題点】　・パイプ長は短くなるが、「バルブ操作性」などは考慮できない
→　遺伝的アルゴリズムによって改善



Material Cost = 2.52
Number of Elbows = 18
Cost of Valve Operability = 0

【組合せ最適化問題の解法（1）】 まずこれらの方法で試す

●腕ずくの方法（brute force method）:

コンピュータの高速な計算力に頼って、**起こりうる全ての場合を調べて、**
その中で最も良いものを選ぶ方法。（列挙法・**全数探索**とも呼ばれる）
〔特徴〕
◎　原理が単純でわかりやすい
◎　実行が可能であるならば、必ず最適解が見つかる
（組合せ最適化問題の解候補は有限な離散集合なので）
小規模問題では実用的
×　場合の数が組合せ爆発を起こす場合は実行不能

●欲張り法（greedy method）:

各選択の時点で、**目先の利益が最大になるよう選び続ける方法**
〔特徴〕
◎　原理が単純で分かりやすい
×　局所的な目先の最適化を行い続けることが全体の最適化になるとは
一般にはいえない…最適解が見つかる保障は一般にはない
◎　多くの場合、最適ではないがそれに近いものが求まる
◎　グラフ最短経路探索問題など一部の問題では、最適解が得られる
（ダイクストラの方法） また、後述の動的計画法とも関連

【例】 ナップサック問題に対する欲張り法

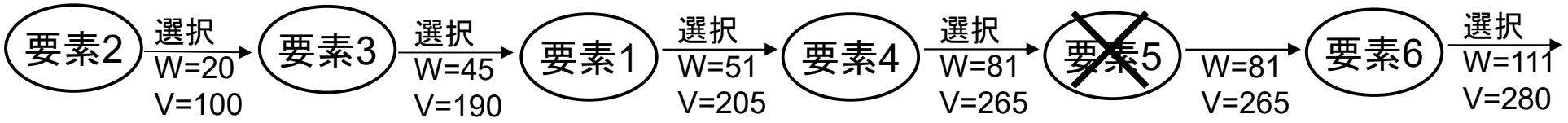
ナップサックの容量： 112

要素	1	2	3	4	5	6	7	8
重さ	6	20	25	30	40	30	60	10
価値	15	100	90	60	40	15	10	3
単位重さあたりの価値	2.5	5	3.6	2	1	0.5	0.166	0.3

（1）要素を単位重さあたりの価値の高い順に並べ替える。 i ＝1 とする。

（2）もし i 番目の要素を選択したら、選択した要素全体の重さがナップサック容量を超えない場合はその i 番目の要素を選択する。さもなければその要素を選択しない。

（3）i が要素数 n と等しくなったら終了。さもなければ i を i+1 として（2）へもどる。



要素数10, 容量100, 価値上限100の問題で欲張り法により最適解を得る割合16～17%　だが最適解と欲張り法の解の違いは1割程度

【組合せ最適化の解法(2)】 前述の方法でダメな場合、次に試す

分枝限定法 (branch and bound method)

発見的手法(ヒューリスティクス)

遺伝的アルゴリズム(メタヒューリスティクス)

次回の講義で説明

まとめ　最適化すべき変数が離散値の場合：整数計画問題・組合せ最適化

【問題の難しさ】

1）P：　決定性アルゴリズムで多項式時間内で解決できる全ての問題。
　経路探索問題：ダイクストラ法
2）NP：　非決定性アルゴリズムで、多項式時間内に解決できる全ての問題。
3）NP困難：NP以上の難しさをもつ問題

