九州大学 海洋システム工学専攻

システム設計特論 （木村）
講義資料

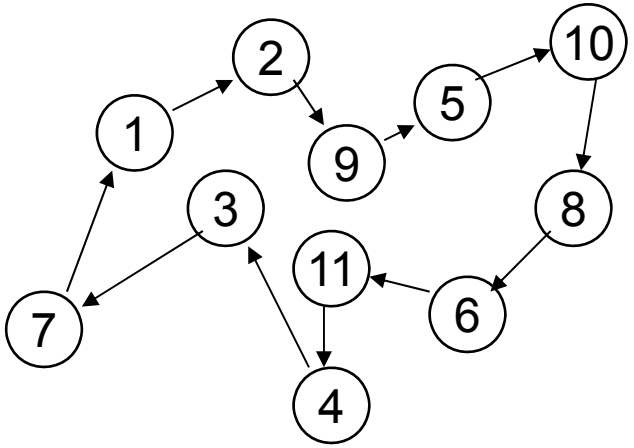
(6) 組合せ最適化・整数計画問題

授業の資料等は

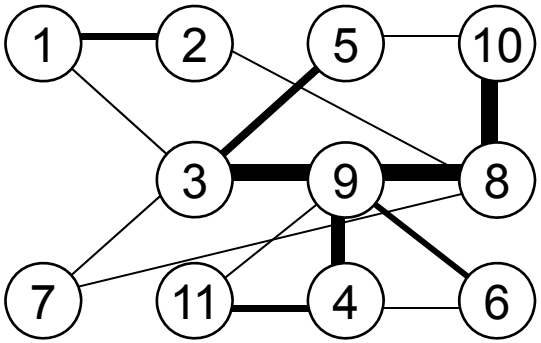
http://sysplan.nams.kyushu-u.ac.jp/gen/index.html

【組合せ最適化問題】

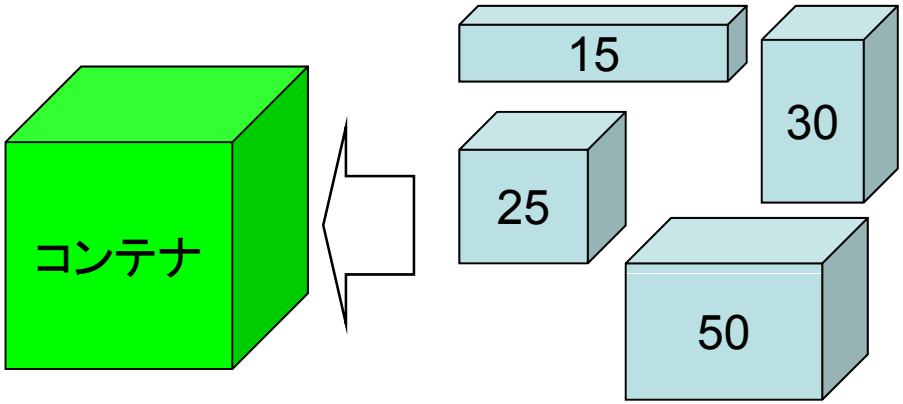
例1： 巡回セールスマン問題（TSP）
全都市を最小コストで巡回する経路は？



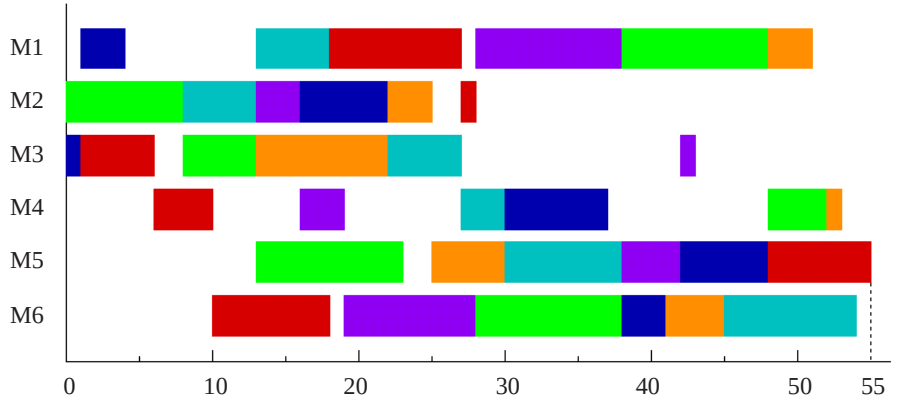
例2： 2次割当配置問題（QAP）
要素間の物流コストとグラフ構造はgiven:
コストが最小になる要素配置は？



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：

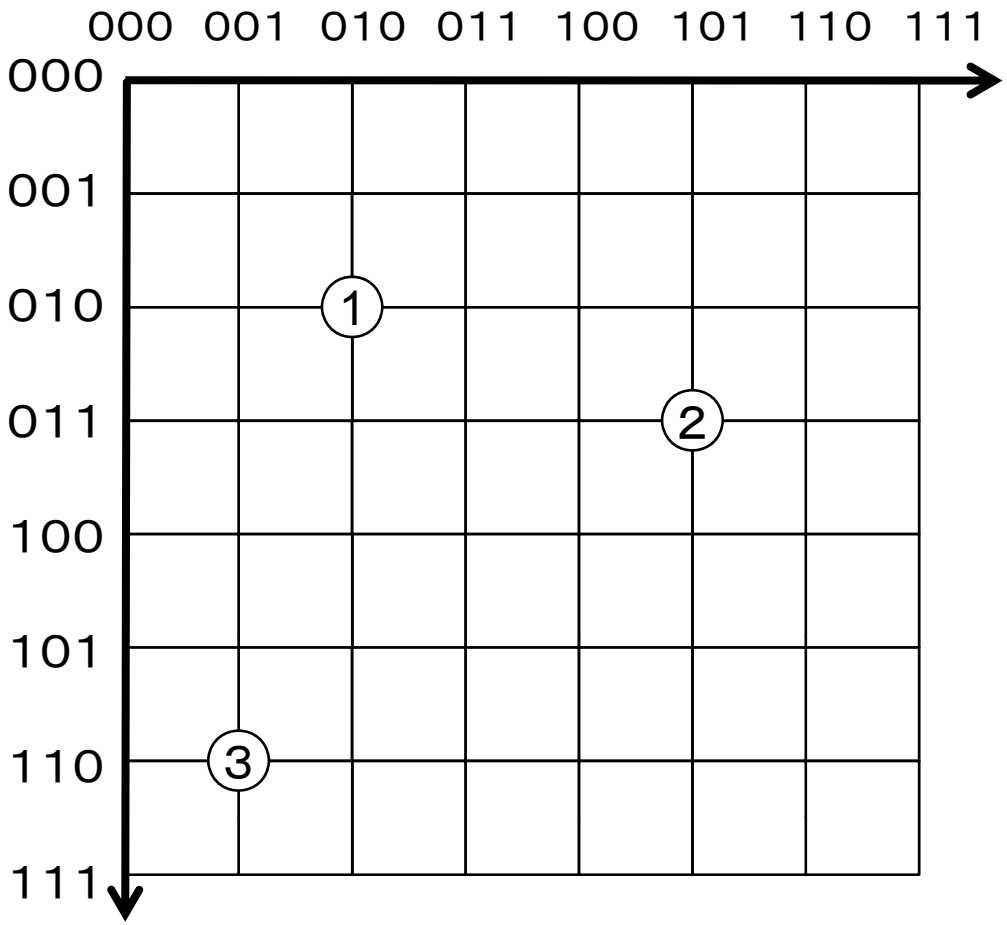
多項式時間問題

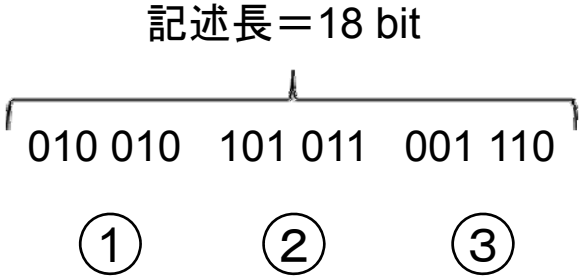
1）P： 決定性アルゴリズムで多項式時間内で解決できる全ての問題。
2）NP： 非決定性アルゴリズムで、多項式時間内に解決できる全ての問題。
3）NP困難：NP以上の難しさをもつ問題

【補足説明】 問題の記述に要するビット長の計算例

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、





【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

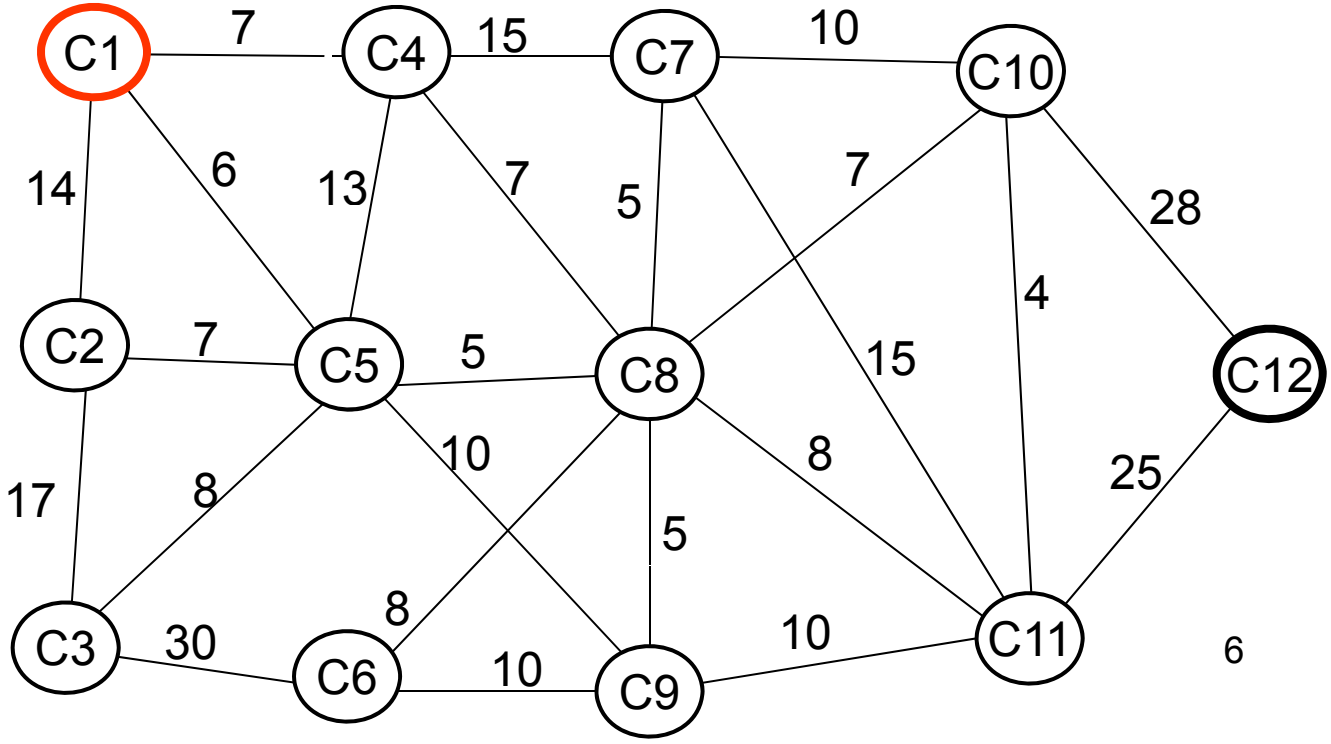
例1: 重み付きグラフにおける2点間の最短路探索問題 → ダイクストラの方法 $O(n^2)$

