動的計画法

この資料では，動的計画法（DP）について解説する。
これまで解説してきた方法は，全て「連続最適化（＝設計変数は，連続値）に属する問題であった。
今回から，離散最適化，あるいは組み合わせ最適化と呼ばれる問題を扱う。
動的計画法は，その代表的な方法である。

動的計画法

動的計画法(DP)は，**逐次決定過程**の最適化問題を解く代表的な方法である。

動的計画法 とは：

逐次決定過程の最適化問題を解く方法

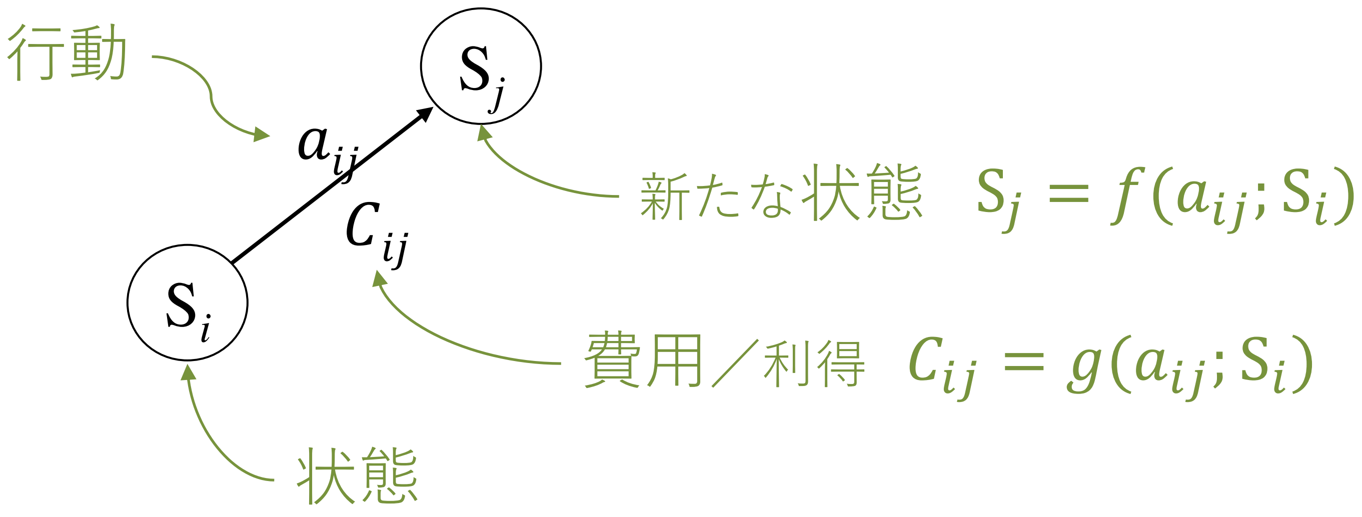
逐次決定過程とは：

過程の各状態において意思決定者が何らかの決定を選択すると状態遷移がおこり、対応する利得（あるいは費用）が発生すると考える数学モデル。

過程から発生する総利得（費用）を最適化する政策（すなわち決定の系列）を求めようとすることが、逐次決定過程の最適化問題。

逐次決定過程

まず，逐次決定過程の基礎となる，**状態遷移モデル**について説明する。

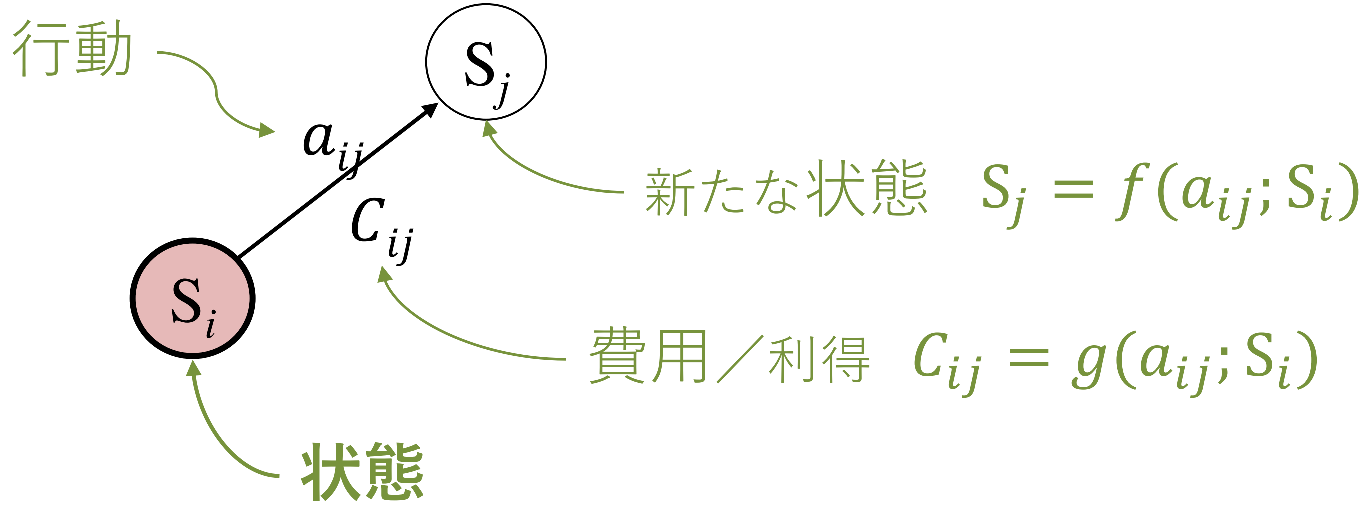


状態遷移モデル：ある状態において意思決定者が何らかの行動をとると，
状態遷移が生じ，遷移に対応して費用（あるいは利得）が発生するモデル

逐次決定過程

状態遷移モデルとは,

ある状態 S_i において,



状態遷移モデル：ある状態において意思決定者が何らかの行動をとると,
状態遷移が生じ，遷移に対応して費用（あるいは利得）が発生するモデル

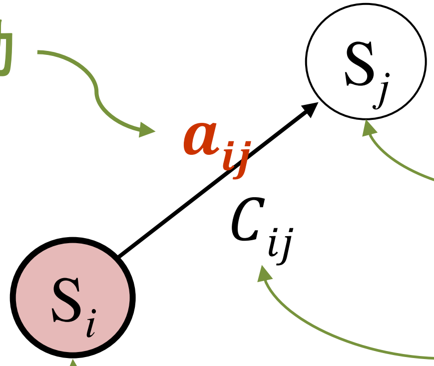
逐次決定過程

状態遷移モデルとは，

ある状態 S_i において，

ある行動 a_{ij} をとると，

行動



新たな状態　$S_j = f(a_{ij}; S_i)$

費用／利得　$C_{ij} = g(a_{ij}; S_i)$

状態

状態遷移モデル：ある状態において意思決定者が何らかの行動をとると，
状態遷移が生じ，遷移に対応して費用（あるいは利得）が発生するモデル

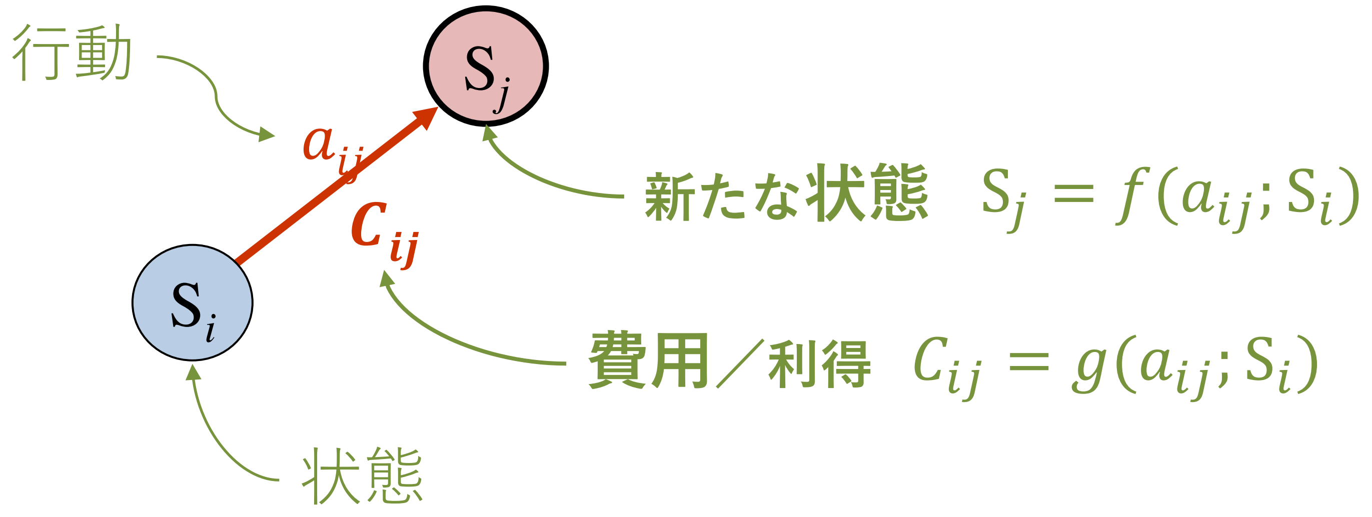
逐次決定過程

状態遷移モデルとは，

ある状態 S_i において，

ある行動 a_{ij} をとると，

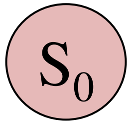
状態が S_j に遷移して，
そこに費用（あるいは利得） C_{ij} が発生する
モデル。