【NP完全問題（NP-complete problems）】

有限時間で解けるが、
多項式時間で解けるようなアルゴリズムがない（見つかっていない）問題群

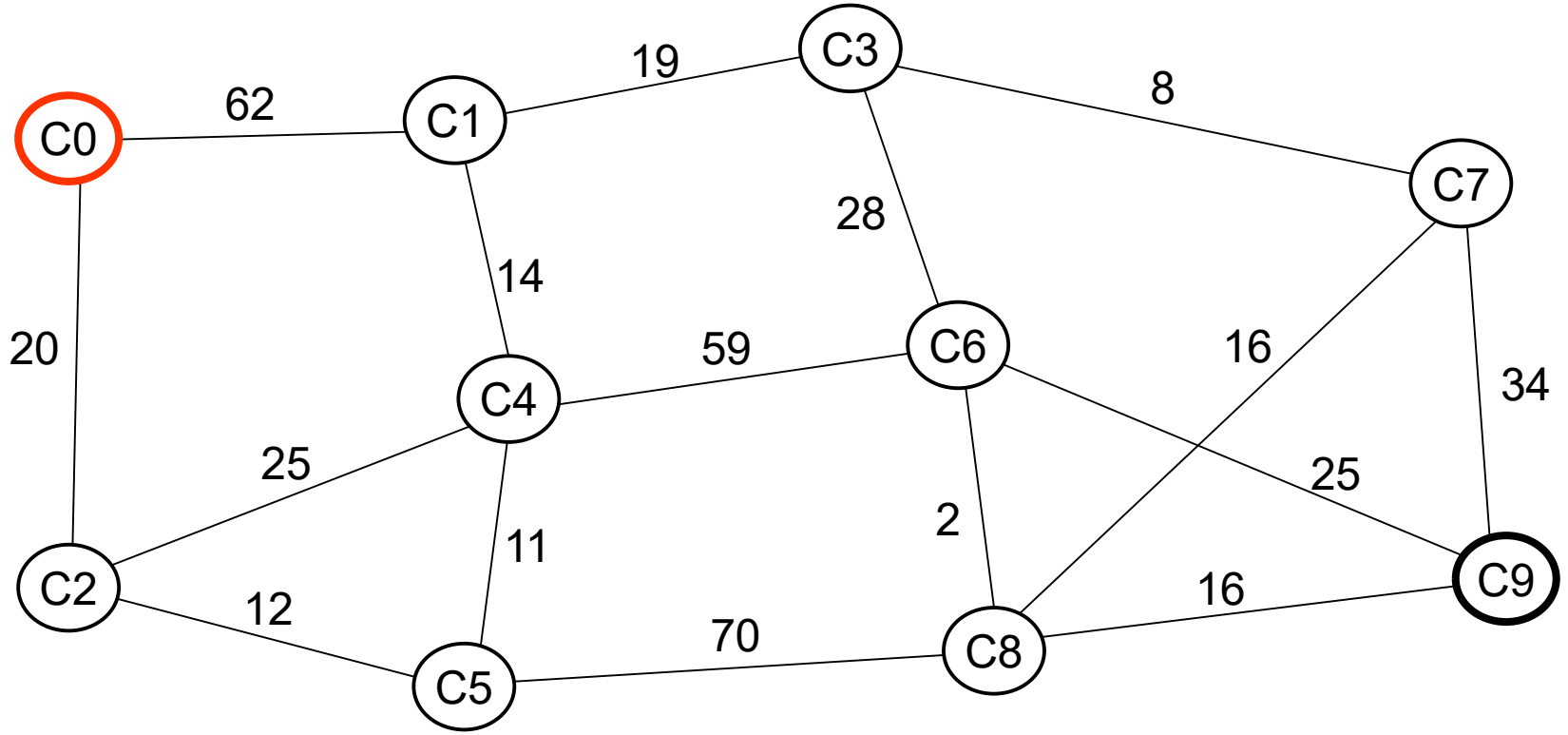
NP完全問題の例：　巡回セールスマン問題（Traveling Salesman Problem: TSP）、
2次割り当て問題（Quadratic Assignment Problem: QAP）
ナップサック問題（knapsack problem）、クリーク問題（creek）
ジョブショップスケジューリング問題（job shop scheduling）
NP完全問題の正準問題：SAT（Boolean satisfiability problem）