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

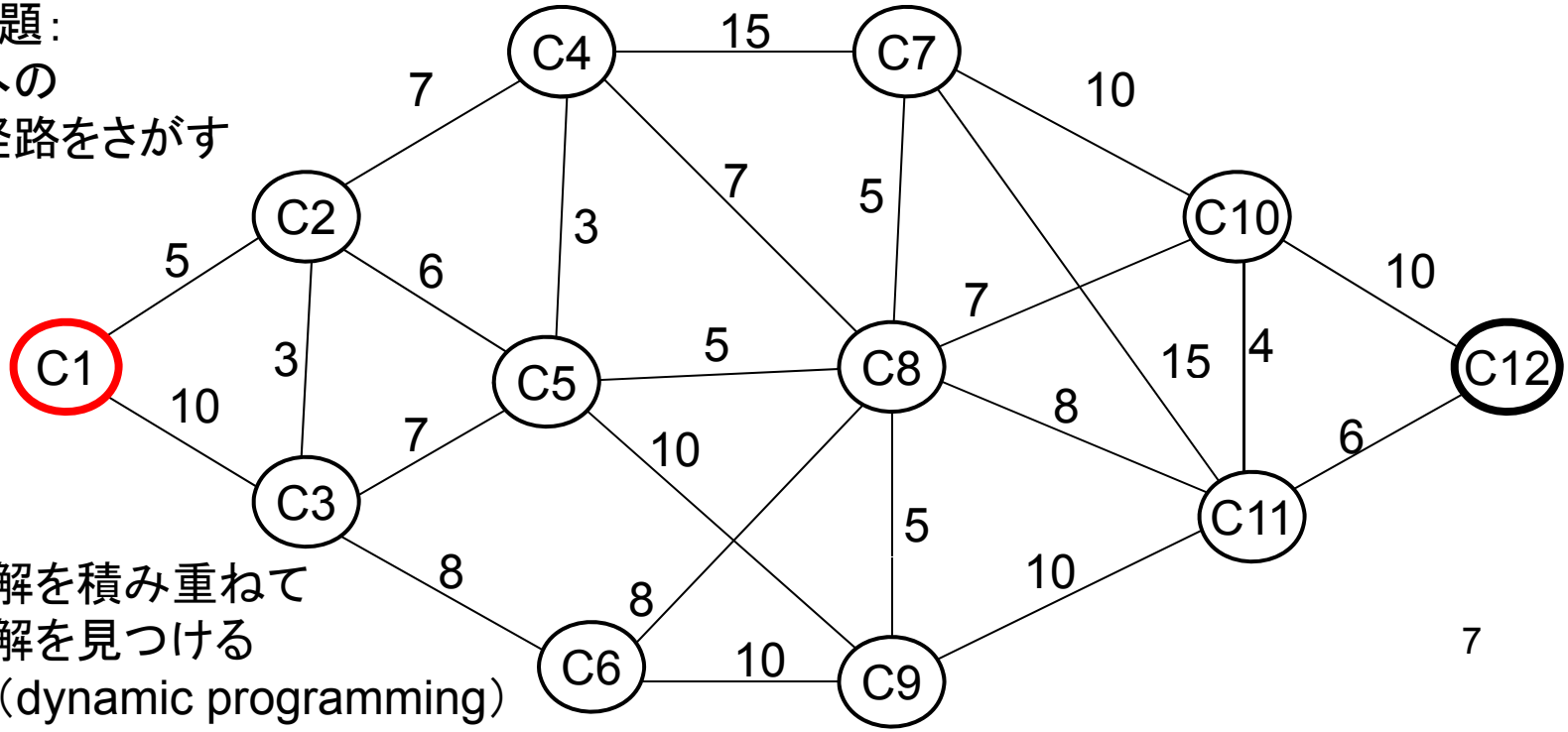


【最短経路探索：ダイクストラの方法】

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

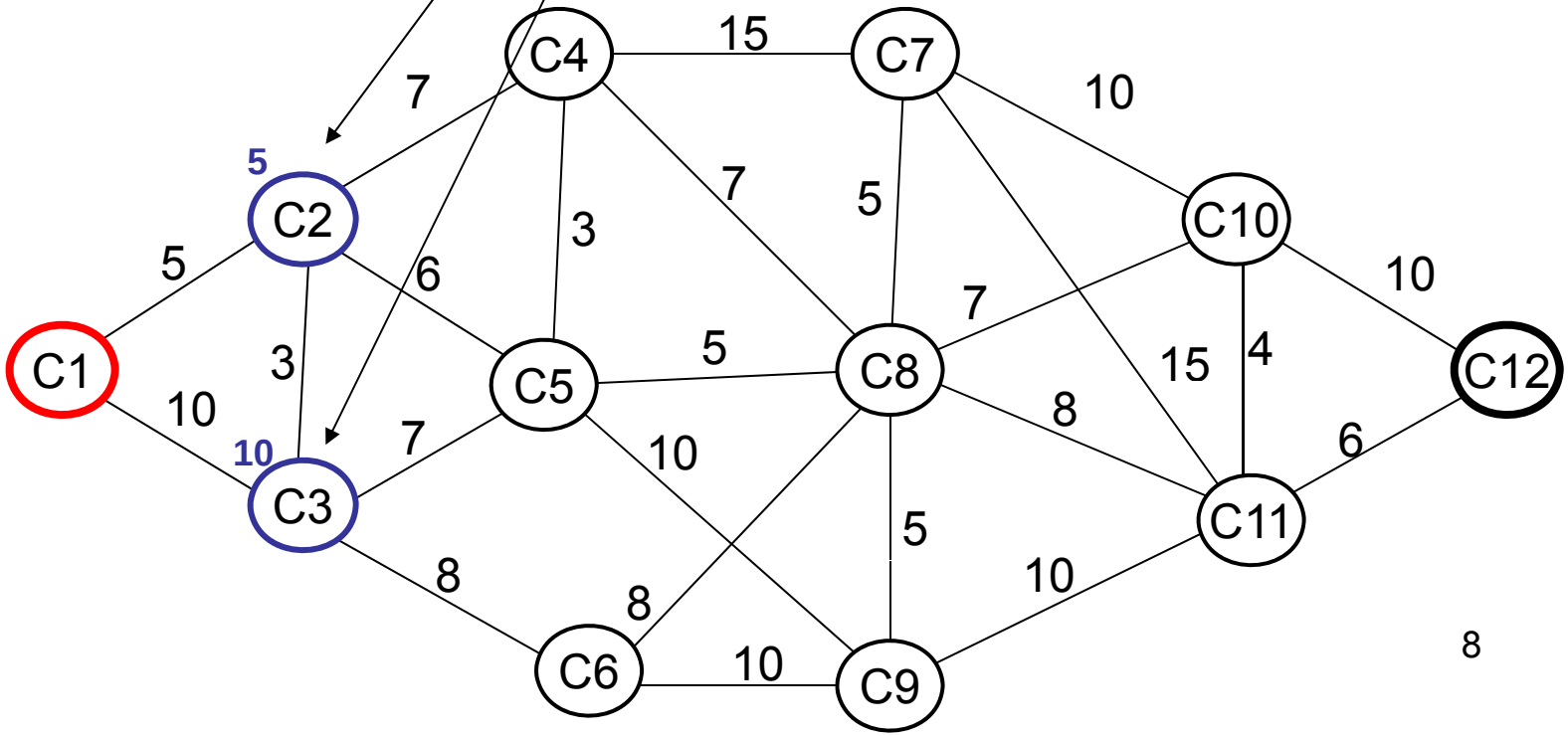


部分的な最適解を積み重ねて
全体的な最適解を見つける
→動的計画法（dynamic programming）

【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

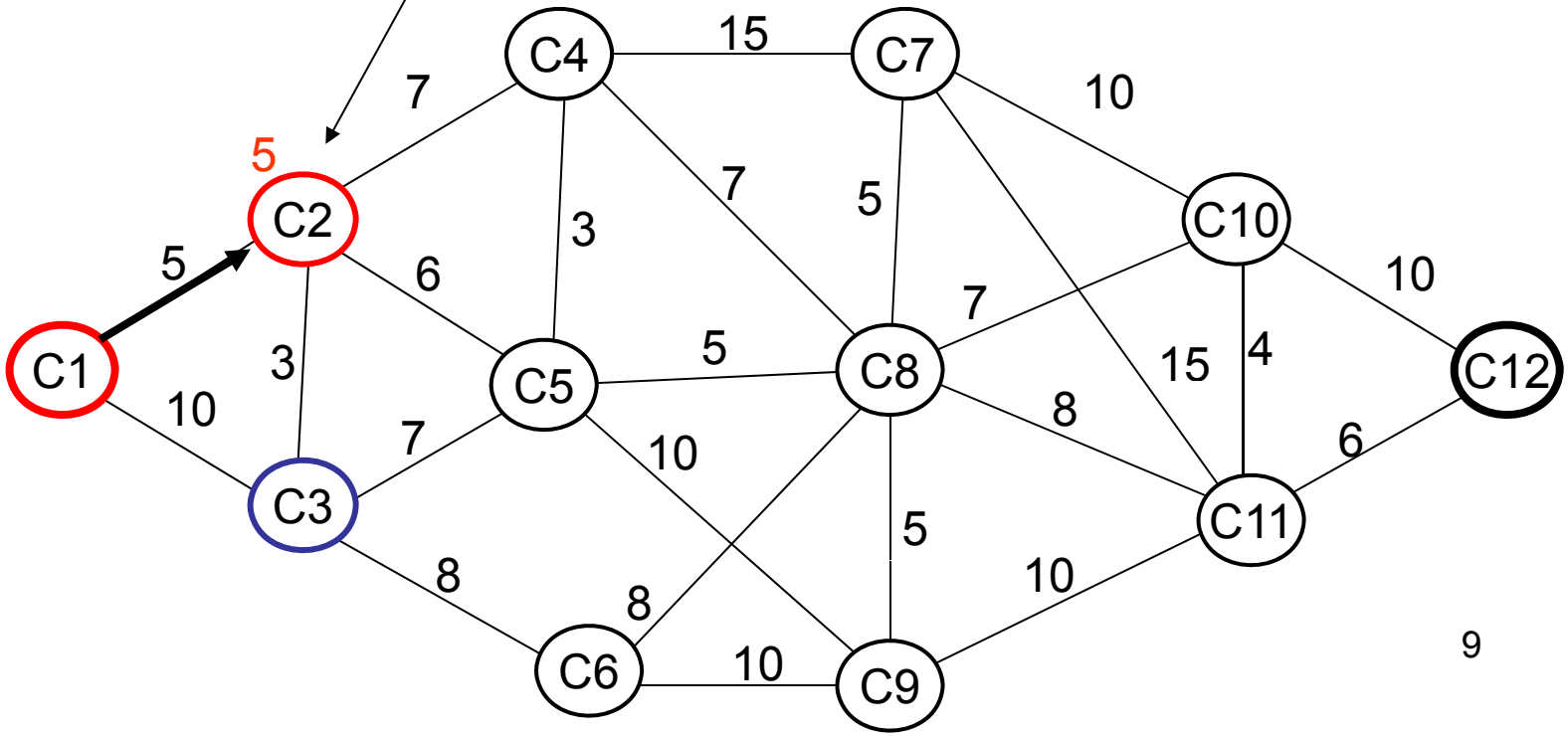
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

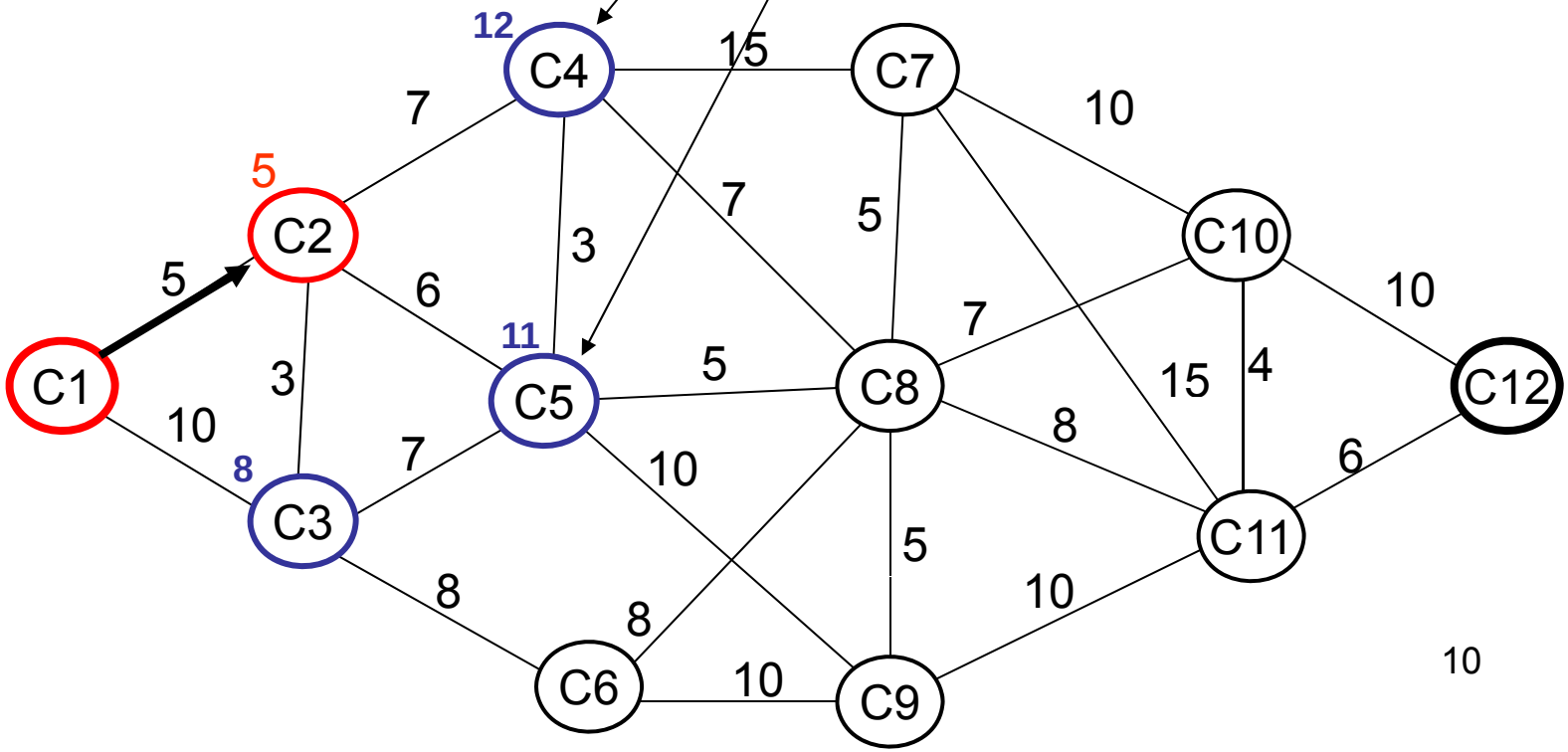
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

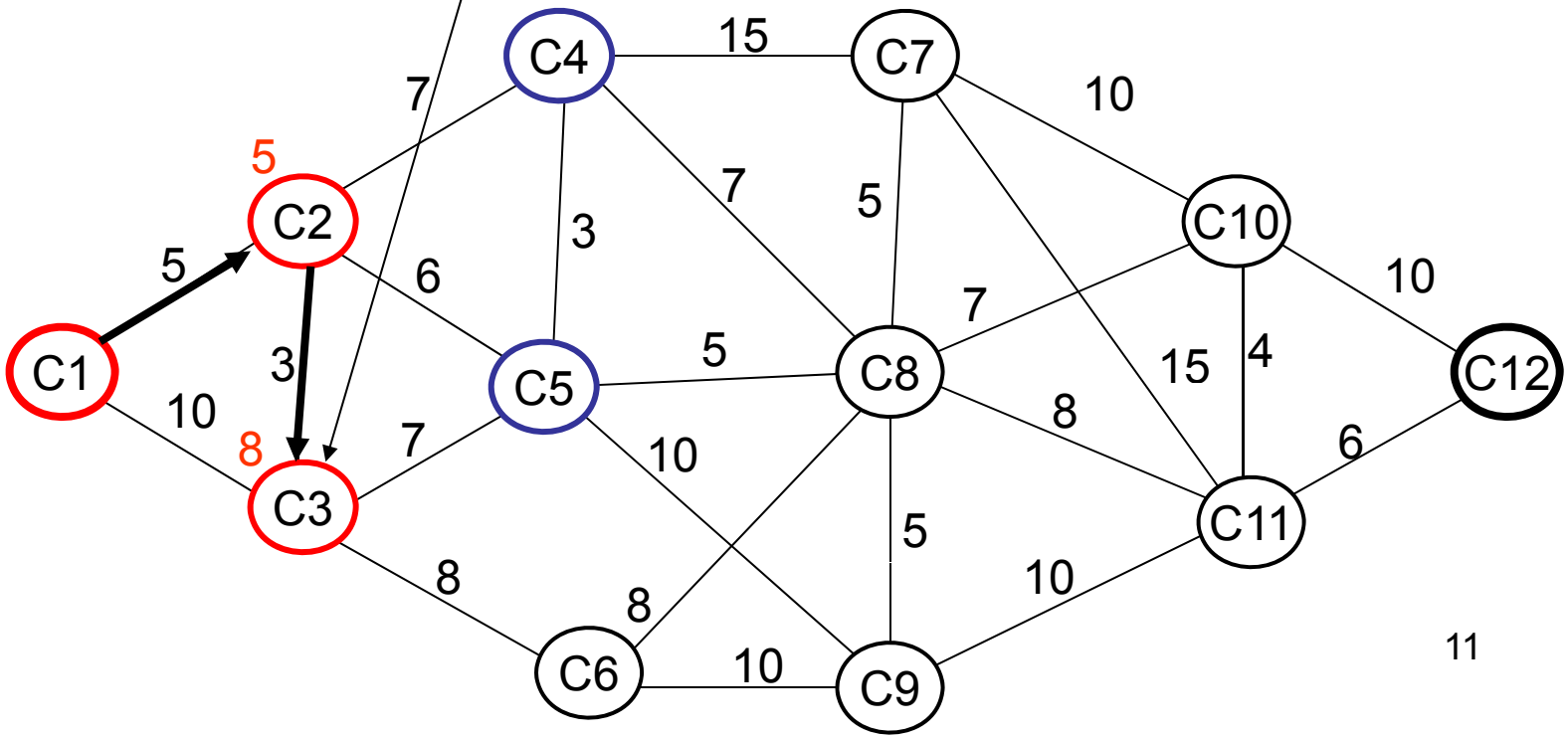
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

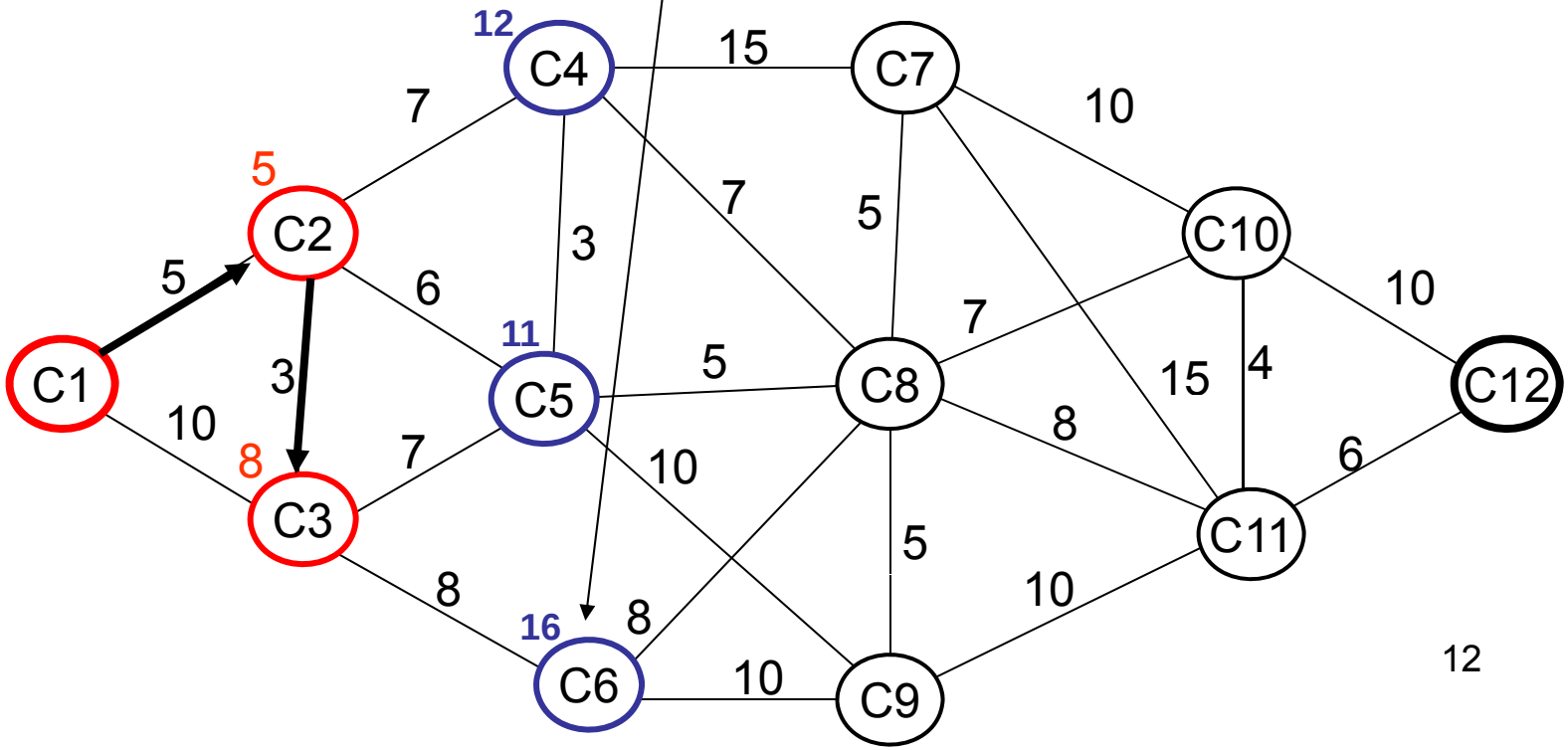
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

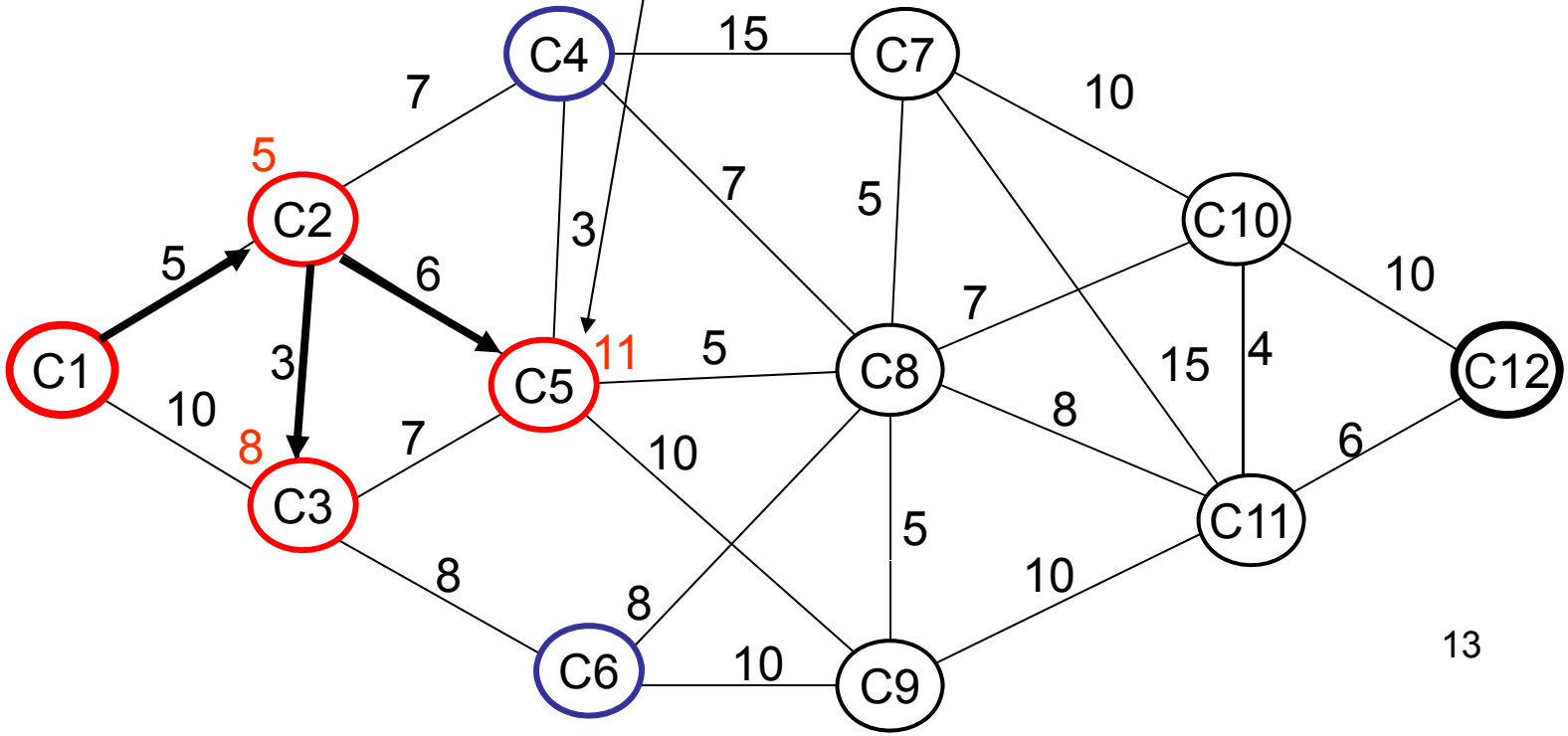
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