状態遷移モデル：ある状態において意思決定者が何らかの行動をとると，
状態遷移が生じ，遷移に対応して費用（あるいは利得）が発生するモデル

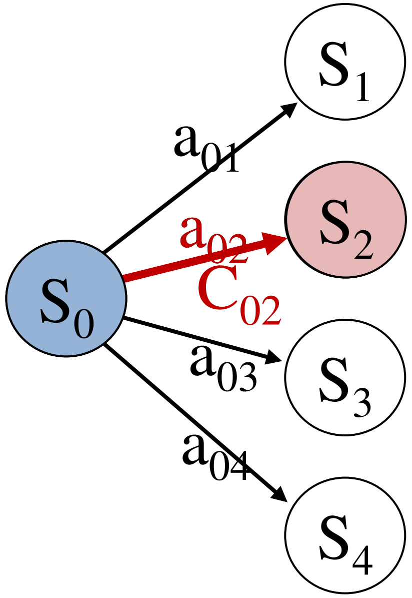
逐次決定過程

逐次決定過程においては
，この処理を繰り返す。



逐次決定過程：行動を逐次選択して状態遷移を繰り返し，
費用（あるいは利得）を繰り返し得る過程

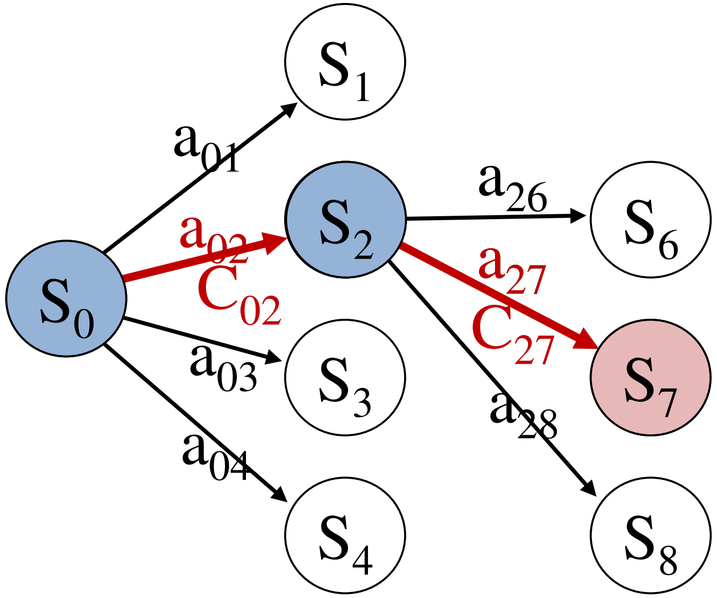
逐次決定過程



逐次決定過程においては，この処理を繰り返す。

逐次決定過程：行動を逐次選択して状態遷移を繰り返し，
費用（あるいは利得）を繰り返し得る過程

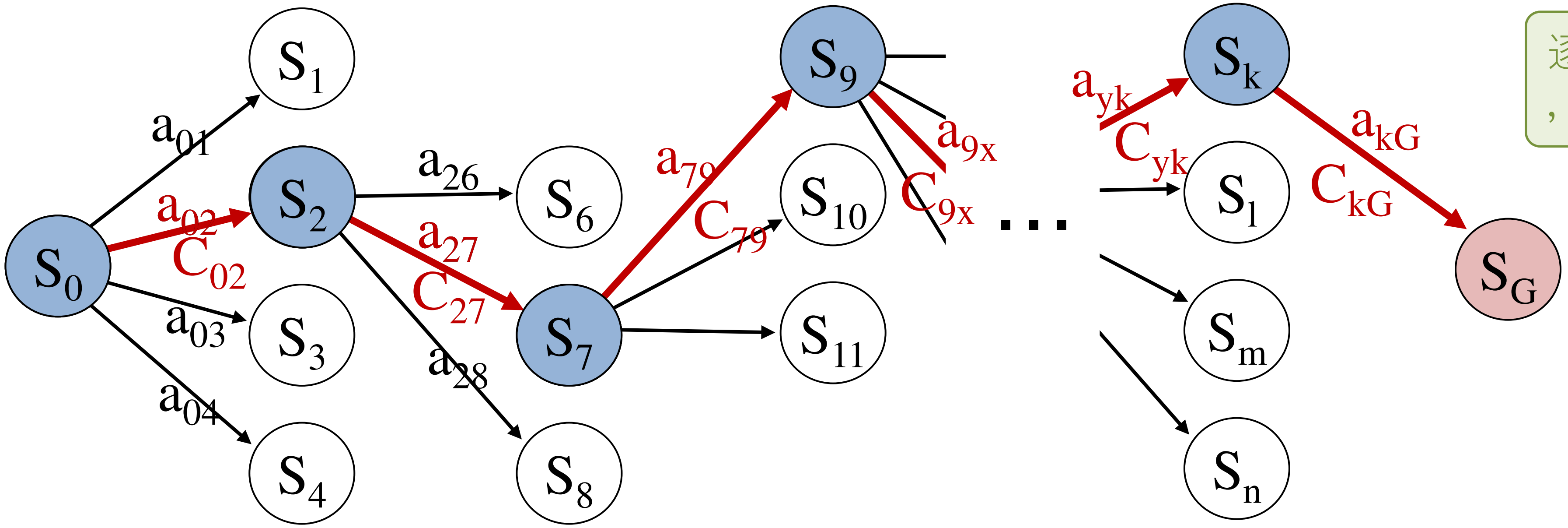
逐次決定過程



逐次決定過程においては，この処理を繰り返す。

逐次決定過程：行動を逐次選択して状態遷移を繰り返し，費用（あるいは利得）を繰り返し得る過程

逐次決定過程

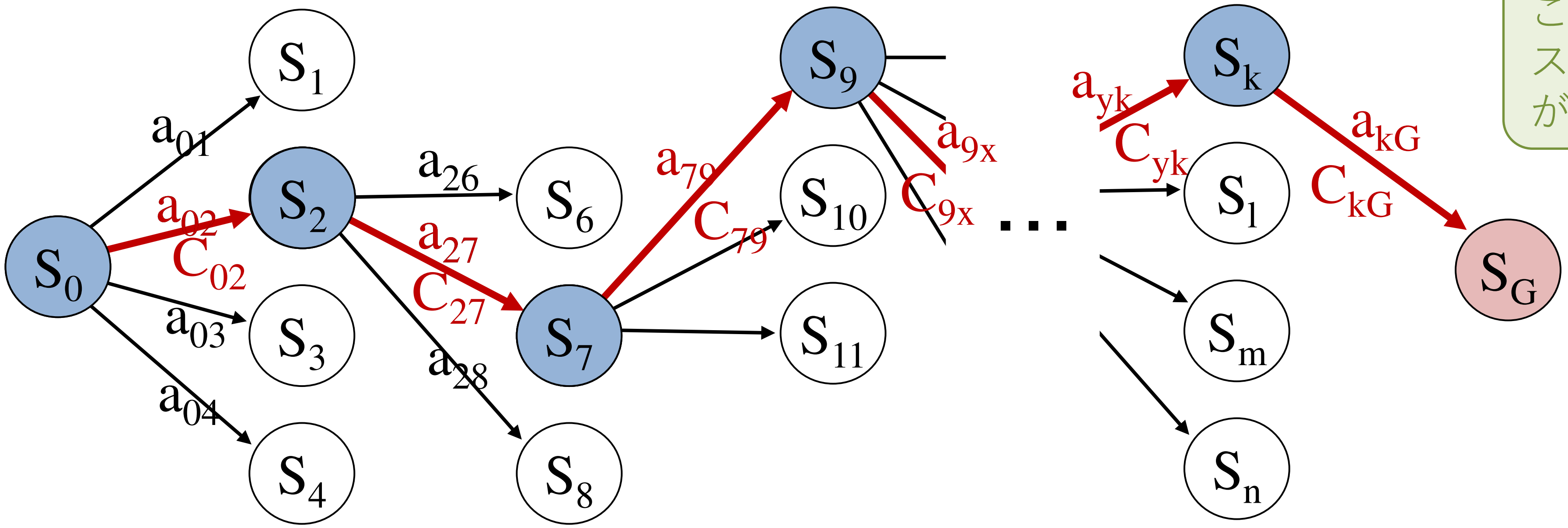