【組合せ最適化問題の解法（1）】

まずは腕ずくの方法（全数探索）と欲張り法を試みよ

⇒ ダメなら
次回以降の方法で

【レポート課題】　2020.01.10



上記課題をPythonのグラフ最短経路探索のソルバを利用してプログラムを組み、
実行して結果を手計算と比較して確認せよ。

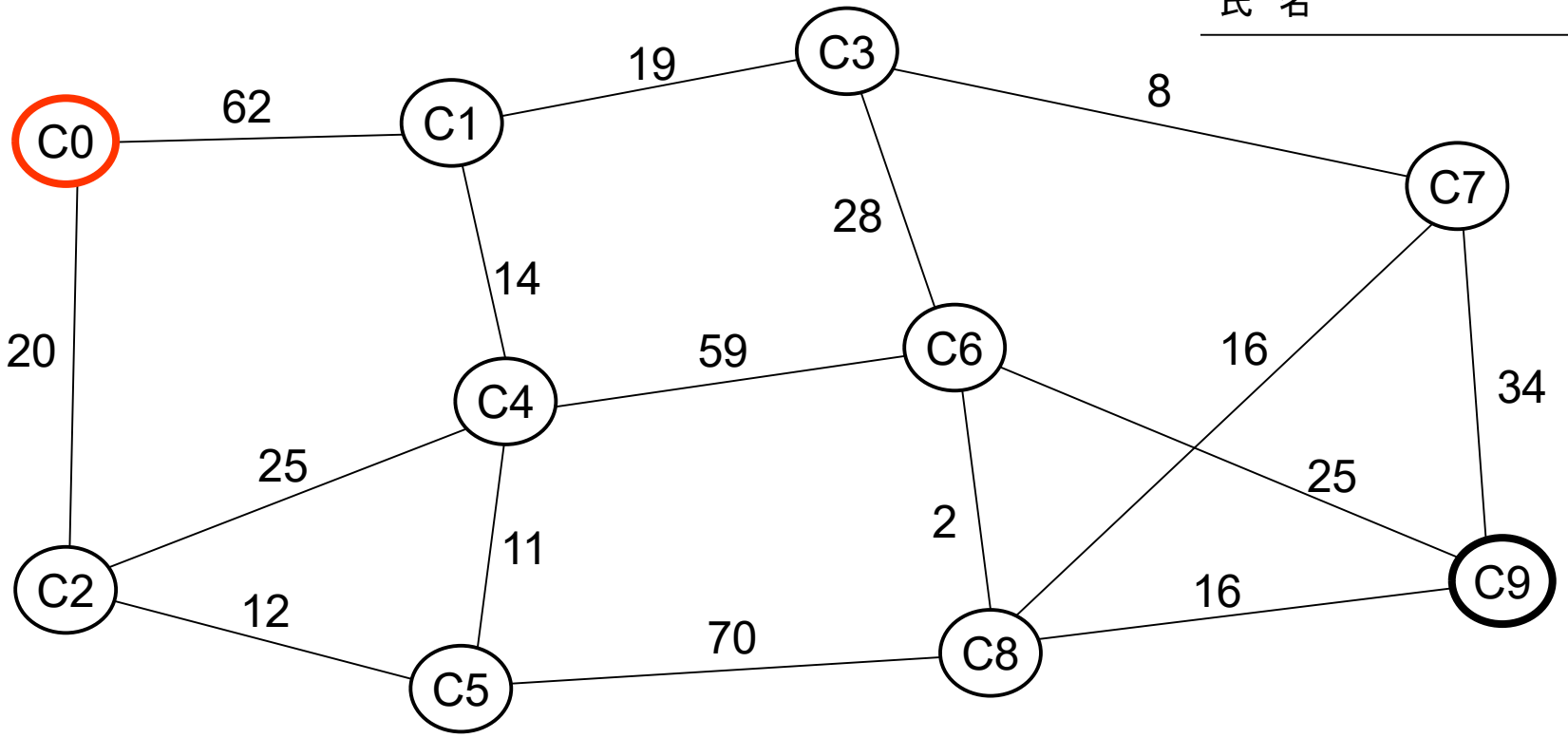
レポート課題提出方法

上記のプログラムを下記の課題提出用フォルダへ、課題の番号と提出者が分かるように
ファイル名を以下のようにしてアップロードせよ（提出期限：1月14日まで）

第6回1TE19xxxZ名前.py
https://share.iii.kyushu-u.ac.jp/public/lRbwAAVITI5A2X4BE45t6TqQIE0UQSQUl5Bap_kZ_sjy

【演習問題】　2020.01.10

学籍番号
氏　名



(1) C0から各都市へ最小コストの経路で移動した場合のコストを
　ダイクストラ法で計算し、計算過程を上の図中に記入せよ。

(2) 上の計算結果からC0からC9への最短経路を示せ。

(3) C0からの最適経路が複数存在する都市はどれか？またそのときの経路をすべて示せ。