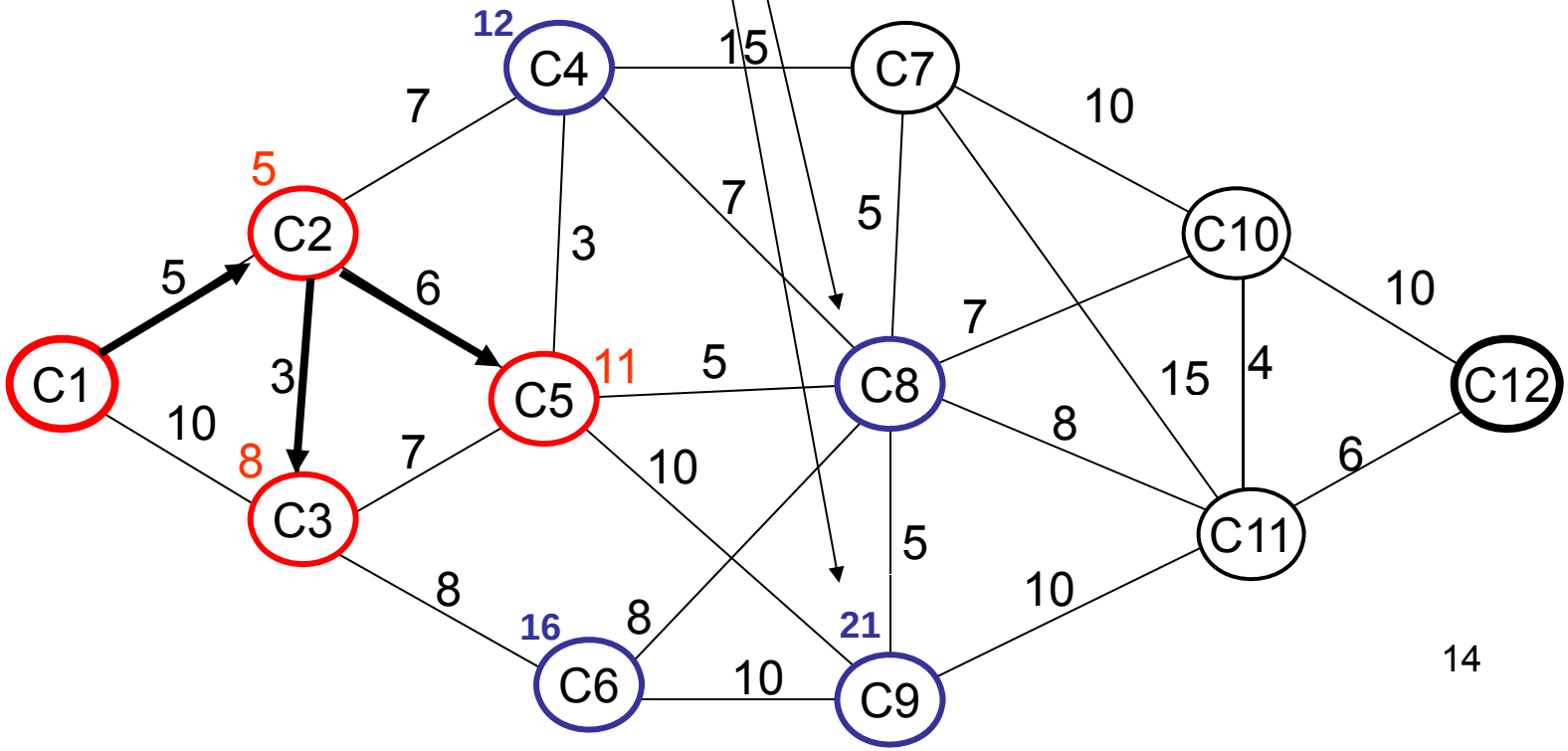
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

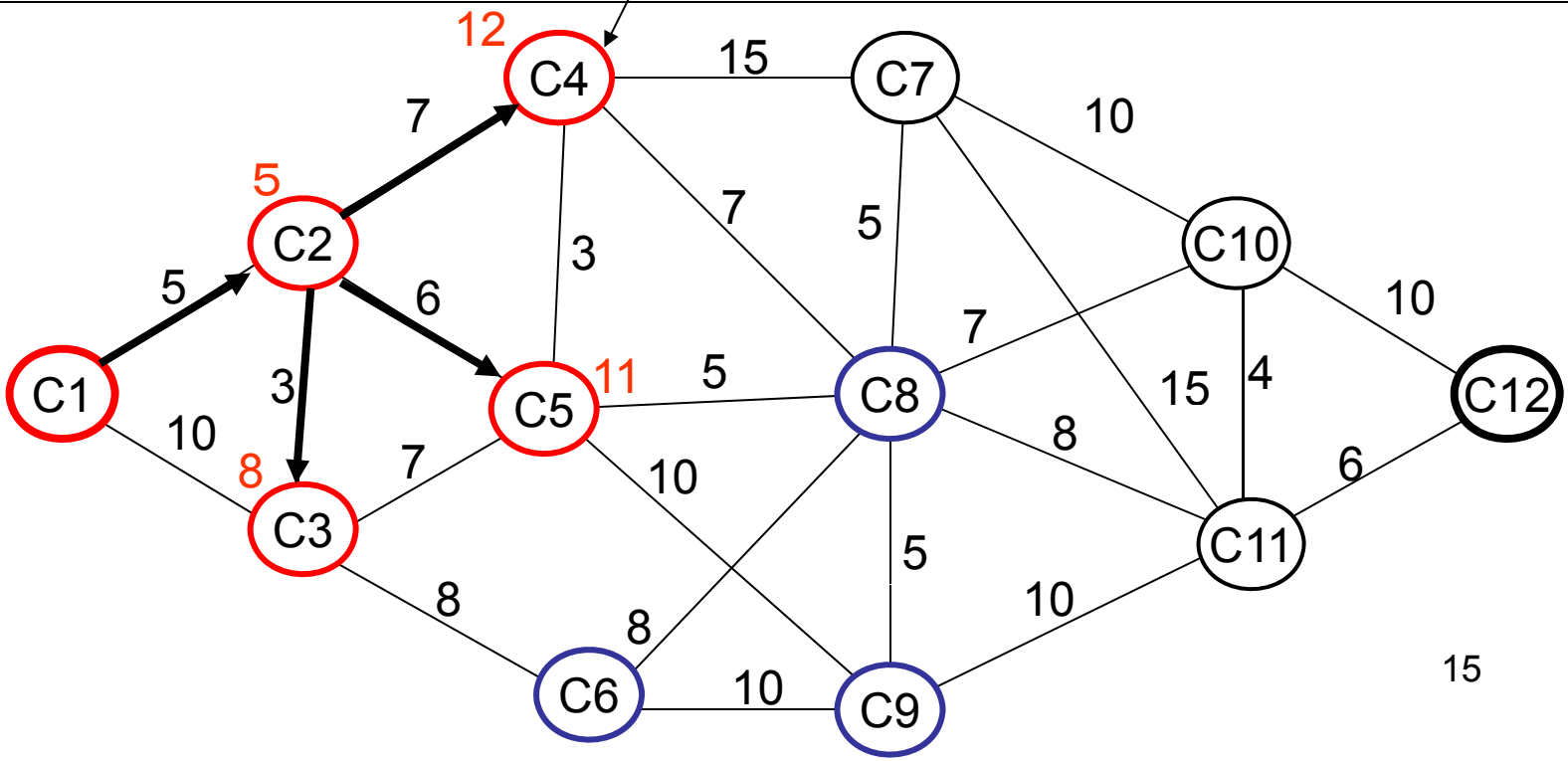
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

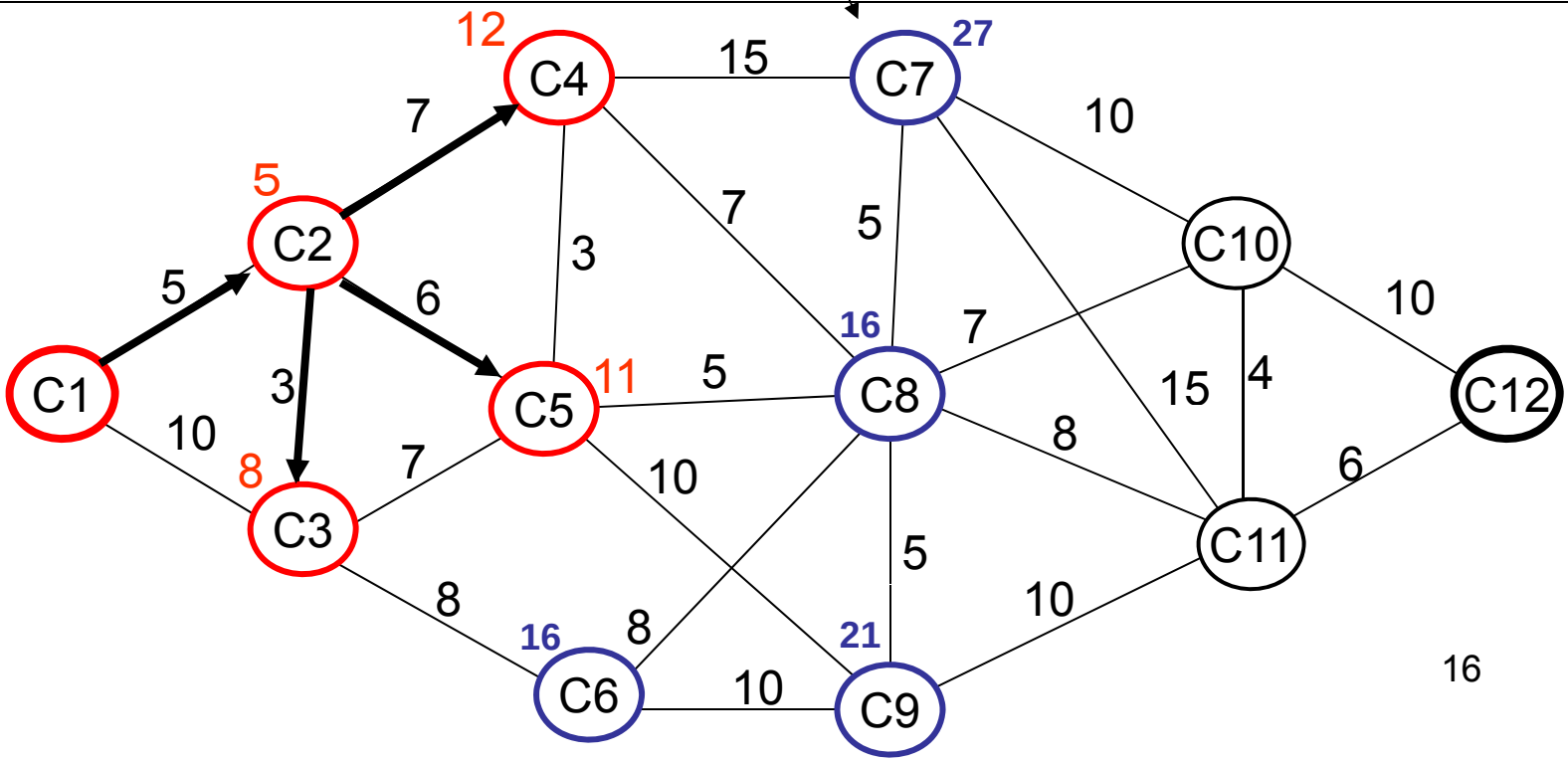
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

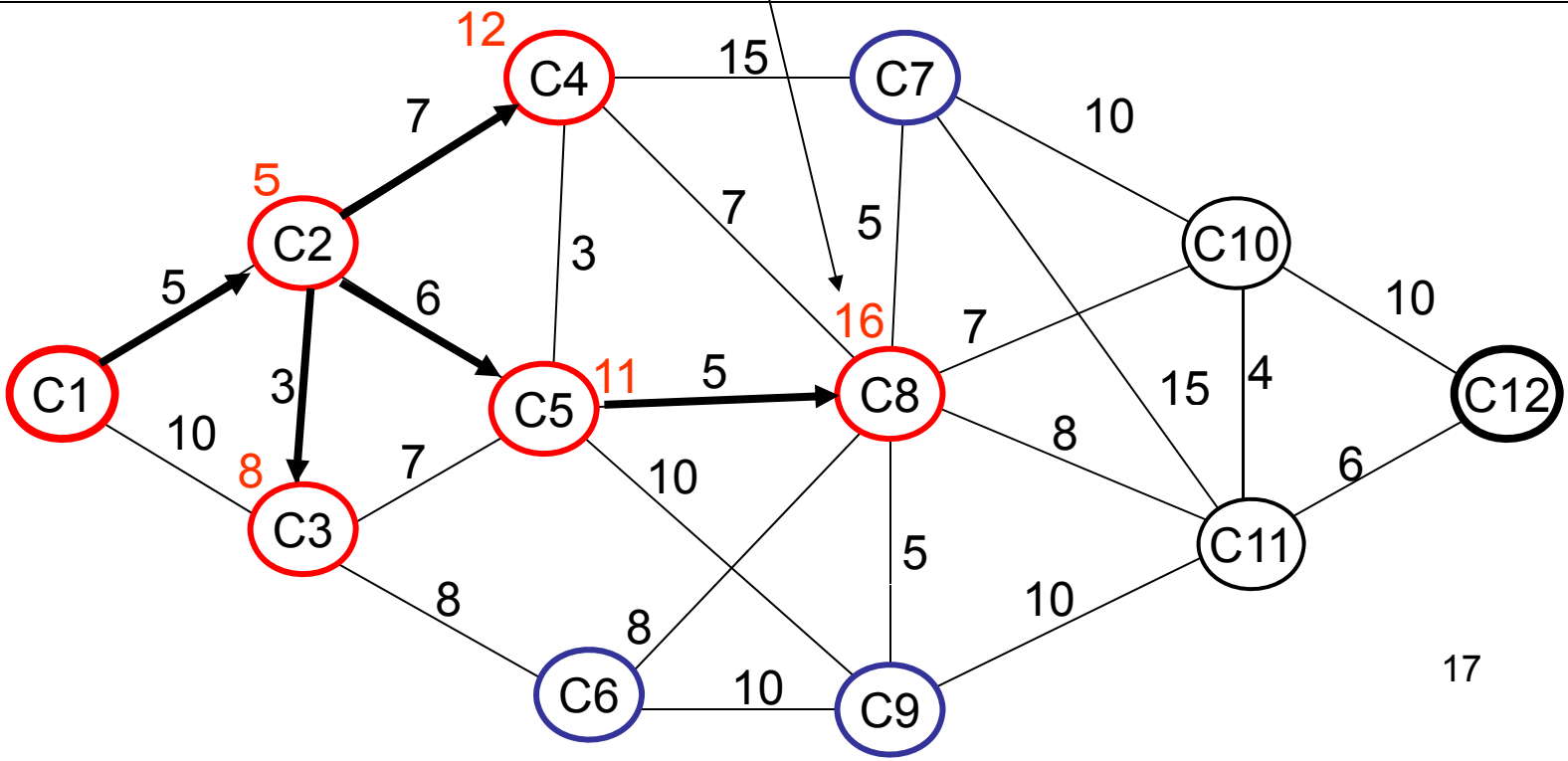
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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

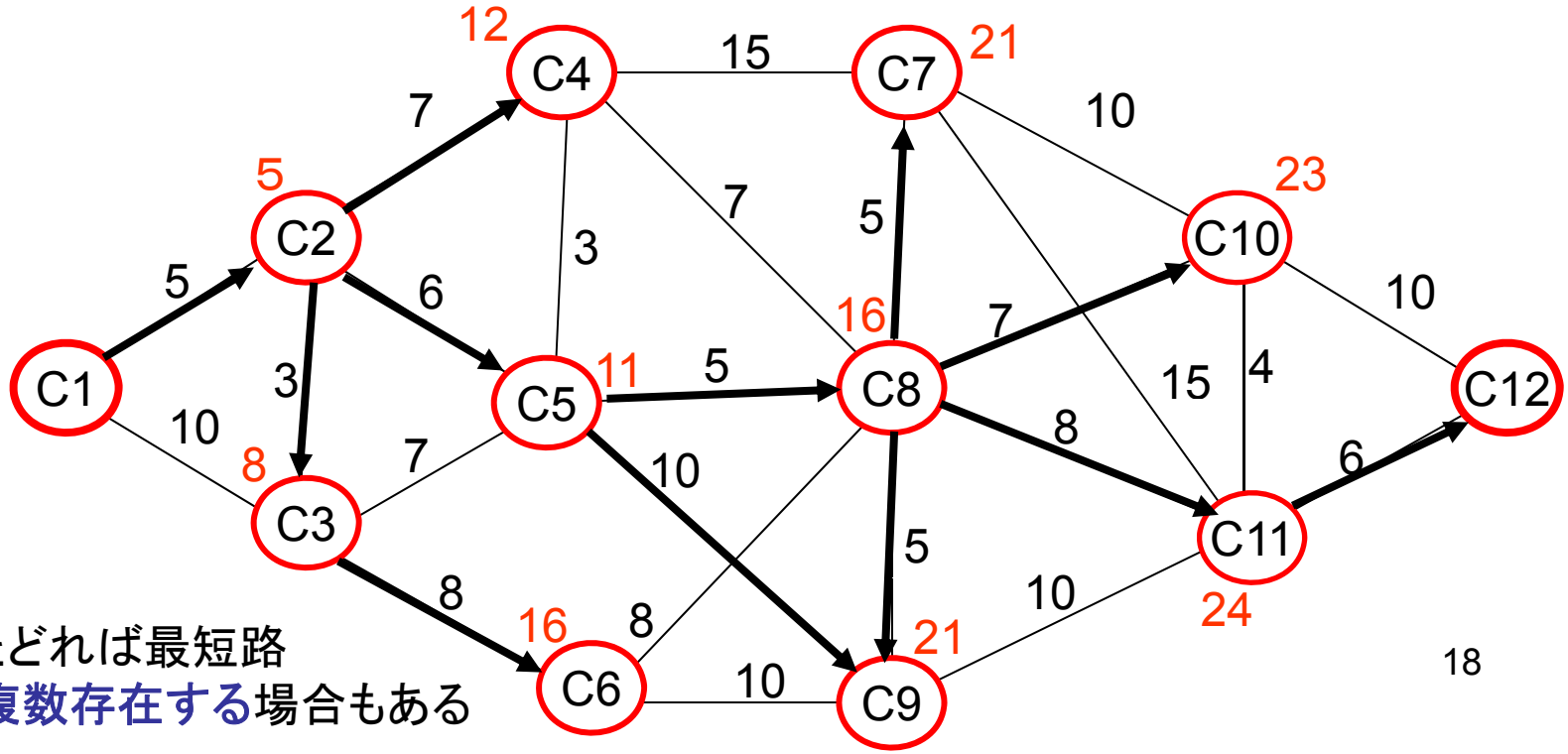
2） Fに属する節点から、始点からの距離が最短のものを見つけて集合Tに加える
   このとき最短コストと、その節点へ行くエッジを記録しておく
3） Tの中に目的地の節点が含まれるまで繰り返してTを増やしていく