逐次決定過程においては，この処理を繰り返す。

逐次決定過程：行動を逐次選択して状態遷移を繰り返し，費用（あるいは利得）を繰り返し得る過程

逐次決定過程

行動の列が決まれば，状態列が決まり，
この状態遷移による総コスト（あるいは総利益）が決まる。



$\theta = \{a_{02}, a_{27}, \cdots, a_{kG}\}$:行動の列，

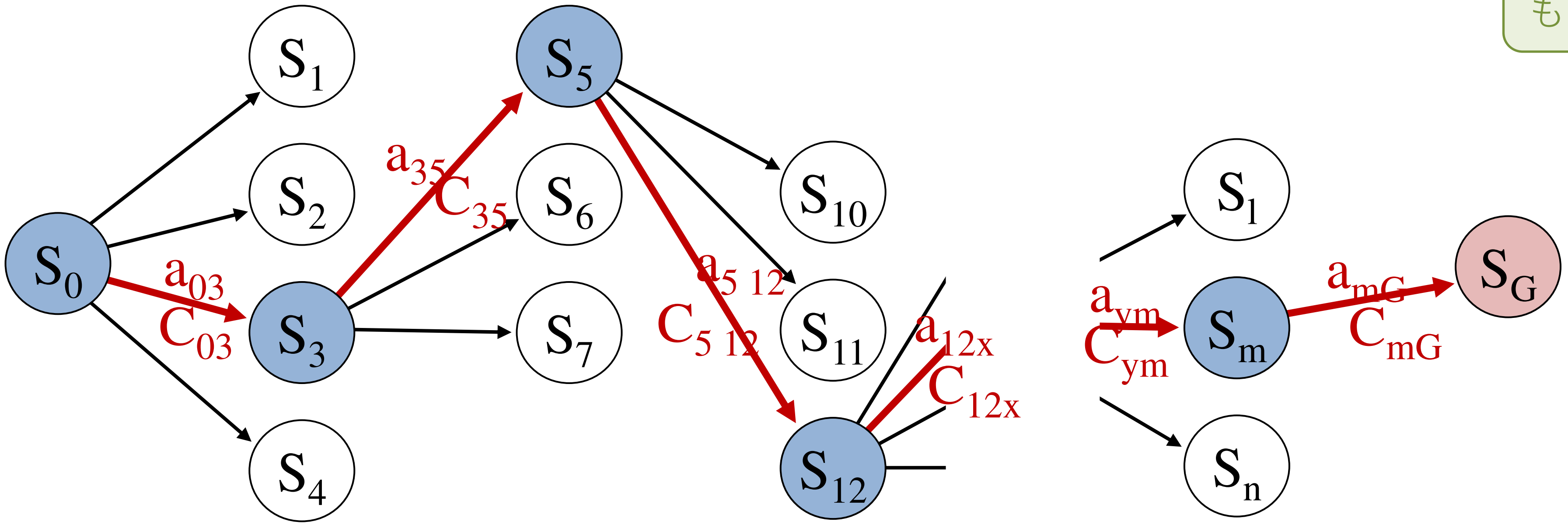
$A = \{02, 27, \cdots, kG\}$:θが決める状態遷移の列，

$\Psi(\theta) = \sum_{ij \in A} C_{ij}$ ：θが決める遷移に沿って得られる費用／利得の総和

逐次決定過程：行動を逐次選択して状態遷移を繰り返し，費用（あるいは利得）を繰り返し得る過程

逐次決定過程

異なる行動を選べば，総費用（あるいは総利益）も変わる。



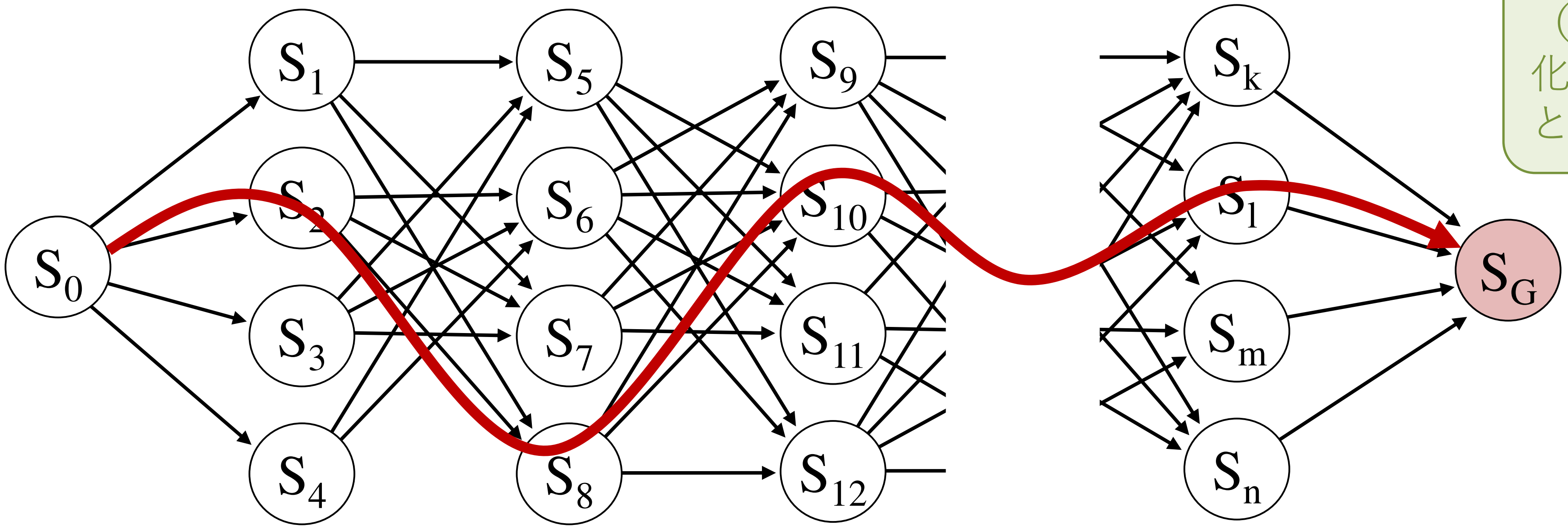
$\theta = \{a_{03}, a_{35}, \cdots, a_{mG}\}$:行動の列，