【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



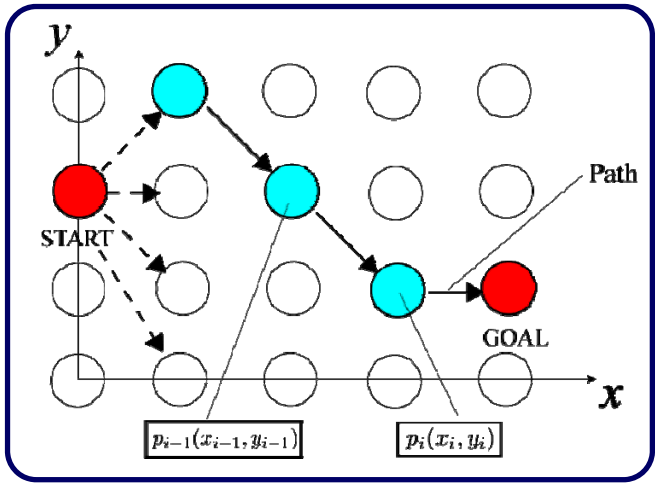
・矢印を逆にたどれば最短路
・**最短経路は複数存在する**場合もある

AUVの経路計画問題の定式化

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

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

AUVの経路計画問題

先行研究：多段決定問題としてモデル化

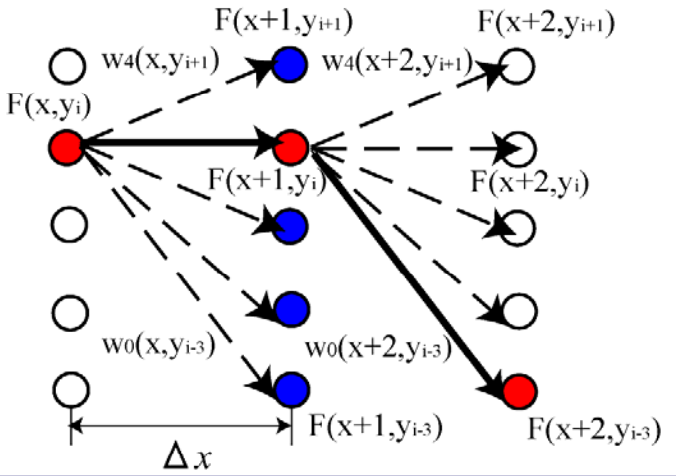
- 経路の制約

  1stepでx軸方向にΔx進むと設定

  ⇒ 後戻りする経路を探索できない

- 探索方法

  動的計画法を用いて最適解を探索



本研究：最短経路問題としてモデル化

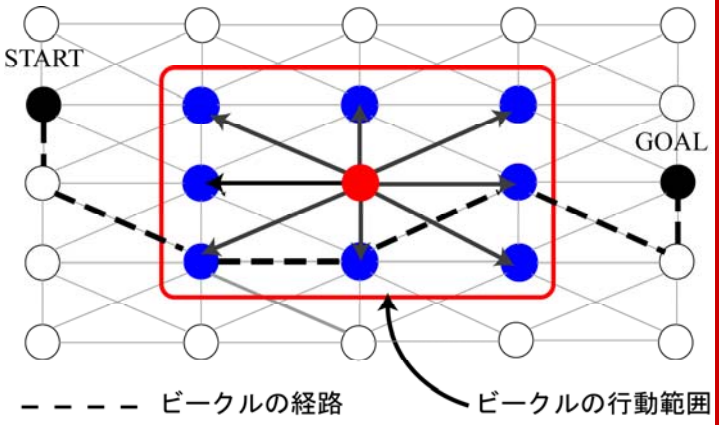
- 経路の制約

  現在点に隣接する8点に進むと設定

  ⇒ 後戻りする経路も含む多様な経路設計を行うことが可能

- 探索方法

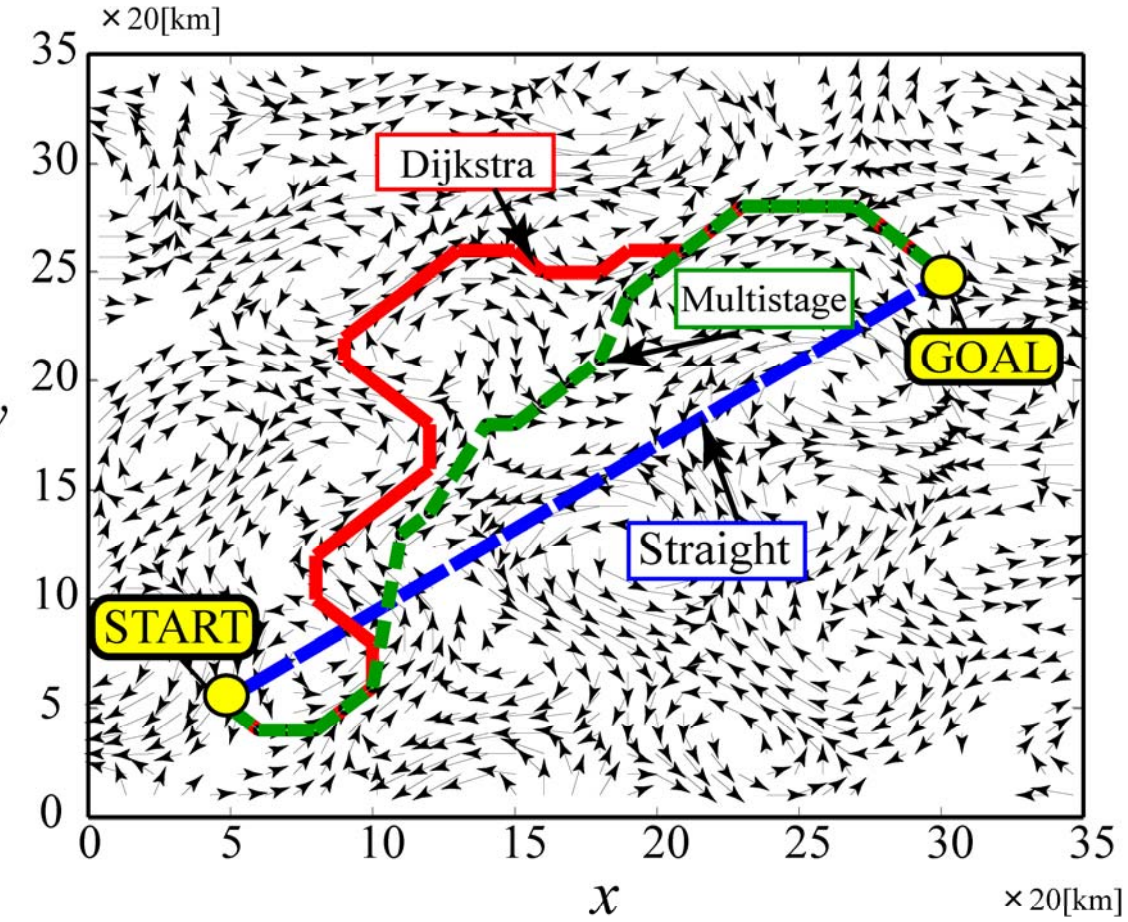
  Dijkstra法によって最適解を探索



（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
  (多段決定問題,動的計画法)
- Straight
  (直進した場合)



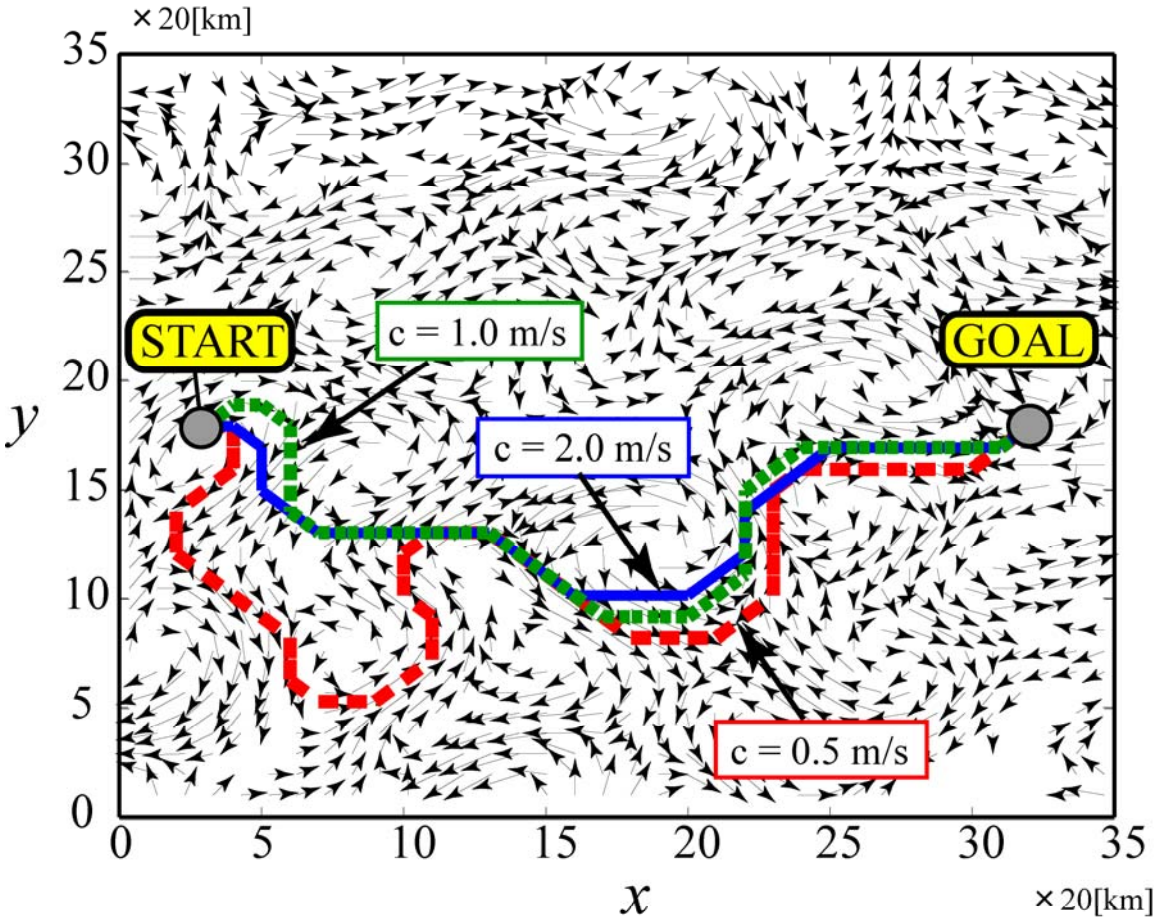
	Cost function $[J]$	Computation time $[ms]$
Straight	42531	13708
Multistage	1553	17219
Dijkstra	965	55360

（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

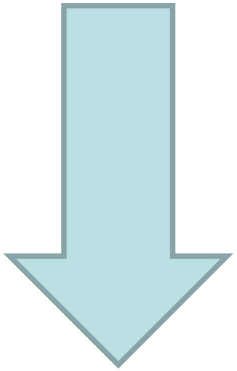


	Cost function $[J]$	Computation time $[ms]$
c = 0.5 m/s	2725	102484
c = 1.0 m/s	13714	58453
c = 2.0 m/s	87007	41488

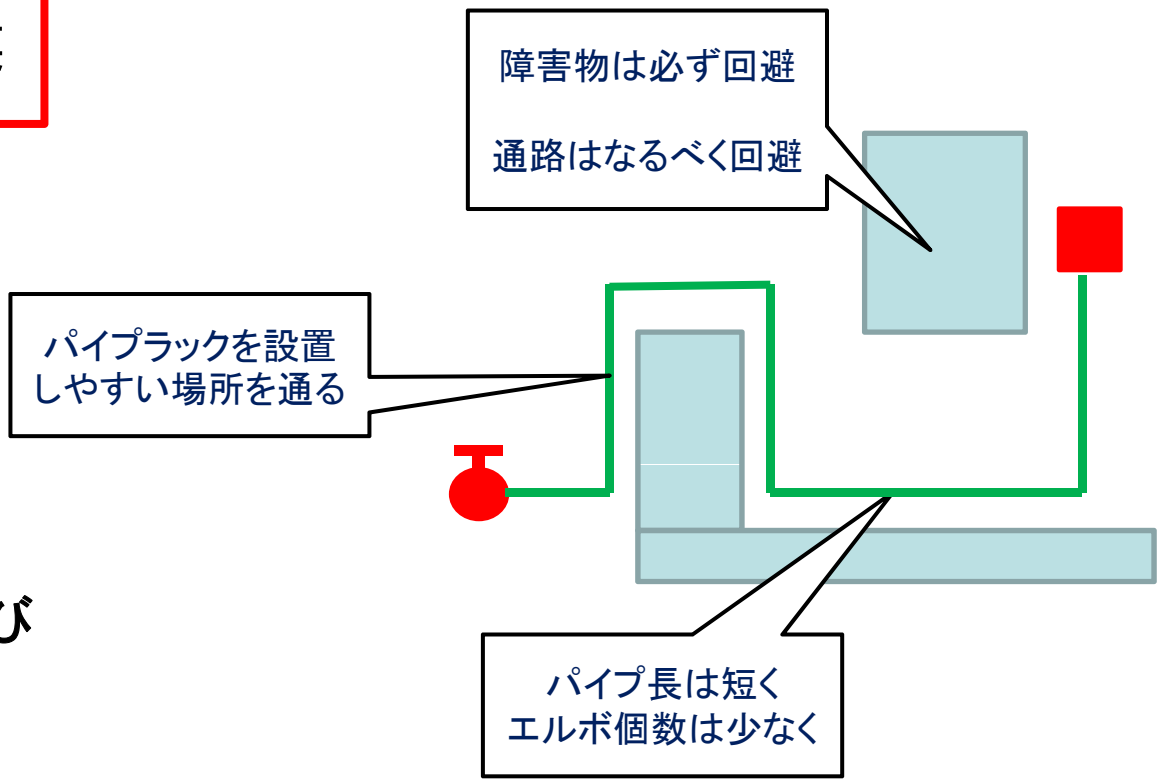
【ダイクストラ法による配管設計】

・知識と経験を必要とする配管設計作業の自動化・省力化

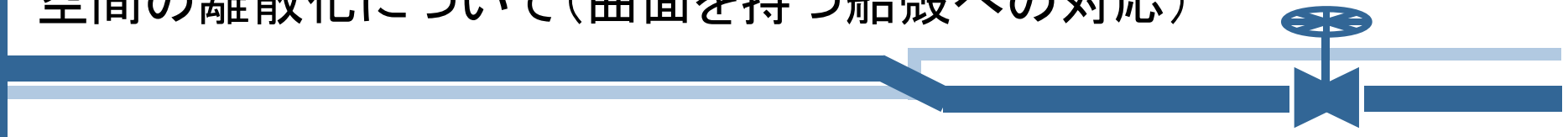
2つの機器間を結ぶ
パイプルーティング作業

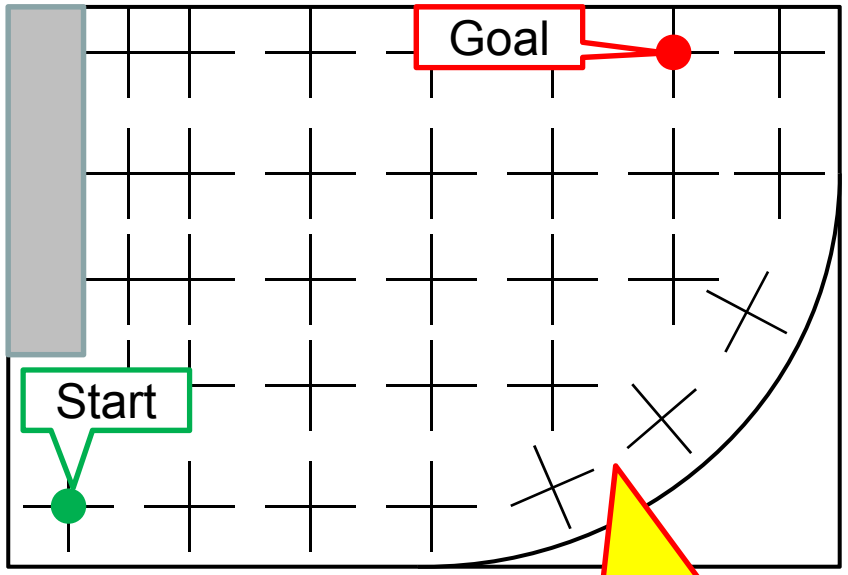


障害物の個数や形状、および
パイプ直径に左右されない
強力なアルゴリズムを構築

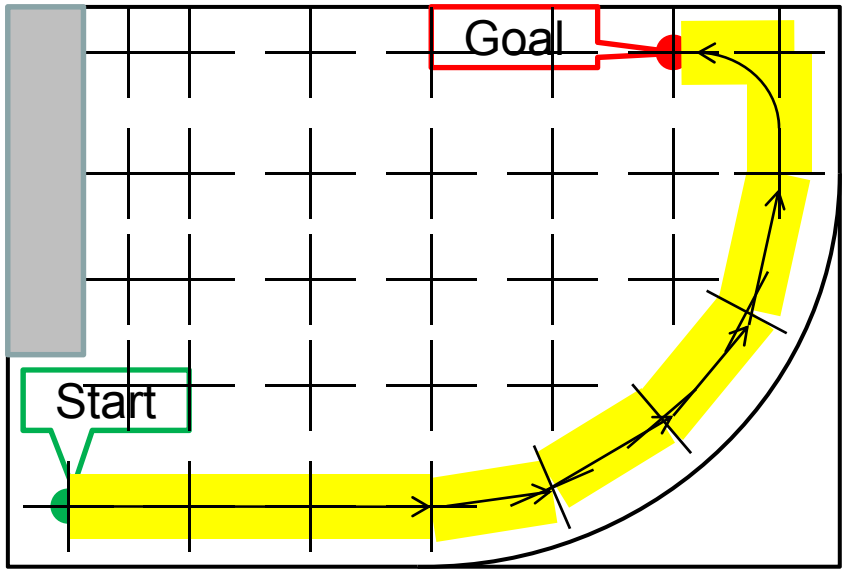


空間の離散化について（曲面を持つ船殻への対応）



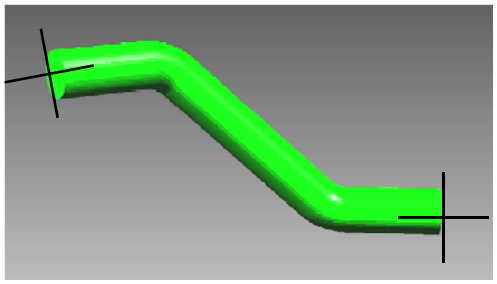


ローカル座標系を
持つ**基準点**を配置

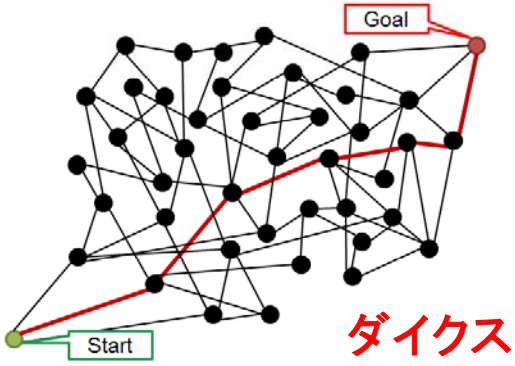


パイプ配置

隣接する基準点同士を各座標軸方向に
ベンド2個以下のパイプで繋がるかどうか判定

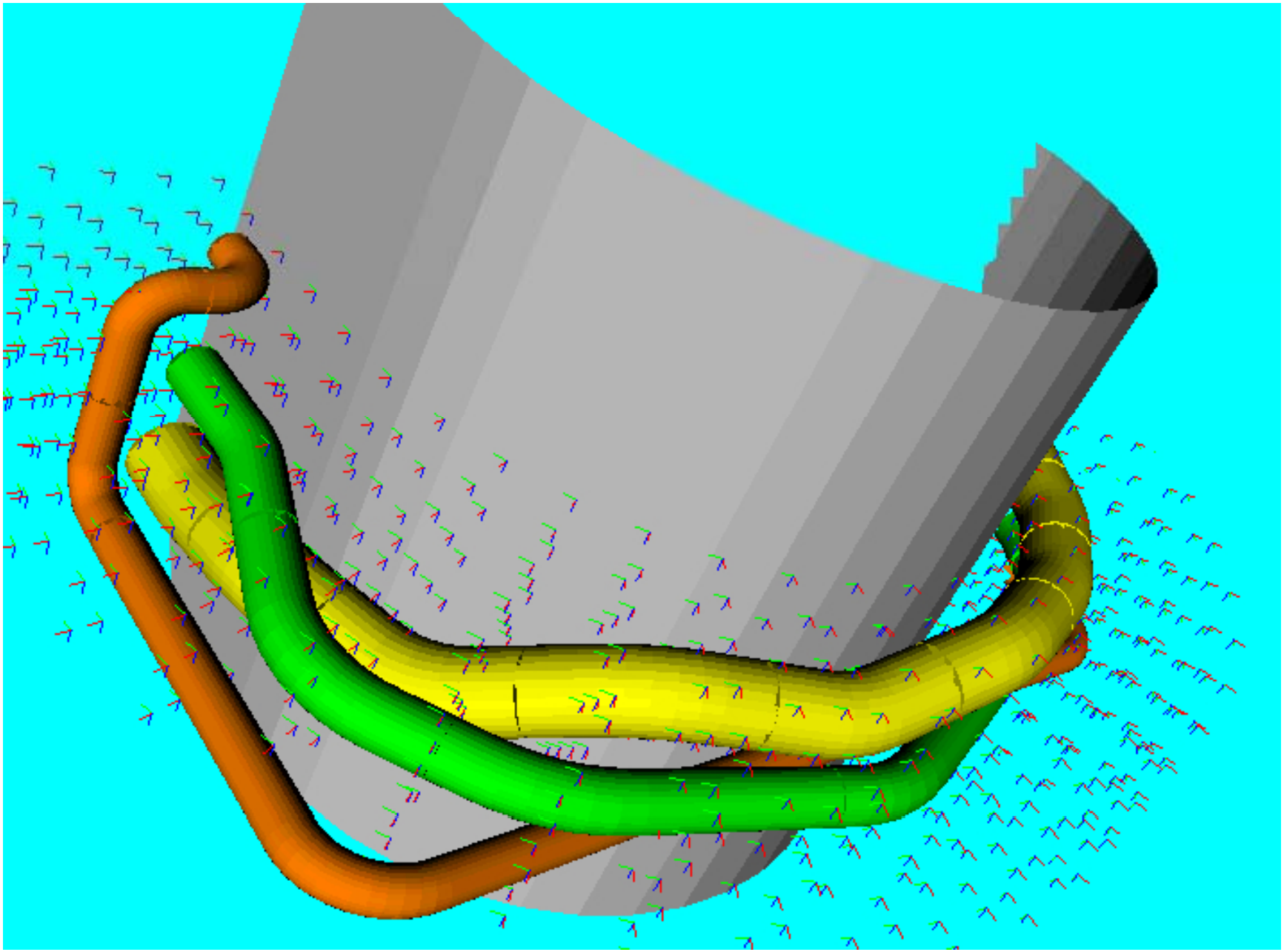


グラフ化



ダイクストラ法で
最短経路探索

ダイクストラ法によるパイプ経路探索結果



【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）**

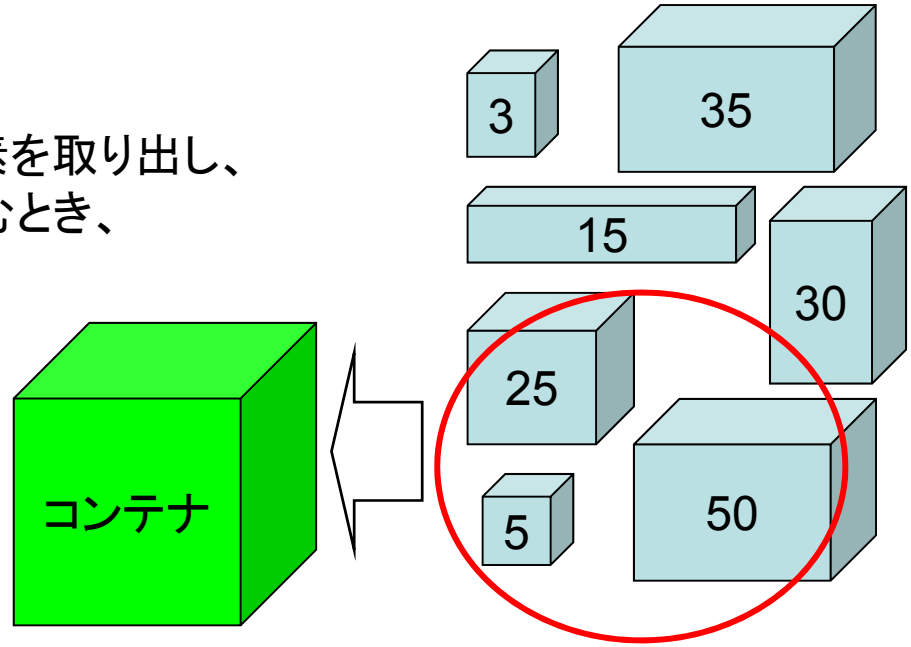
クックの定理（Cooke's Theorem）：
全てのNP完全問題は多項式時間内にSATへ変換できる

●**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 $\sum_{i=1}^{n} x_i \, w_i \leq W$

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

【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

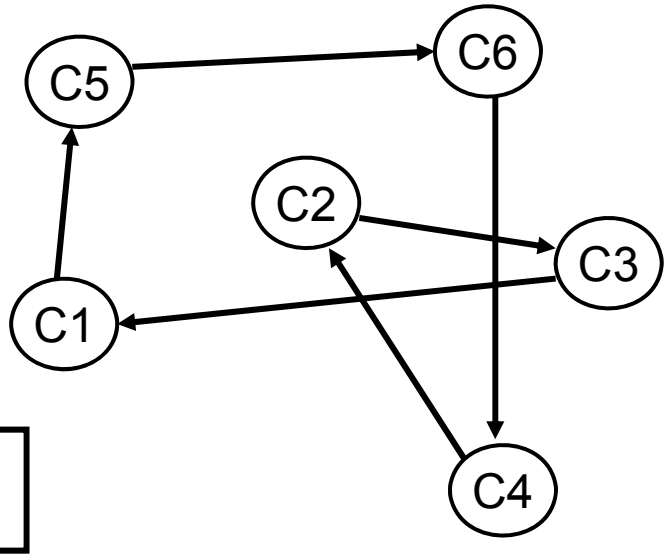
TSPのベンチマークを集めたサイト：TSPLIB
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

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

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

（i：行、j：列）

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}

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

（i：行、j：列）

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

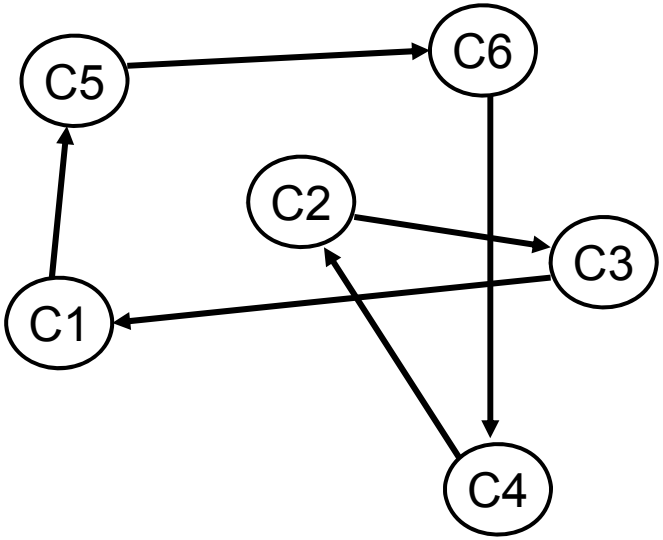
$\sum_{j=1}^{n} x_{ij}=1$　町 i から出る矢印は一つ

$x_{ii}=0$　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

【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

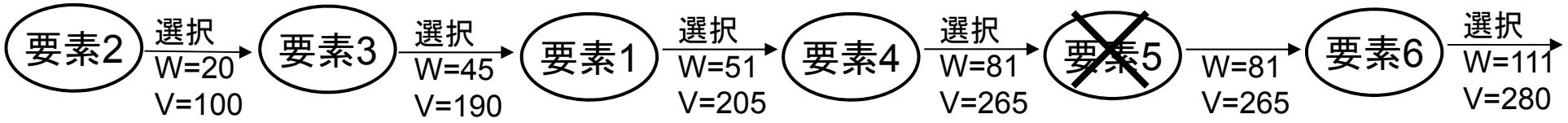
ナップサックの容量： 112

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。

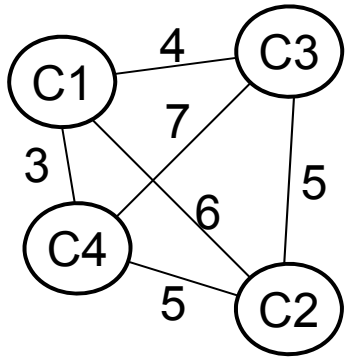


要素数10, 容量100, 価値上限100の問題で欲張り法により最適解を得る割合16～17%　だが最適解と欲張り法の解の違いは1割程度

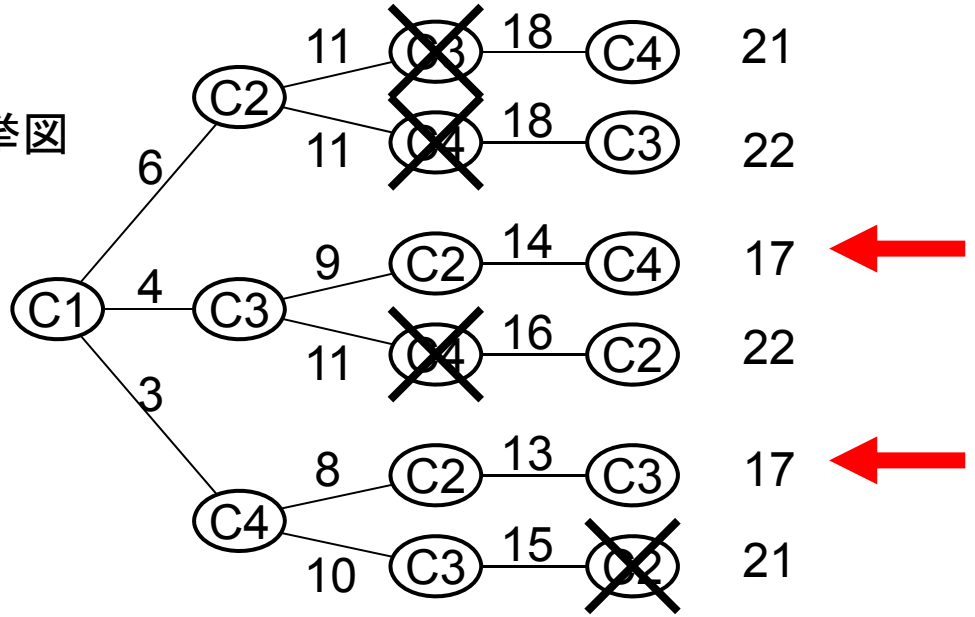
【組合せ最適化の解法(2)】　分枝限定法

全ての解候補を列挙して，その中で評価値最大(最小)のものを見出す:腕ずくの方法

例） 4都市TSP



列挙図



問題の規模が大きくなると、組合せ爆発のため実行不能！

→ **列挙図の木の生成途中で、部分評価値の悪い枝を刈る**

最適解が存在する可能性の高い領域を探索

発展形として
タブサーチ
Tabu Search

分枝限定法 (branch and bound method)

◎…「腕ずく法」と「欲張り法」の中間的なバランスの良い方法

×…一般に最適解が求まるとは限らない

【組合せ最適化の解法(3)】

● 経験則的手法・発見的手法(Heuristics)

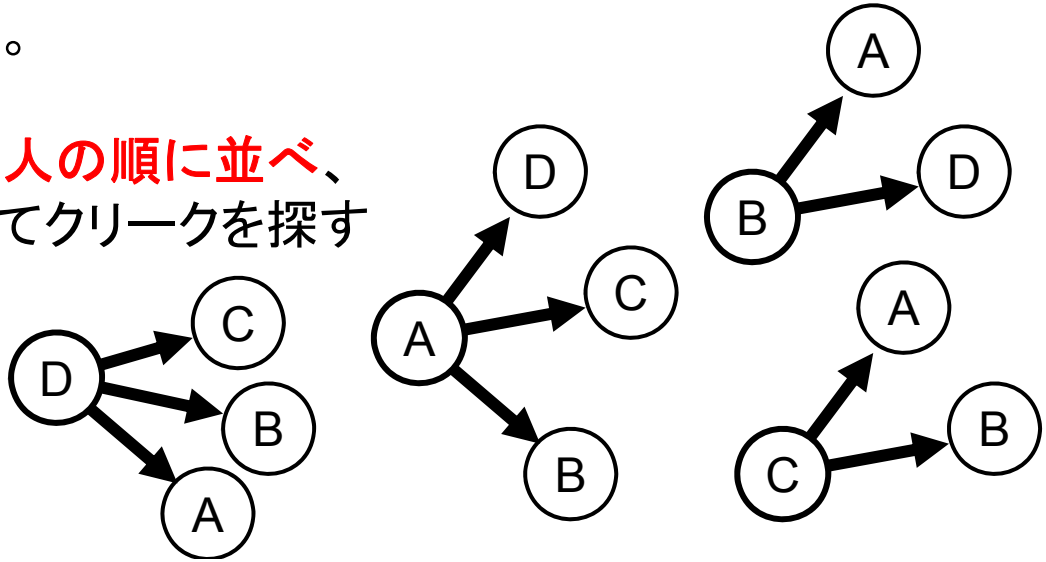
常識的な知識や経験則あるいは試行錯誤的に得た知識などを用いて、
最適ではないが現実的に実行可能な解を得るやり方の総称
〔特徴〕
◎　単純な規則で表現され、計算量が少ない割に良質な近似解を得られる
×　扱う問題毎に方法が異なり、一般的な扱いができない

〔例1〕**クリーク問題**

1000人の電子メール使用者を集め、おのおのはその中の100人以上にメールを出したことがあるものとする。この1000人中からn人を抜き出し、そのn人は全て互いにメールを出しているようにするとき、最大のnになるグループ(クリーク)を作る問題。

→ **受け取っているメール数が多い人の順に並べ**、
その順番にグループを大きくしてクリークを探す

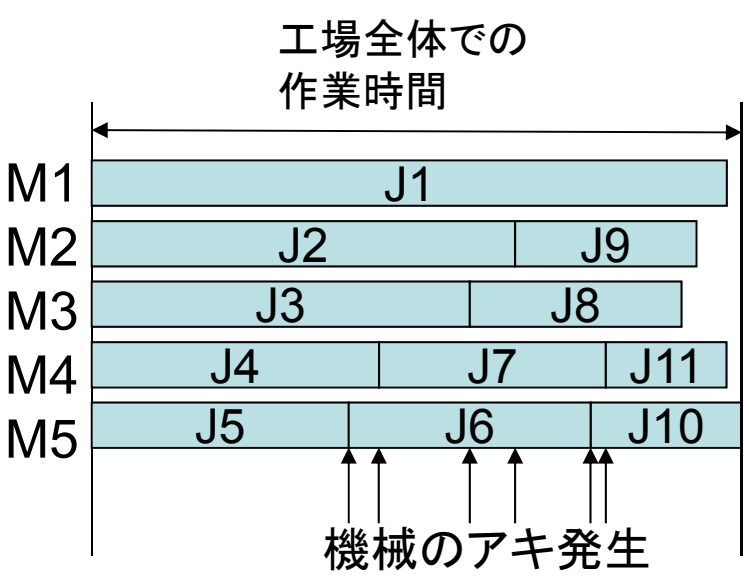
膨大なデータ集合において、
つながりのある無しでデータ
をクラスタリングするのに用いる。
より具体的には、不具合報告の事
例分類など