$A = \{03{,}35, \cdots, mG\}$:θが決める状態遷移の列，

$\Psi(\theta) = \sum_{ij \in A} C_{ij}$ ：θが決める遷移に沿って得られる費用／利得の総和

異なる行動を選べば，費用／利得の総和も変わる

逐次決定過程の最適化

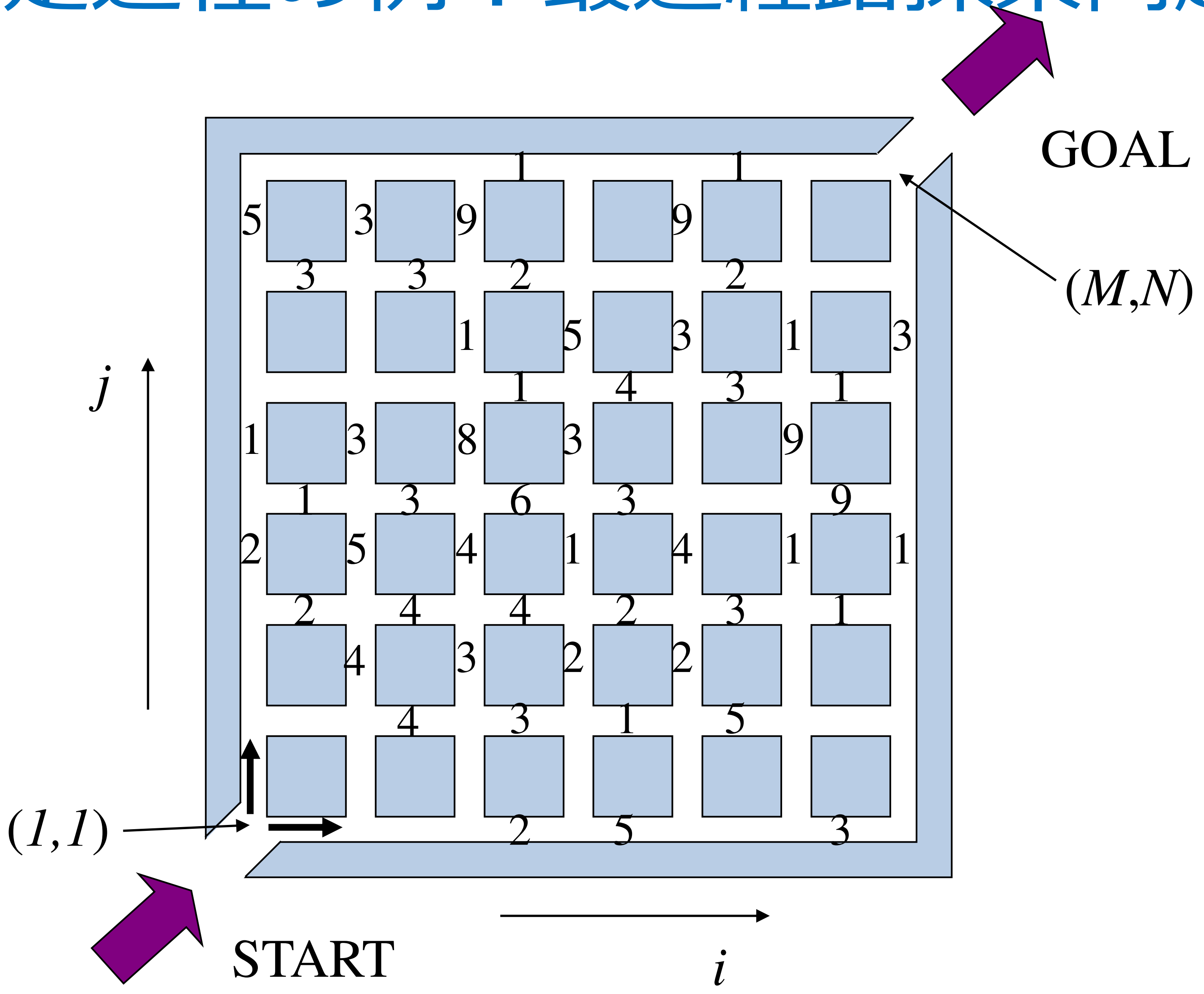


ここで，どういう行動を選べば，総費用を最小化（あるいは総利益を最大化）することができるかという問題が生じる。

$$\min_{\theta} \Psi(\theta),$$ θは選びうる行動の列

逐次決定過程において，発生する 総費用/利得 を 最小化/最大化 するには，
行動はどのように選択するべきか?

逐次決定過程の例：最適経路探索問題



典型的な問題として，最適経路問題があげられる。
左は，STARTから始まって，上か右に移動しながらGOALに至るとき，経路上にある数値の総和を最大化（あるいは最小化）するためには，どういう経路をとればよいかという問題である。

ベルマンの最適性の原理

ここでは，この種の問題をDPで解く方法について解説するが，そこで基本となるのが，ベルマンの最適性の原理である。

◎ 最適政策では，初期状態，初期の行動が何であろうと，以後の政策は最初の遷移から生じた状態に関して適切でなければならない。

◎ ある期間を通じての最適問題の解として最適政策は，元々の問題を部分期間に区切った部分最適問題の解を一部として持つ。

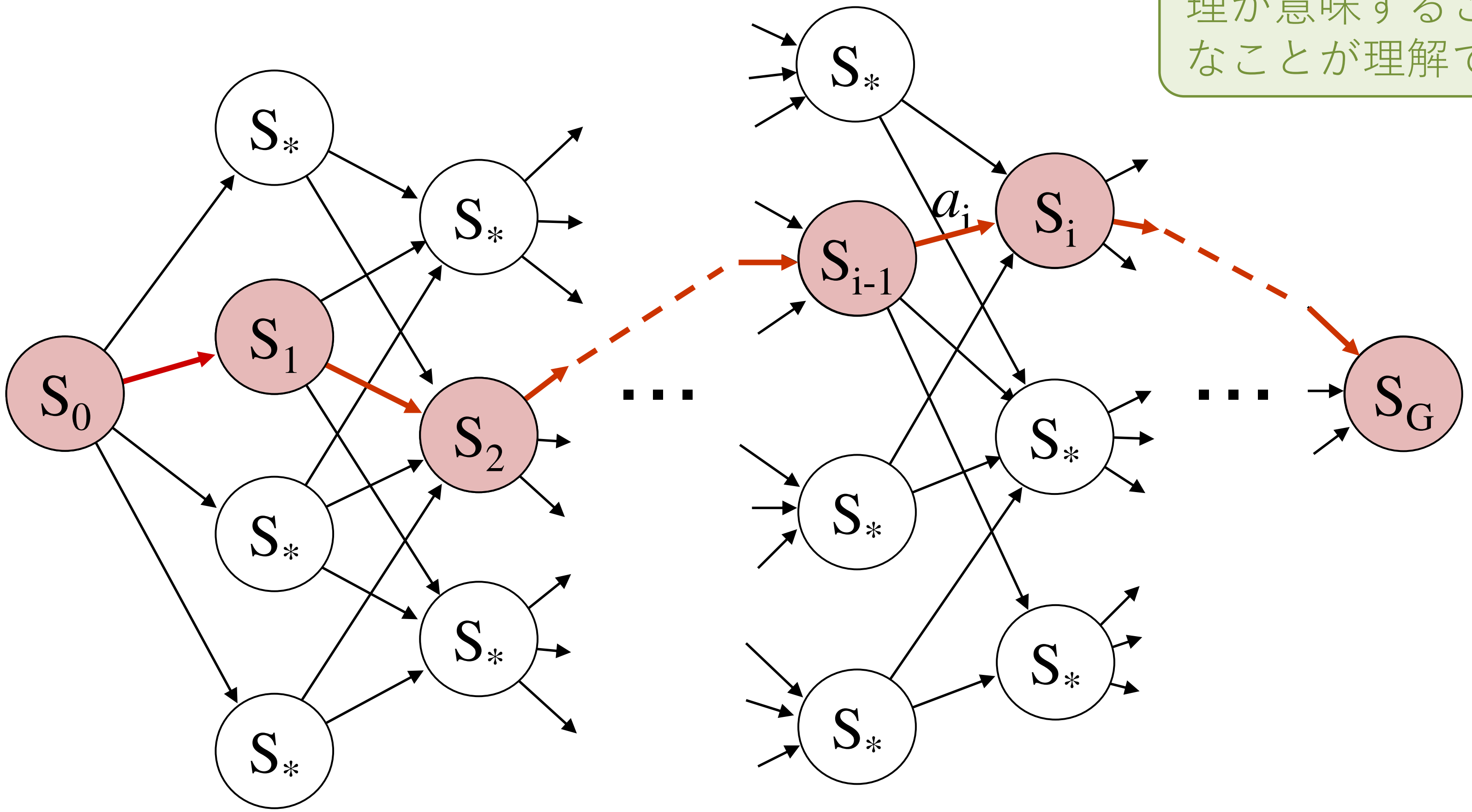
･･･言い換えると・・・

◎ N 個の決定の系列 $a_1 a_2 \cdots a_i \cdots a_N$ によって，状態が $\mathrm{S}_0 \mathrm{S}_1 \mathrm{S}_2 \cdots \mathrm{S}_i \cdots \mathrm{S}_N = \mathrm{S}_\mathrm{G}$ と変化するものとする。（S_0 は初期状態，S_G は目的状態である。）

このとき，この行動の系列が，S_0 から S_G への最適政策であるならば，この状態系列上の任意の S_i に対し，$a_{i+1} \cdots a_N$ は，S_i から S_G への最適政策になっていなければない。