〔例2〕ジョブショップスケジューリング問題を解くためのヒューリスティクス：LPT則（longest processing time）

単一種類の機械 5台，作業時間の異なる仕事が 11 job存在
全体の作業時間を最短にするよう各機械に仕事と順番を割当てる問題

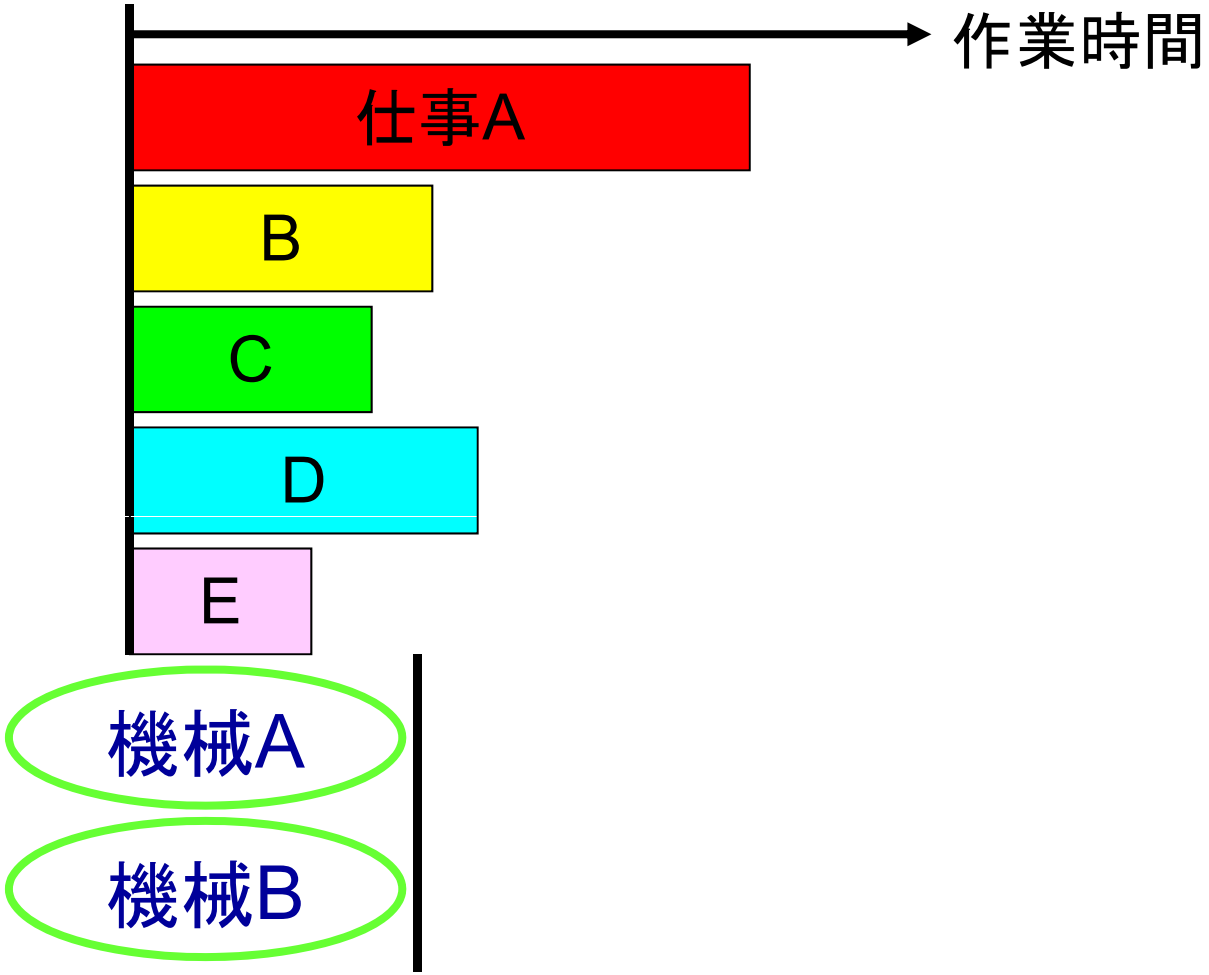
→ 機械があいたとき、残った仕事で最も時間を要するものを割当てる



【参考】 機械m台、仕事数 j とすると、可能な解の組み合わせ総数は m^j
上の簡単な例題でも解の組み合わせ数4882万通り以上

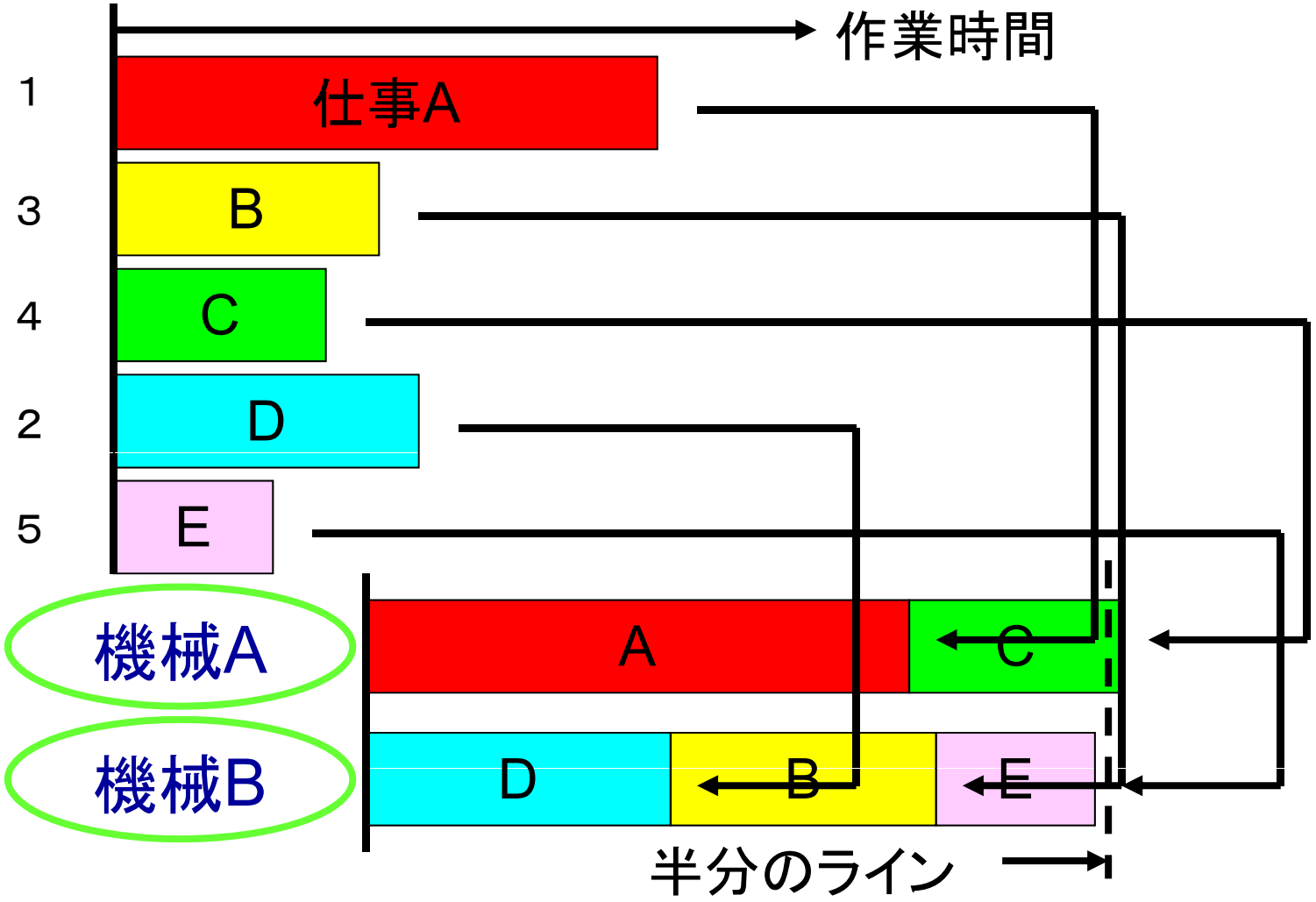
【2機械5Jobのジョブショップスケジューリングにおける LPT則】

まず仕事時間の長い順に順位付け



【2機械5Jobのジョブショップスケジューリングにおける LPT則】

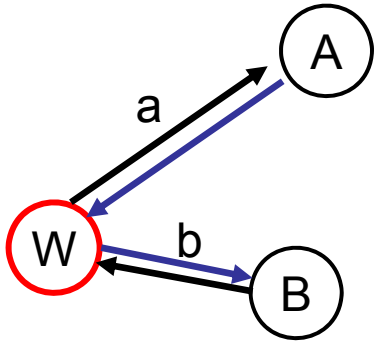
機械が空いたとき、残った仕事で最も時間を要するものを割当てる



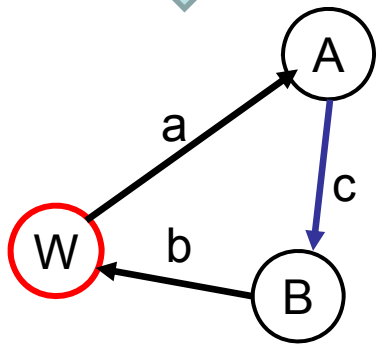
〔例3〕配送問題を解くヒューリスティクス：セービング（saving）

ある倉庫Wから配送先A,B,C,…へトラックで物を配送

WからAへ行き、一度戻ってまたBへ行く配送コスト：2a+2b
W→A→B→Wと倉庫へ戻らずに配送するコスト：a+b+c



上記の差　(2a+2b)-(a+b+c) = a+b-c
これを「ABをつなぐことによるセービング」と呼ぶ
この値がプラスなら、倉庫に戻らずA，Bへ配送するべき



セーブングを利用した配送ルート作成法

1）各2点のセービングを計算し、大きい順に並べる
2）大きい順に2点を結合してルートを作っていく
　（ただし配送上の制約の範囲内で）

［例4］ 制約充足問題をローカルサーチで解くヒューリスティクス：
min-conflicts heuristics

・解は複数のパラメータで表現されているものとする。
・違反している制約条件の数が数えられるものとする。

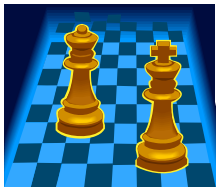
【処理手順】

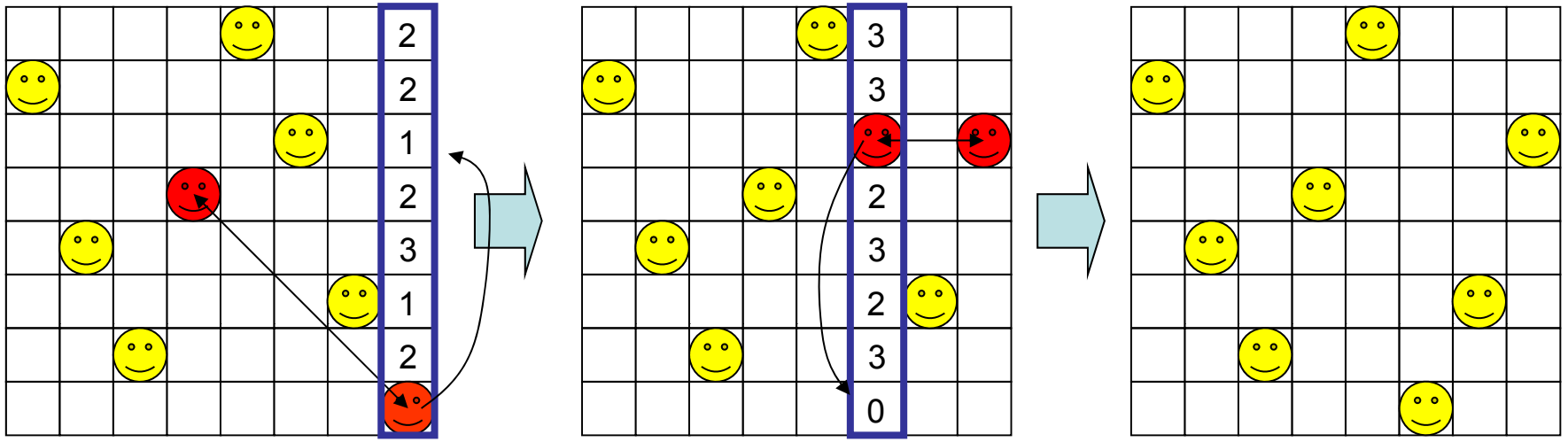
1） 初期解となる解候補を1つランダムに生成する。
2） 解候補のパラメータのうちどれか一つをランダムに選択し、
　　選んだパラメータの値を変えて制約違反の数が最小になるようにする。
3） 制約違反数が0になるまで2)を繰り返す

ローカルサーチ

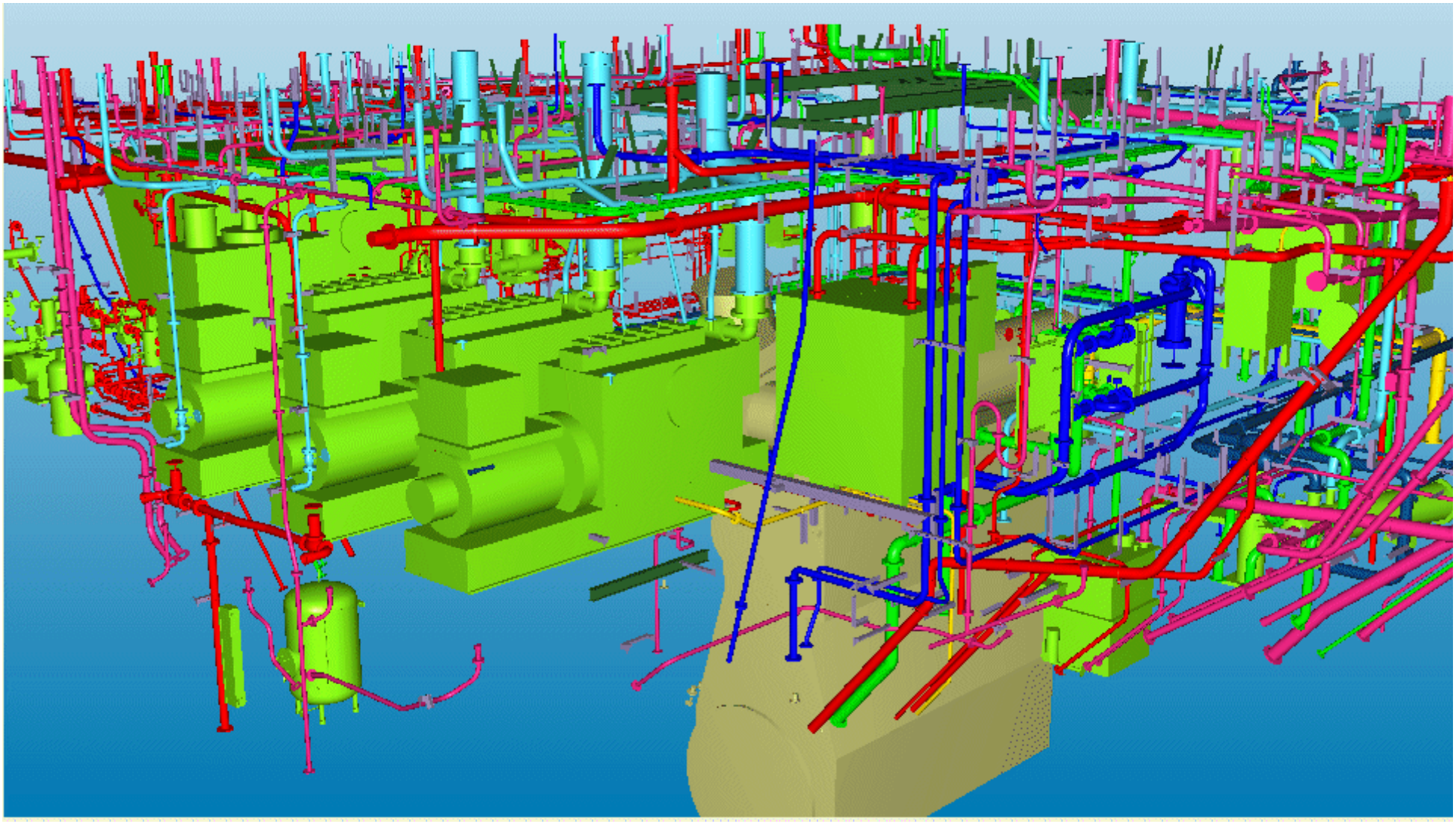
8クイーン問題への適用例

チェスのクイーン：縦横ななめに進める
互いに干渉しないよう盤面に8個配置する問題



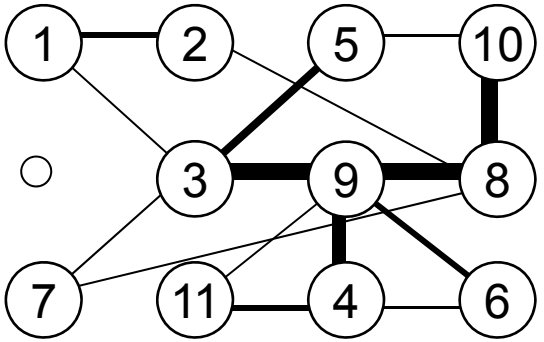


船のエンジンルーム周辺はパイプだらけ　パイプは船価の大きな比率を占める



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

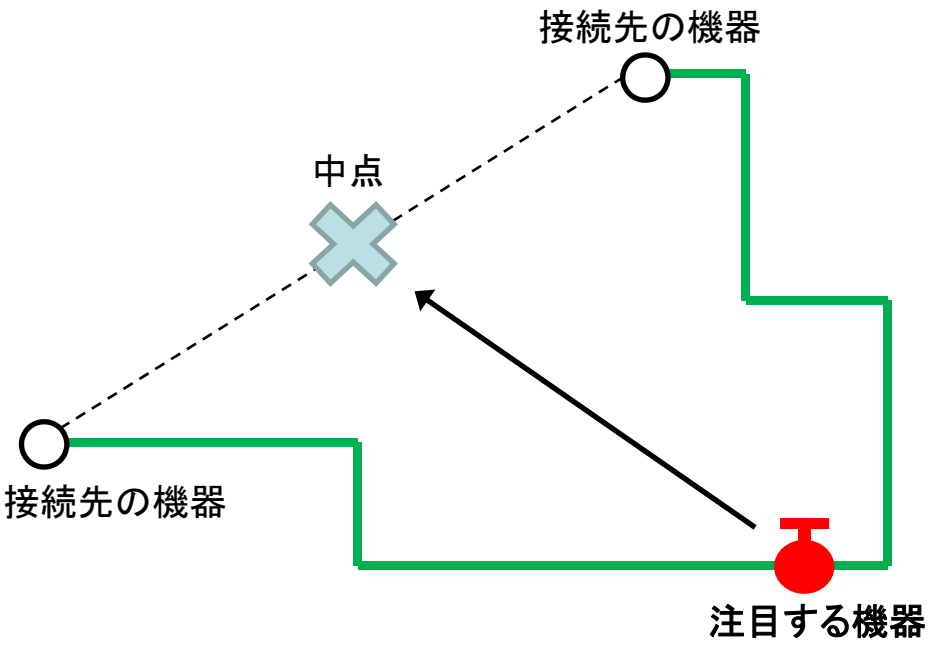
最適化すべき変数 x の2次式

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

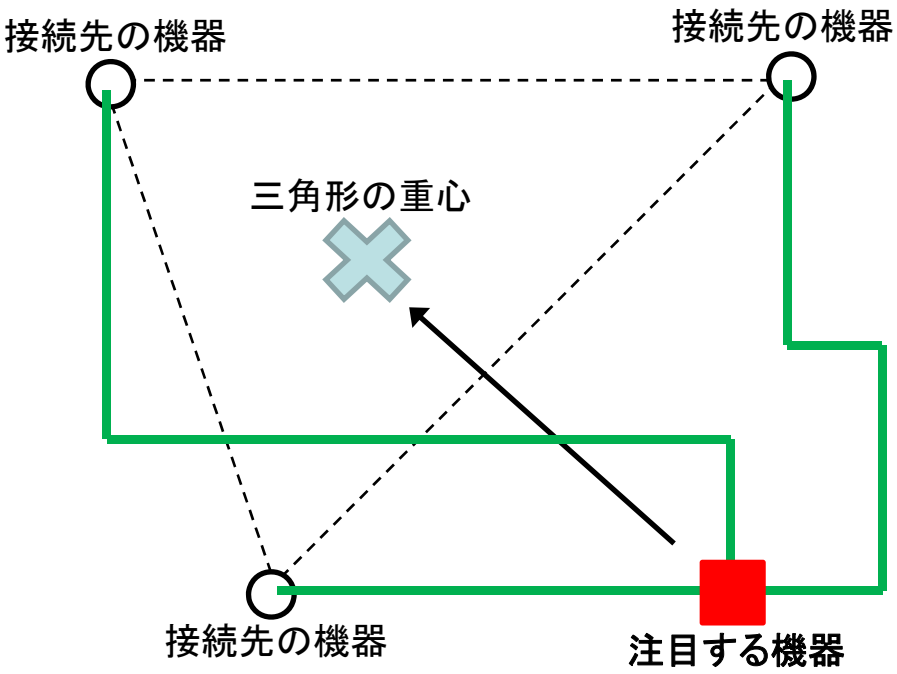
総コストが最小になる配置行列を見つける

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

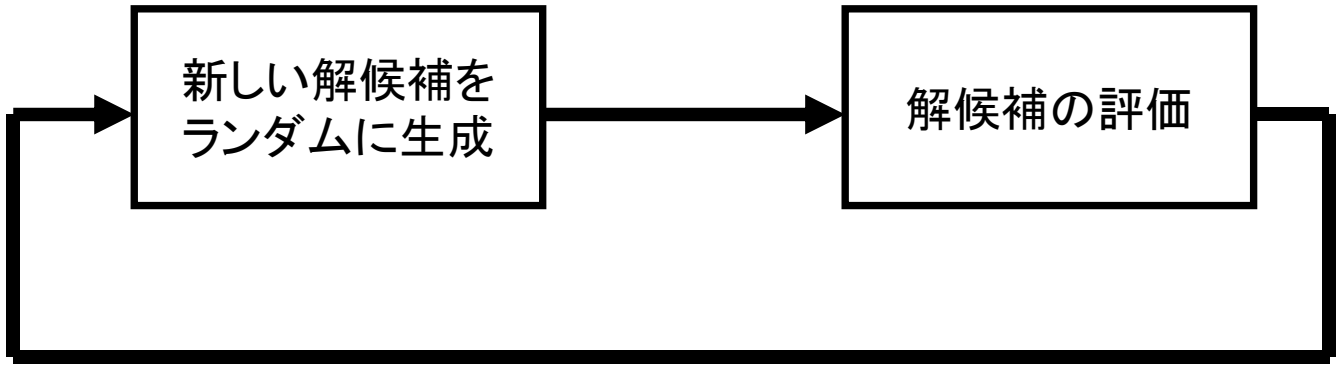
この方法の欠点：注目する機器の移動先にすでに別の機器がある場合どうするか？

【ランダムサーチ・網羅探索による最適化】

モンテカルロ法とも呼ばれる
全数探索と同義な場合と区別する場合がある

・最も単純な方法で、どんな問題に対しても適用可能
・時間さえかければ、理屈の上ではいつかは最適解を発見できる
・今まで生成した解候補と評価値を参考にしないので、探索効率は悪い
　探索性能の下限としての比較対象