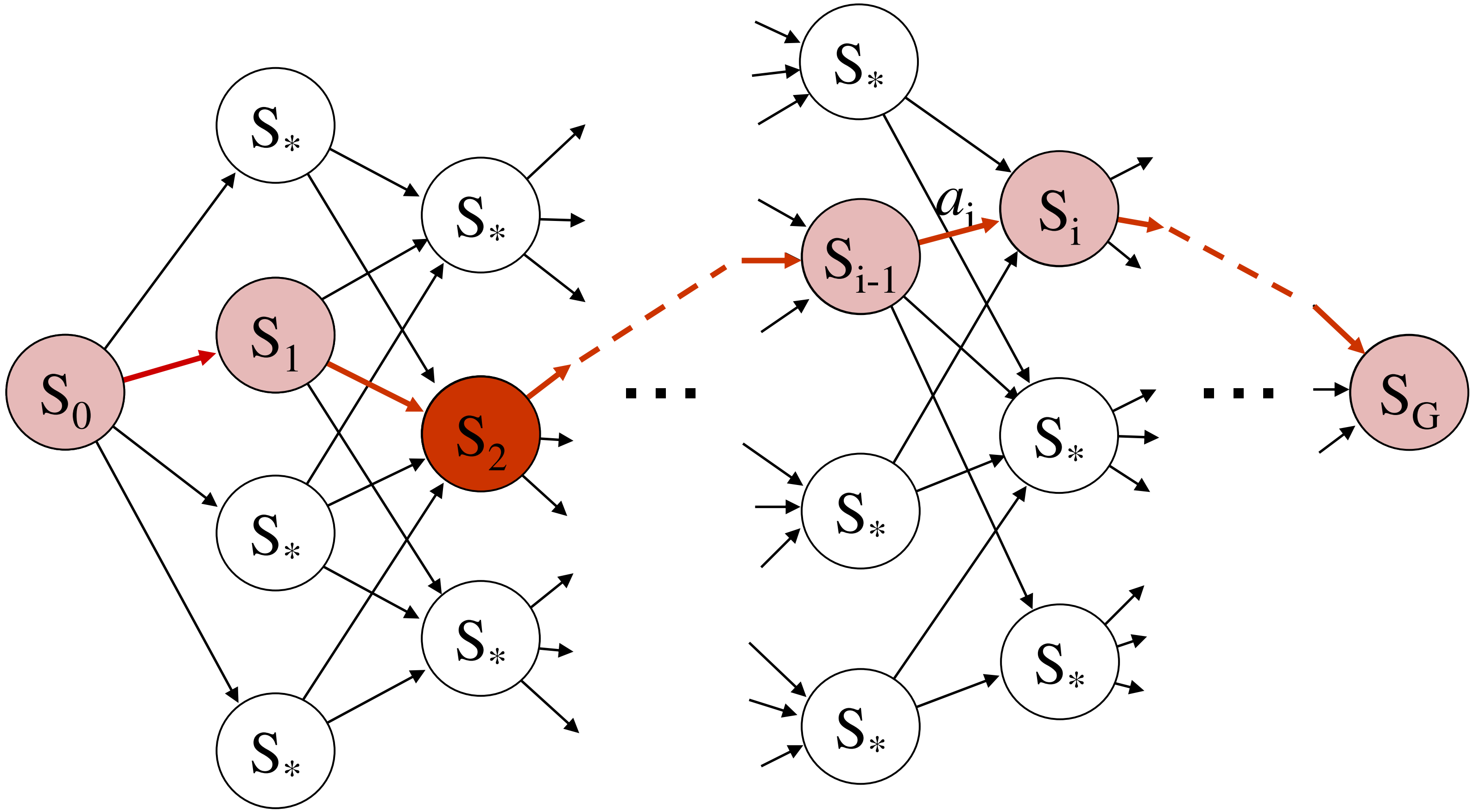
最適性の原理

言葉で書くと，なかなかわかりにくいが，図を使えば，最適性の原理が意味することは，比較的簡単なことが理解できる。



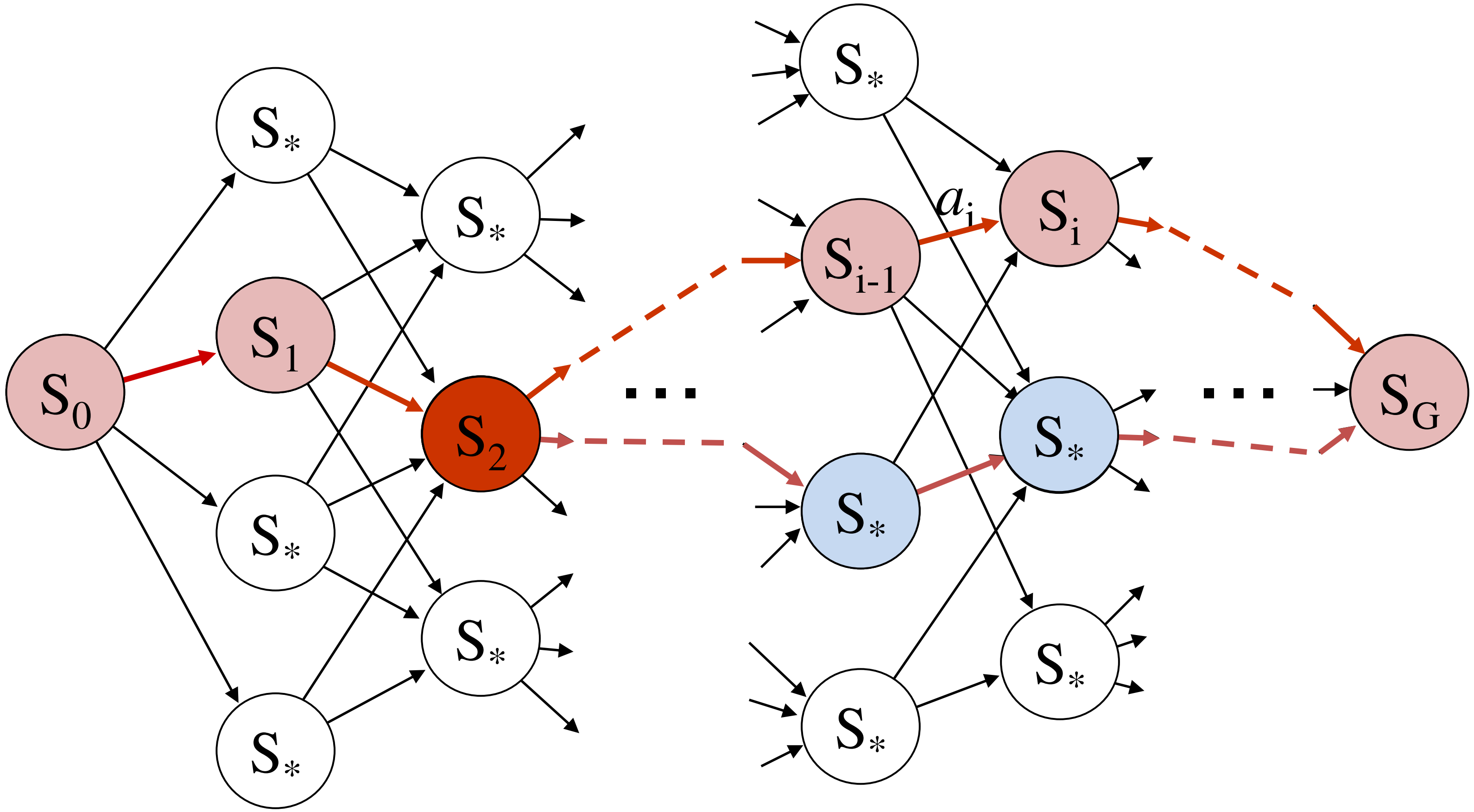
赤で示したような最適政策（最適パス）が与えられたとする。

最適性の原理



最適政策上のある状態（例えば S_2）を考える。このとき，S_2 から S_G までの最適政策も赤のパスに一致する。