ランダムサーチによる最適化のプロセス：



それまで見つけた最良の解候補よりも高い評価値だった場合、
最良解候補を置き換える

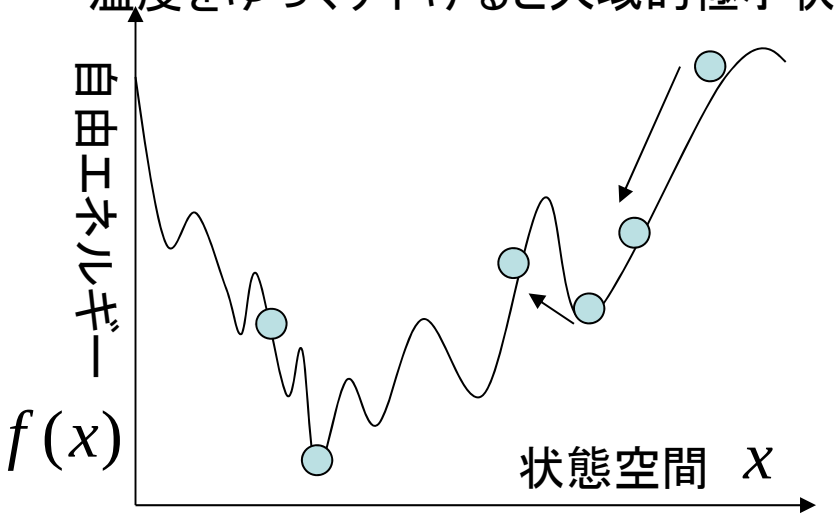
【焼きなまし法(simulated annealing)による最適化】

統計物理学からのアナロジー:

・物質の分子結晶はいろいろな状態を遷移
・エネルギーの低い状態へ遷移しやすい傾向
・温度を下げるとエネルギー極小状態へ到達
・温度を一気に下げると局所的極小状態へ
・温度をゆっくり下げると大域的極小状態へ

「状態」　→　解候補
「エネルギー」　→　評価値

エネルギー最小状態＝最適解
とした最適化のプロセス



システムがある状態 x_i になっている確率

$$\mathrm{P}(x_i \mid T) = \frac{\exp(-f(x_i/T))}{\sum_{x \in S} \exp(-f(x)/T)}$$

ギブス分布（ボルツマン分布）
Tは温度パラメータ

特徴:

1）時間をかければかけるほど解候補が改善され、
　十分時間をかければ最適解の発見が保障される
2）連続関数最適化，および組合せ問題最適化の両方に適用できる

【焼きなまし法(simulated annealing)による最適化】

処理手順

1） 初期解候補 x を生成

2） 何らかの確率分布によって解候補 x から次の解候補 x'を生成
（ランダムに xの近傍を選ぶことが多い）

3） 次の確率に従って乱数Zを生成:
ただしZは0または1の値をとる

$$P(Z=1)=\frac{1}{1+\exp((f(x')-f(x))/T(t)}$$

ただし温度関数 $T(t)=\frac{k}{\ln(t+2)}$

4） $Z=1$ のとき x を x'に置換える、

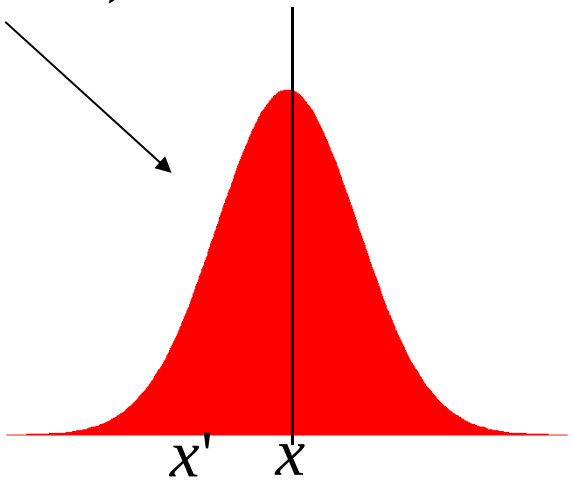
5）時刻 t を1進めて2）から繰返す

これでは収束が遅すぎて非実用的
k/t の関数にすることもある

「近傍」の解候補の生成方法

1） 数値最適化問題の場合： 解候補 x は連続値パラメータなので、
中心値 x, 分散 σ^2 の正規分布によって生成： $x' = N(x, \sigma^2)$

分散のパラメータは設計者が問題に応じて適当に与える
時間とともに小さくしていく場合もある



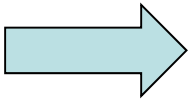
2） 組合せ最適化の場合：
まず解候補の「近傍」を定義する必要がある
→ 問題毎に個別に設定： 実行可能な表現になるよう工夫を要する

例1） TSP
順列中の都市を入れ替える ABCDEF → AECDBF
順列中の部分順列を移動するなど ABCDEF → ACDEBF

例2） SAT
論理変数の論理を反転させる 1011101 → 1001101

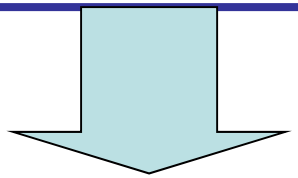
【遺伝的アルゴリズム（GA）による最適化】
組合せ最適化問題における解候補のコード化

・ナップサック問題 → 選択行列
・巡回セールスマン問題 → 順回路行列
または都市順列
・2次割当問題 → 配置行列



これらを遺伝子と考え、
異なる解候補間で交叉
して新しい解候補を生成
交叉オペレーションの設計

【問題点】 交叉で生成された解候補は、たいてい実行不能（致死遺伝子）



SAやランダム
サーチでも同様

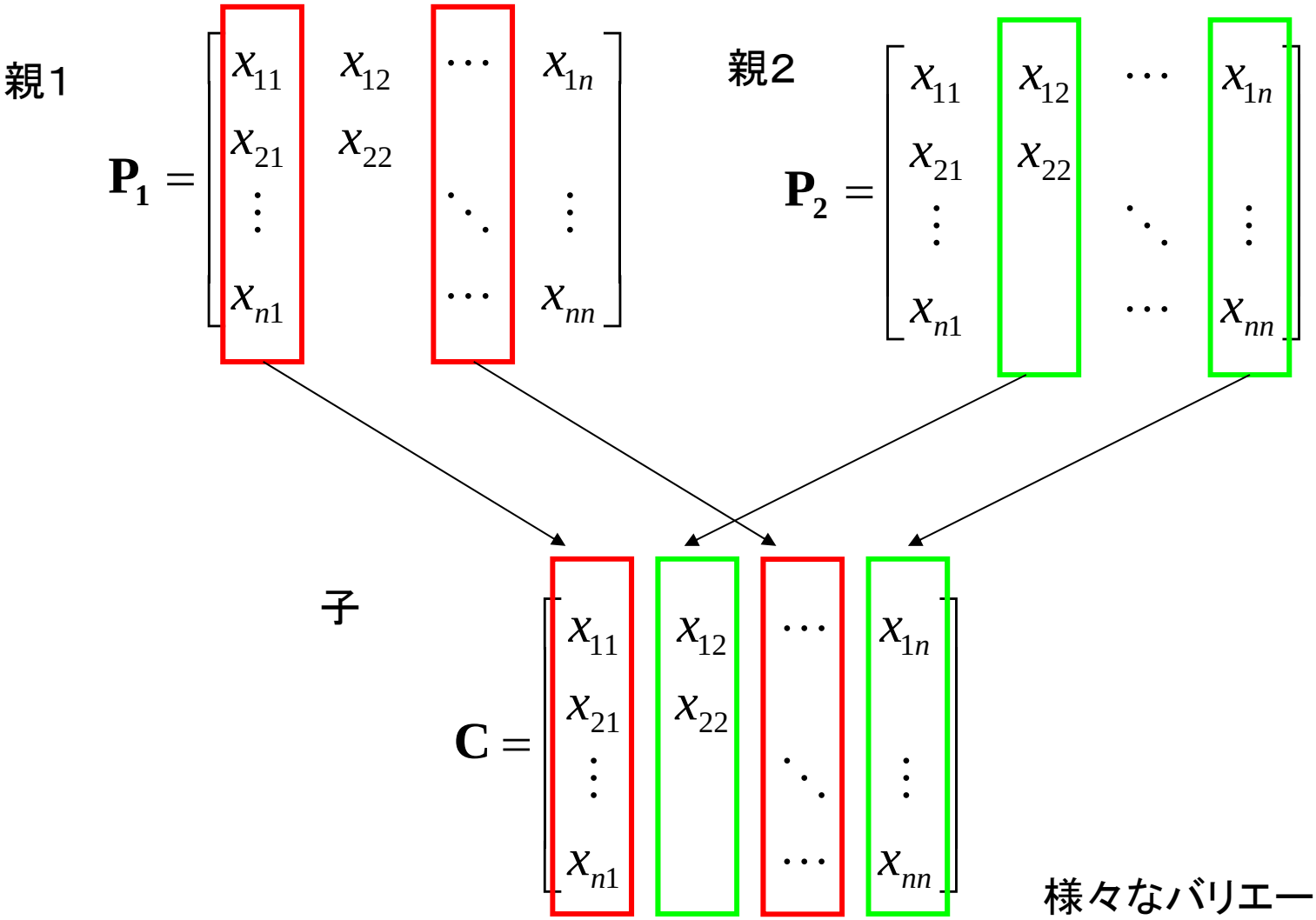
【解決策1】 実行可能な解を得るまで交叉する （場合によっては無限ループに陥る）

【解決策2】 **修正オペレータ**の導入：
致死遺伝子の解候補にローカルサーチを施し、制約を充足するよう修正

【解決策3】 ペナルティ関数法を用いる
制約違反をペナルティとして評価 （ただし適用できる問題に限界あり）

交叉オペレーションの例

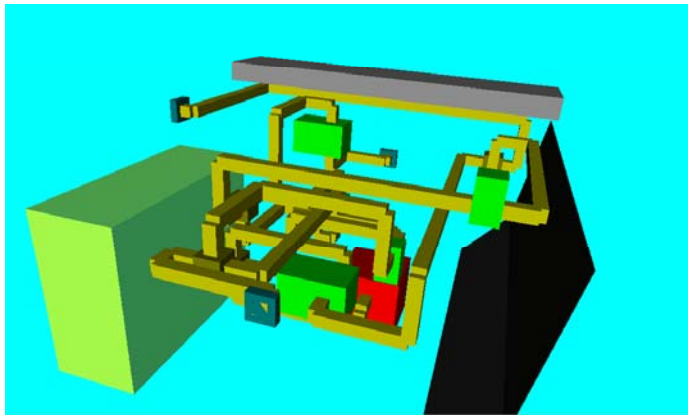
配置行列
順回路行列など

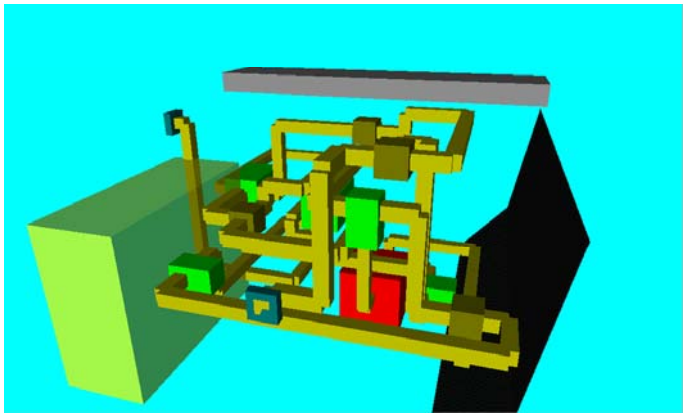


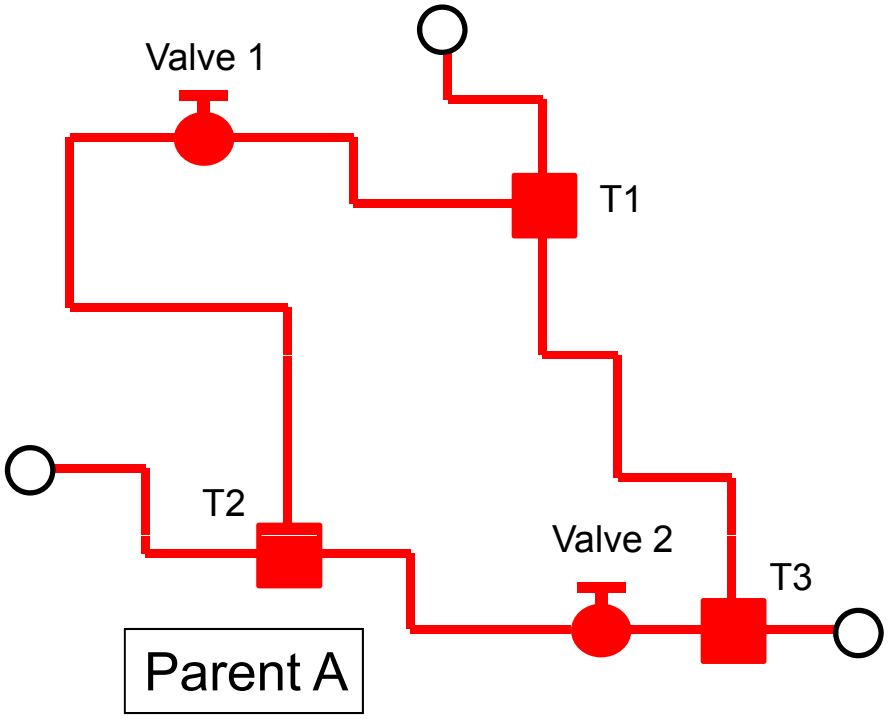
様々なバリエーションが
存在する

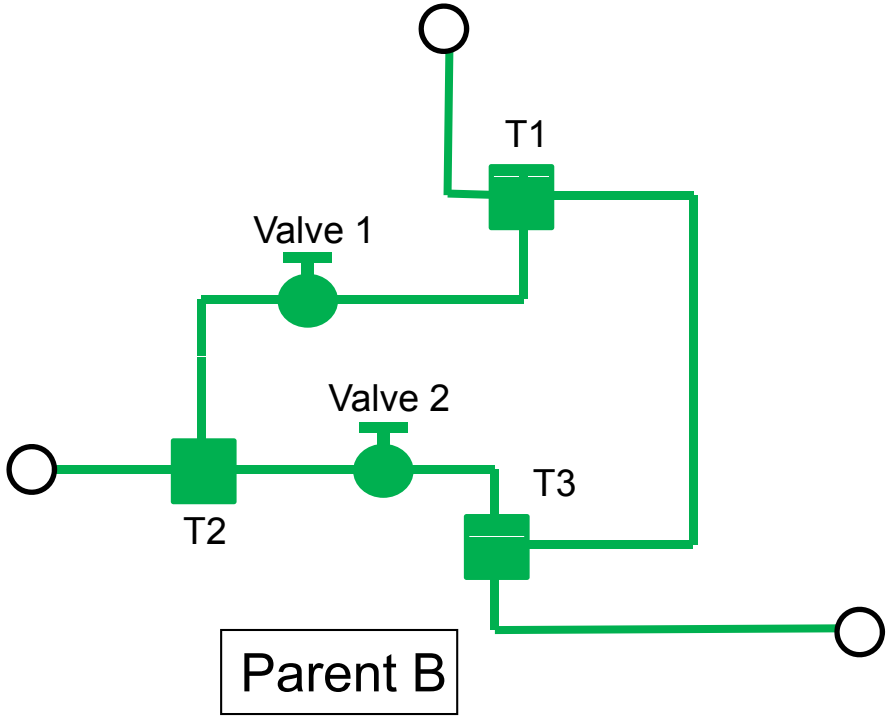
機器配置図の交叉オペレーション

親個体から機器の位置と方向を選んで子個体を形成



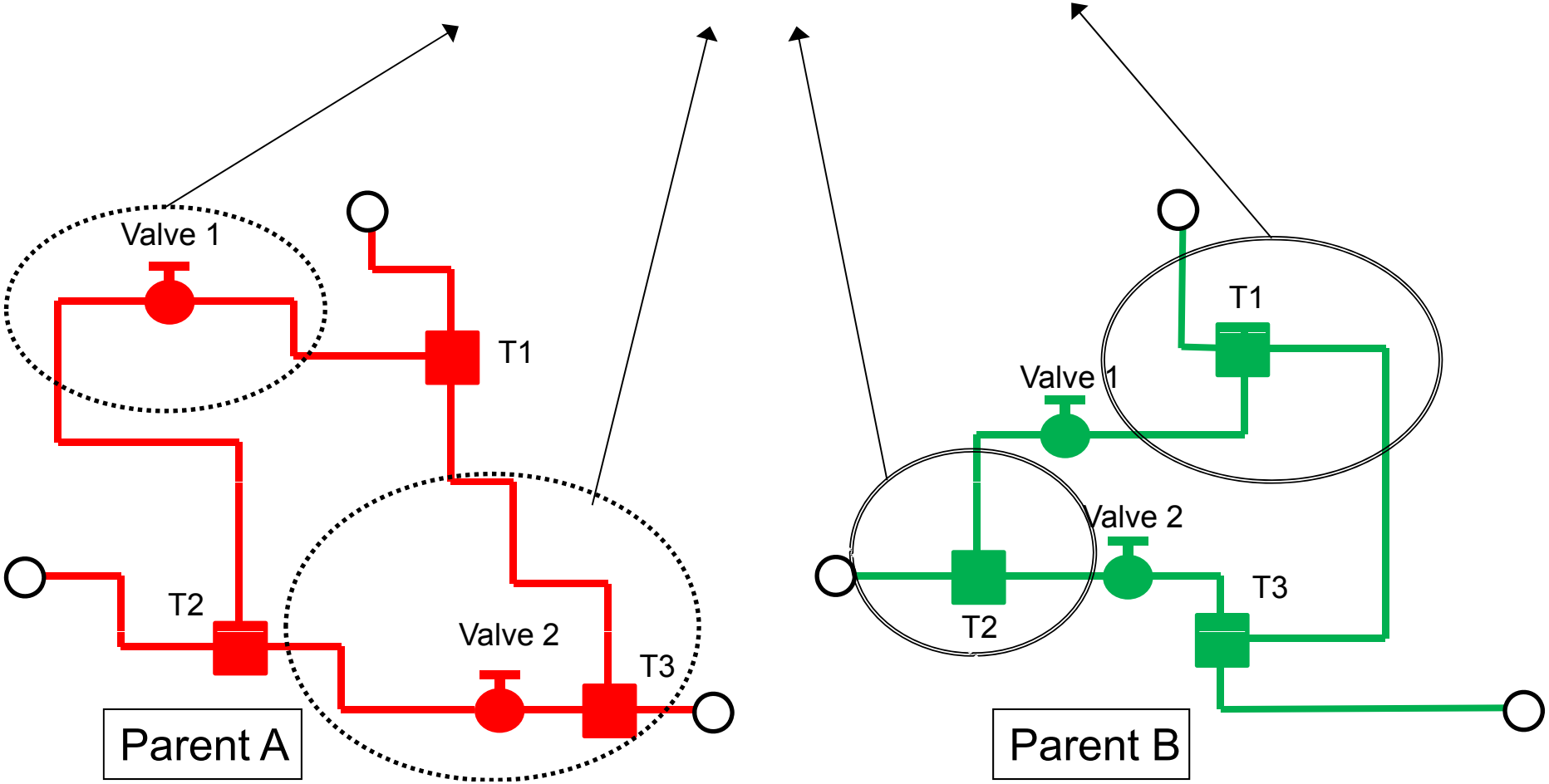


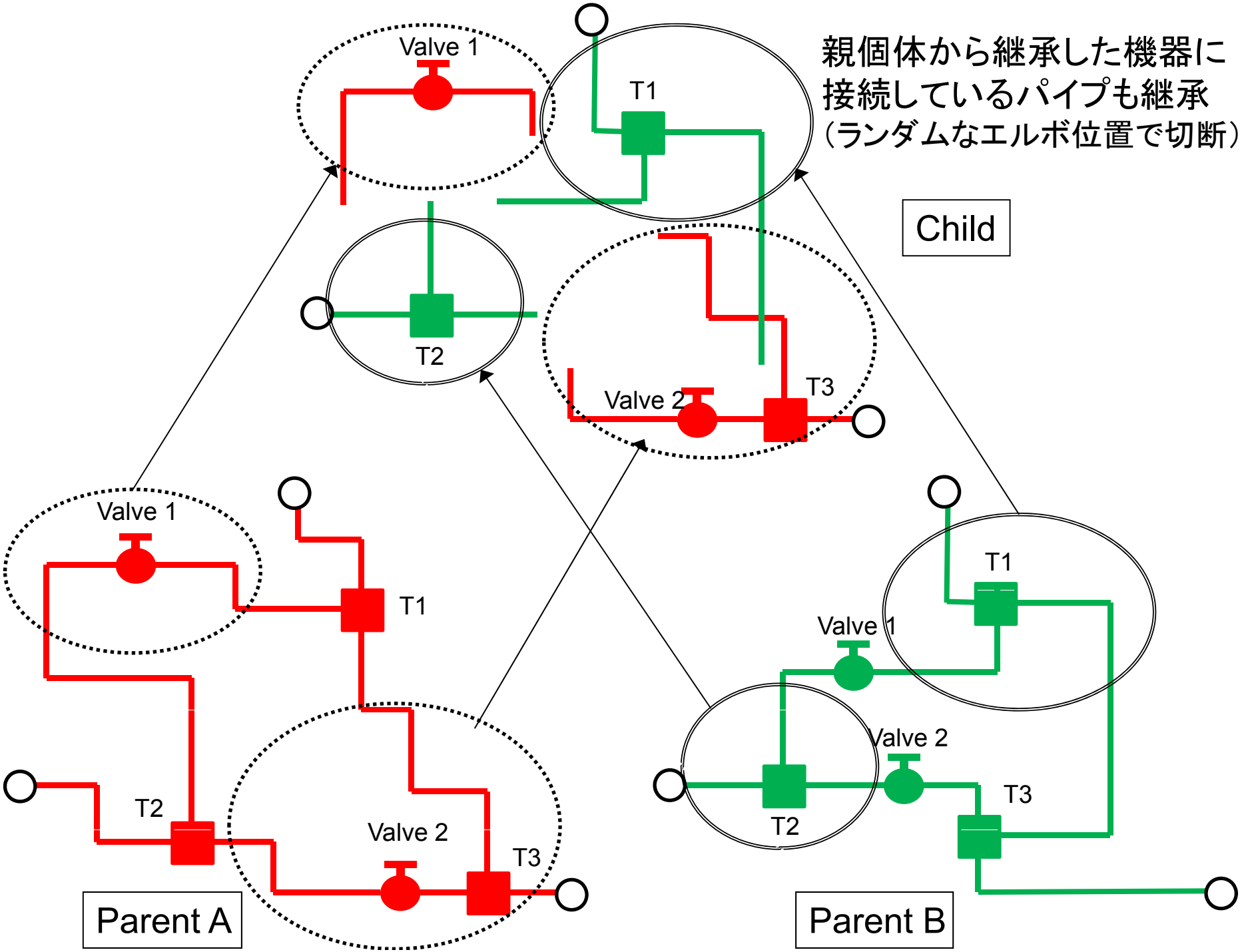


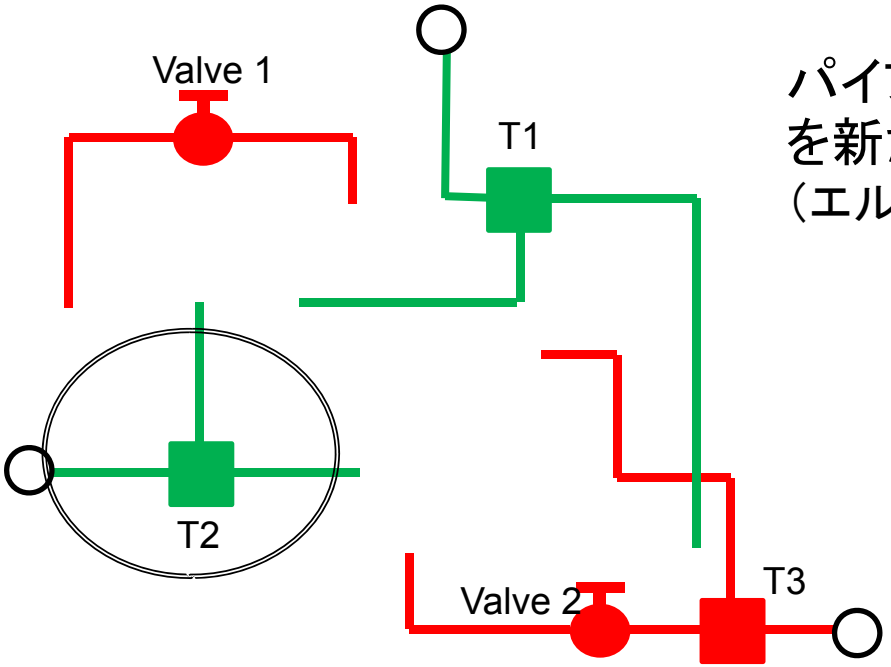


機器配置図の交叉オペレーション

親個体から機器の位置と方向を選んで子個体を形成

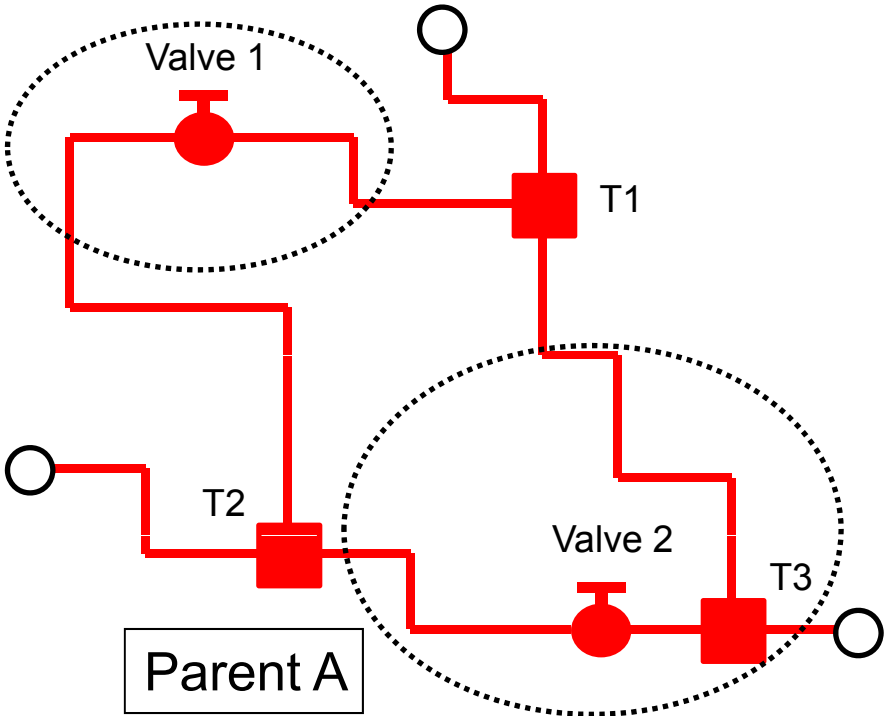



親個体から継承した機器に
接続しているパイプも継承
（ランダムなエルボ位置で切断）
Child
Valve 1
T1
T2
Valve 2
T3
Parent A
Valve 1
T1
T2
Valve 2
T3
Parent B
T1
Valve 1
Valve 2
T2
T3

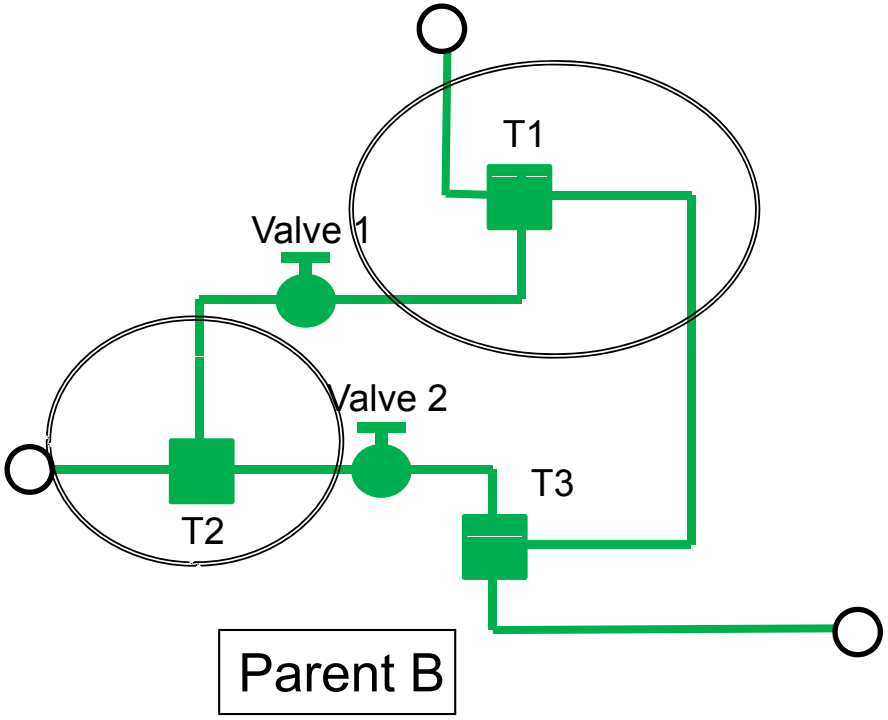


パイプが途切れている部分を新たに繋ぎ直す
（エルボ3個以下で）

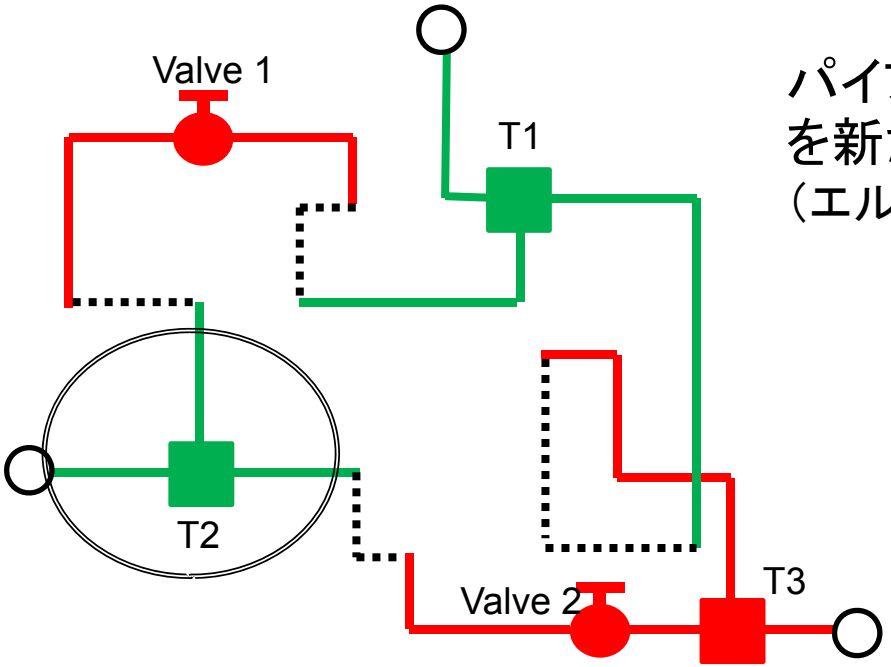
Child



Parent A

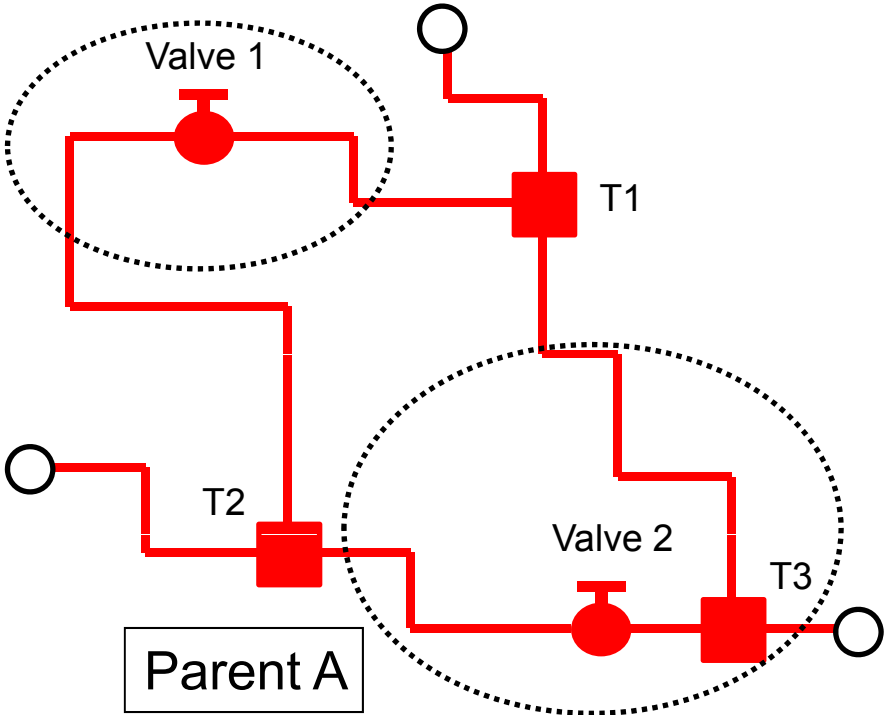


Parent B

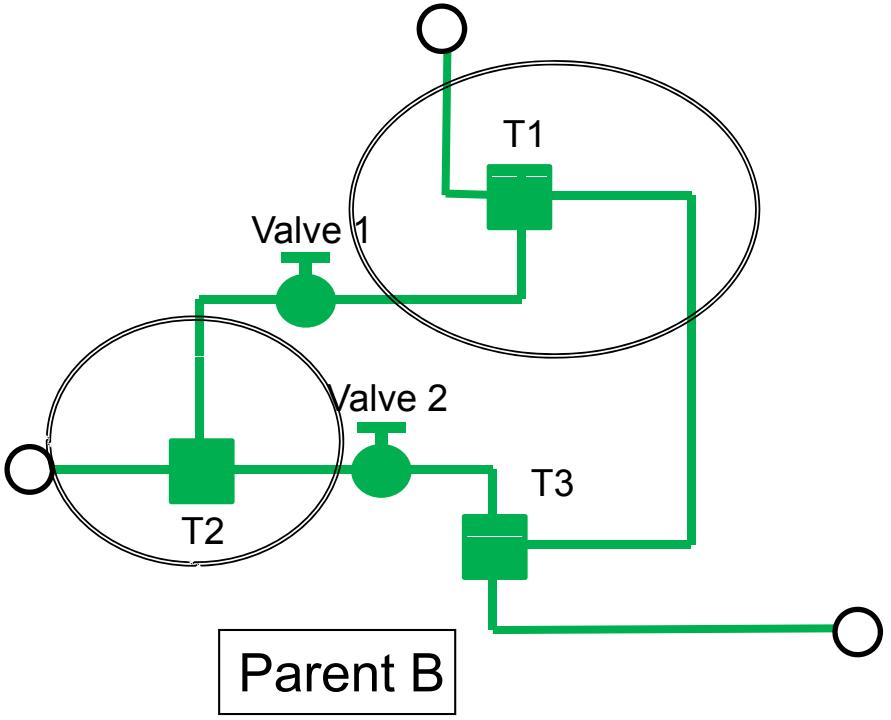


パイプが途切れている部分を新たに繋ぎ直す
（エルボ3個以下で）

Child



Parent A



Parent B

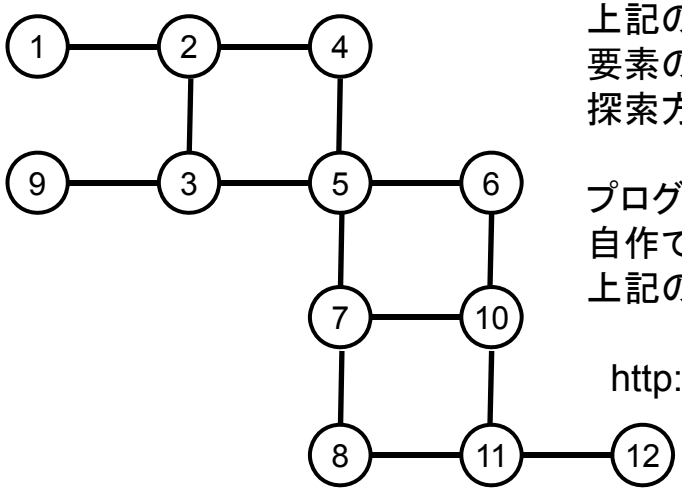
【レポート課題】

各場所間の距離

0	1	2	2	3	4	4	5	3	5	6	7
1	0	1	1	2	3	3	4	2	4	5	6
2	1	0	2	1	2	2	3	1	3	4	5
2	1	2	0	1	2	2	3	3	3	4	5
3	2	1	1	0	1	1	2	2	2	3	4
4	3	2	2	1	0	2	3	3	1	2	3
4	3	2	2	1	2	0	1	3	1	2	3
5	4	3	3	2	3	1	0	4	2	1	2
3	2	1	3	2	3	3	4	0	4	5	6
5	4	3	3	2	1	1	2	4	0	1	2
6	5	4	4	3	2	2	1	5	1	0	1
7	6	5	5	4	3	3	2	6	2	1	0

各要素間のフロー

0	3	4	6	8	5	6	6	5	1	9	6
3	0	6	3	7	9	9	2	2	7	4	7
4	6	0	2	6	4	4	4	2	6	3	6
6	3	2	0	5	5	3	3	9	4	4	6
8	7	6	5	0	4	3	4	5	7	6	7
5	9	4	5	4	0	8	5	5	5	7	5
6	9	4	3	3	8	0	6	8	4	5	7
6	2	4	3	4	5	6	0	1	5	6	3
5	2	2	9	5	5	8	1	0	4	5	9
1	7	6	4	7	5	4	5	4	0	7	7
9	4	3	4	6	7	5	6	5	7	0	2
6	7	6	6	7	5	7	3	9	7	2	0



上記の表で示されるQAPの最適な配置を探索し、
要素の配置およびコストを示せ。
探索方法は問わないが、できる限り工夫せよ

プログラムを自作した場合、動作原理の説明とプログラムリストを添付
自作ではない場合、動作原理の説明とプログラムの入手先等について説明せよ。
上記の表のテキストデータは下記URLからダウンロードすること。

http://sysplan.nams.kyushu-u.ac.jp/gen/edu/SystemsDesign/2015/index.html

【提出期限】 12月22日(火)