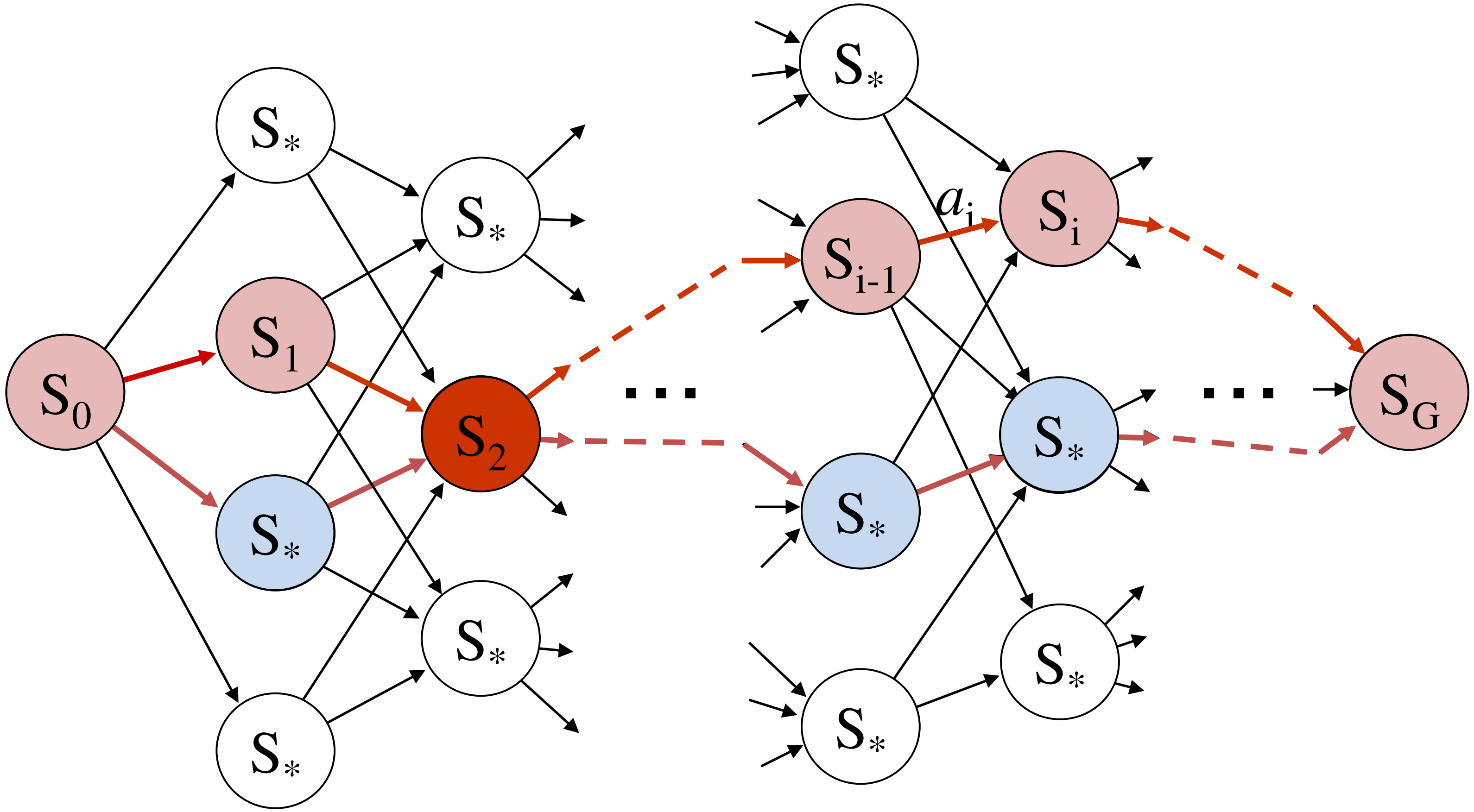
最適性の原理



仮に，青のパスが S_2 から S_G までの最適政策なら，全体の最適政策も
S_0 ー(赤)ーS_2ー(青)ーS_G となって，仮定に反する。

最適性の原理

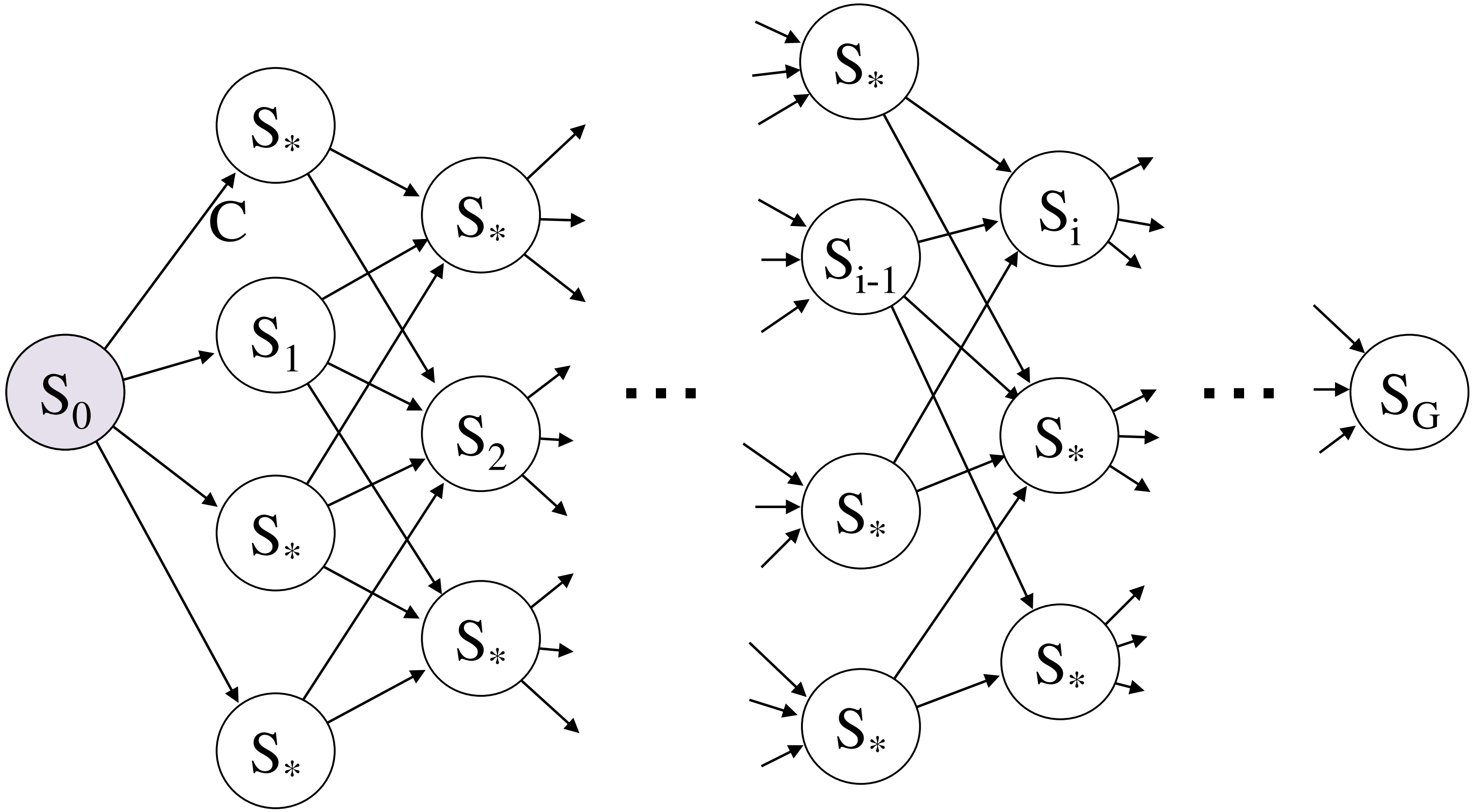
基本的には，このことを表したのでベルマンの最適性の原理。



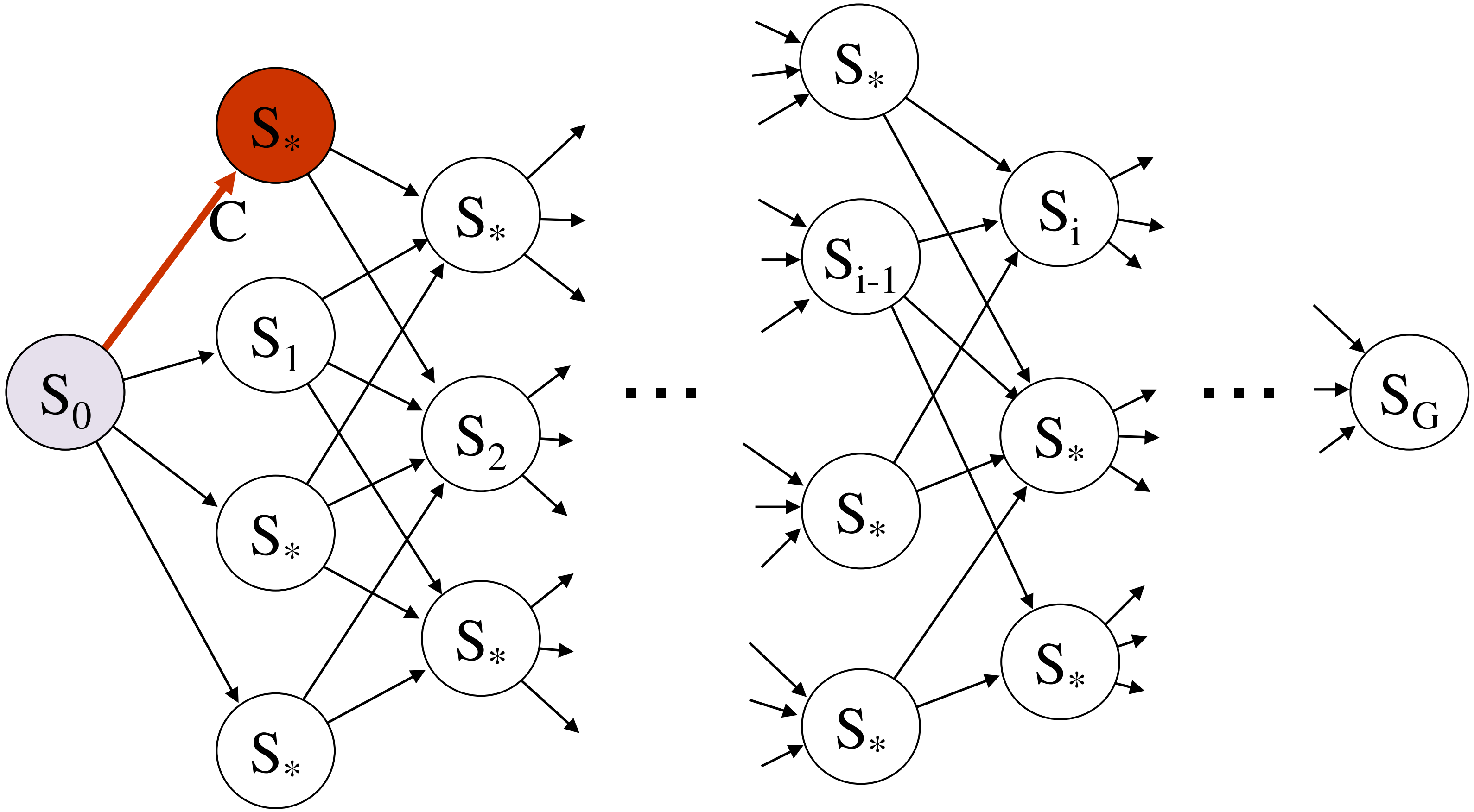
初期状態から S_2 までの最適政策も同様に赤いパス。

動的計画法

最適性の原理を踏まえて，動的計画法のアルゴリズムを解説する。

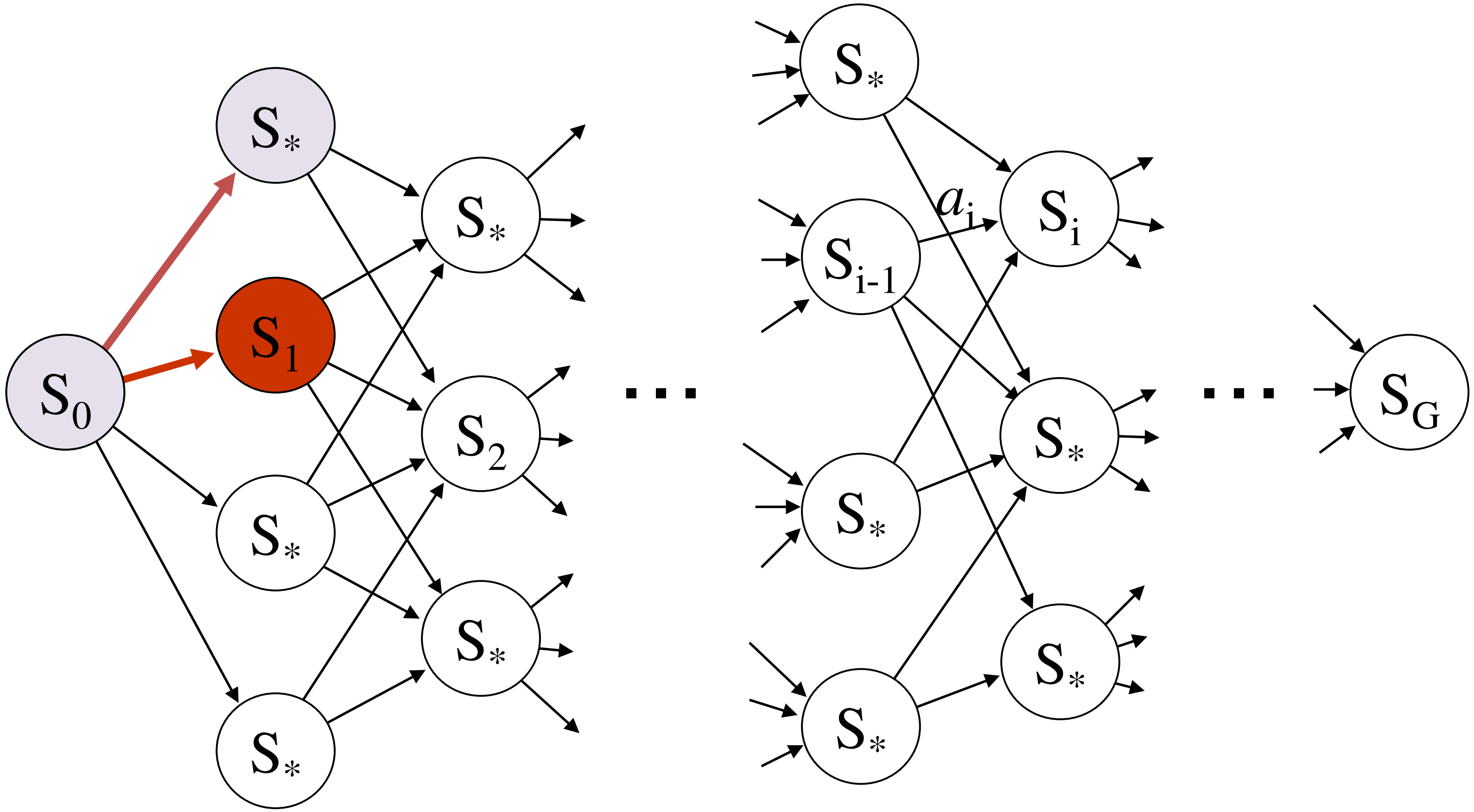


動的計画法



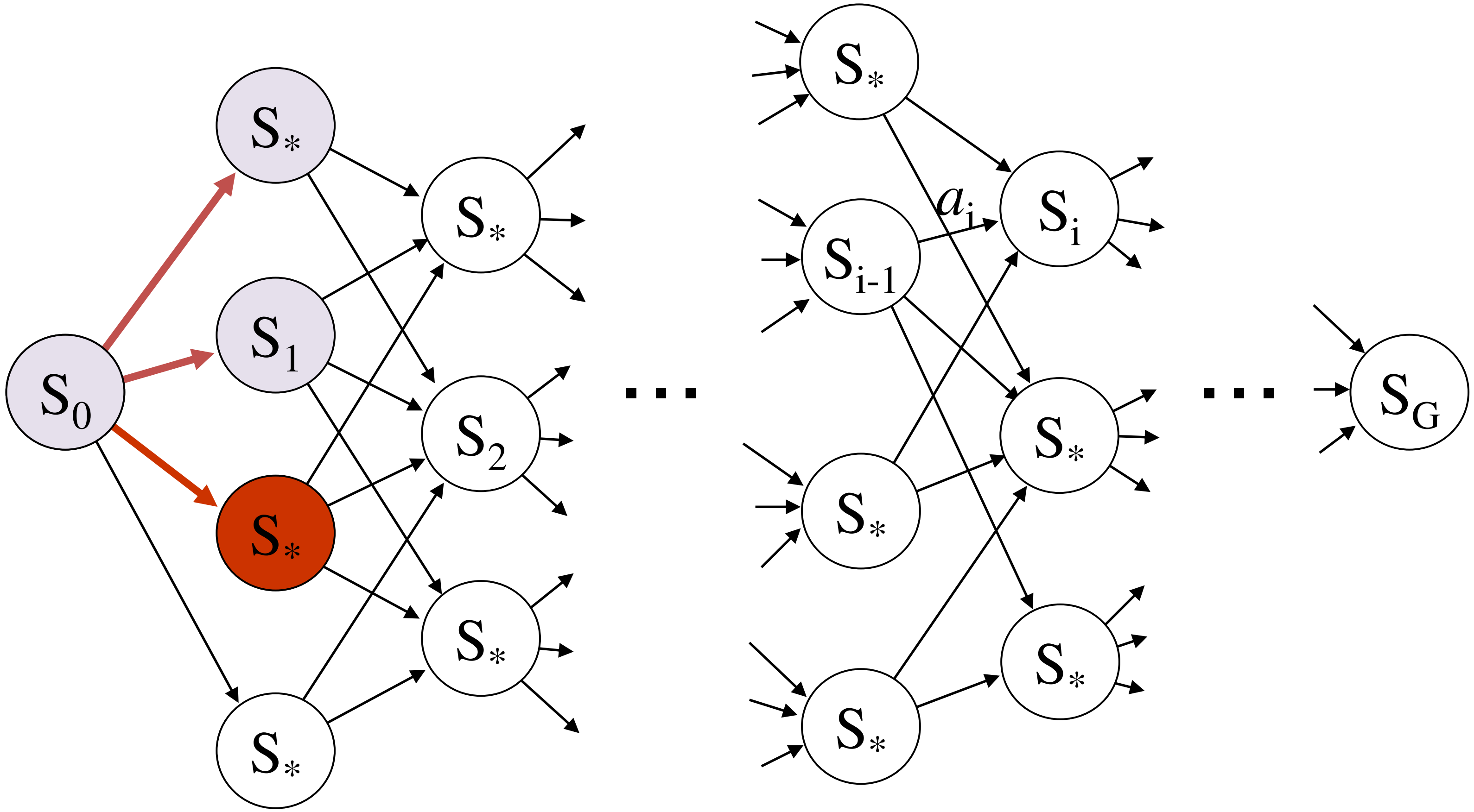
初期状態から経路長が１の状態への政策は一通りであるから，遷移先の状態においては最適政策はユニークに定まる。

動的計画法



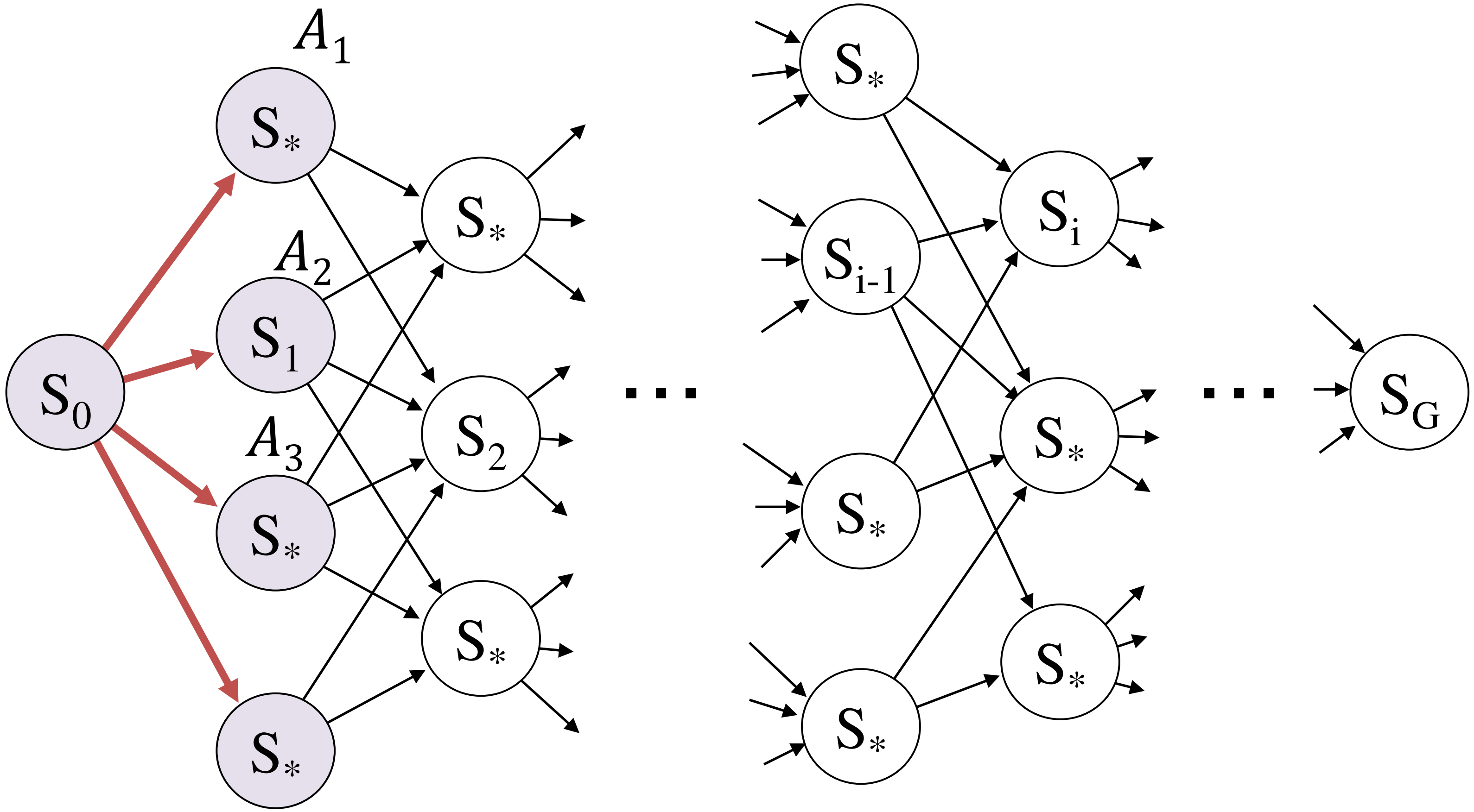
初期状態から経路長が１の状態への政策は一通りであるから，遷移先の状態においては最適政策はユニークに定まる。

動的計画法



初期状態から経路長が１の状態への政策は一通りであるから，遷移先の状態においては最適政策はユニークに定まる。

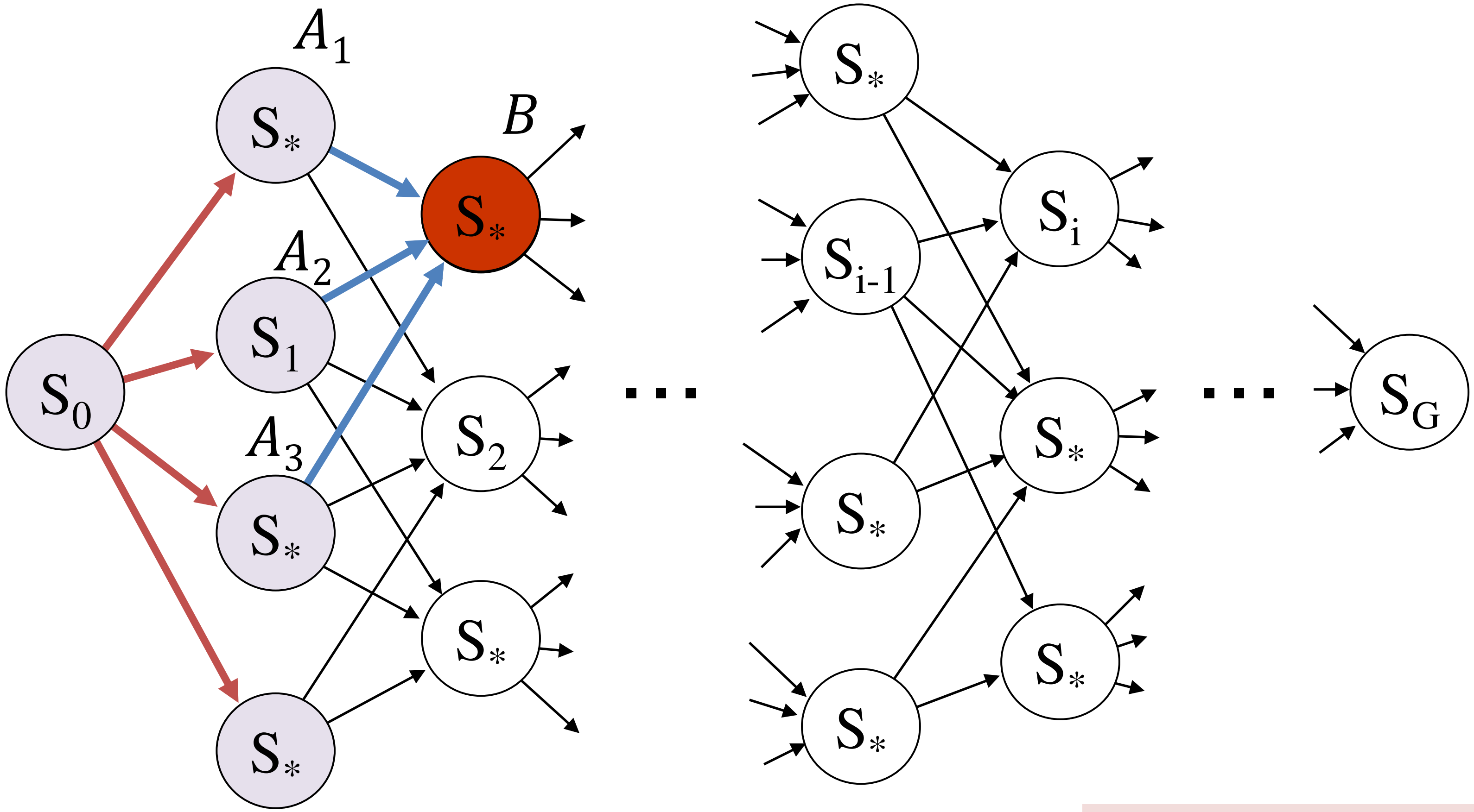
動的計画法



最適政策が既に定まった状態の変数として，図中の A_i が与えられたとする。

動的計画法

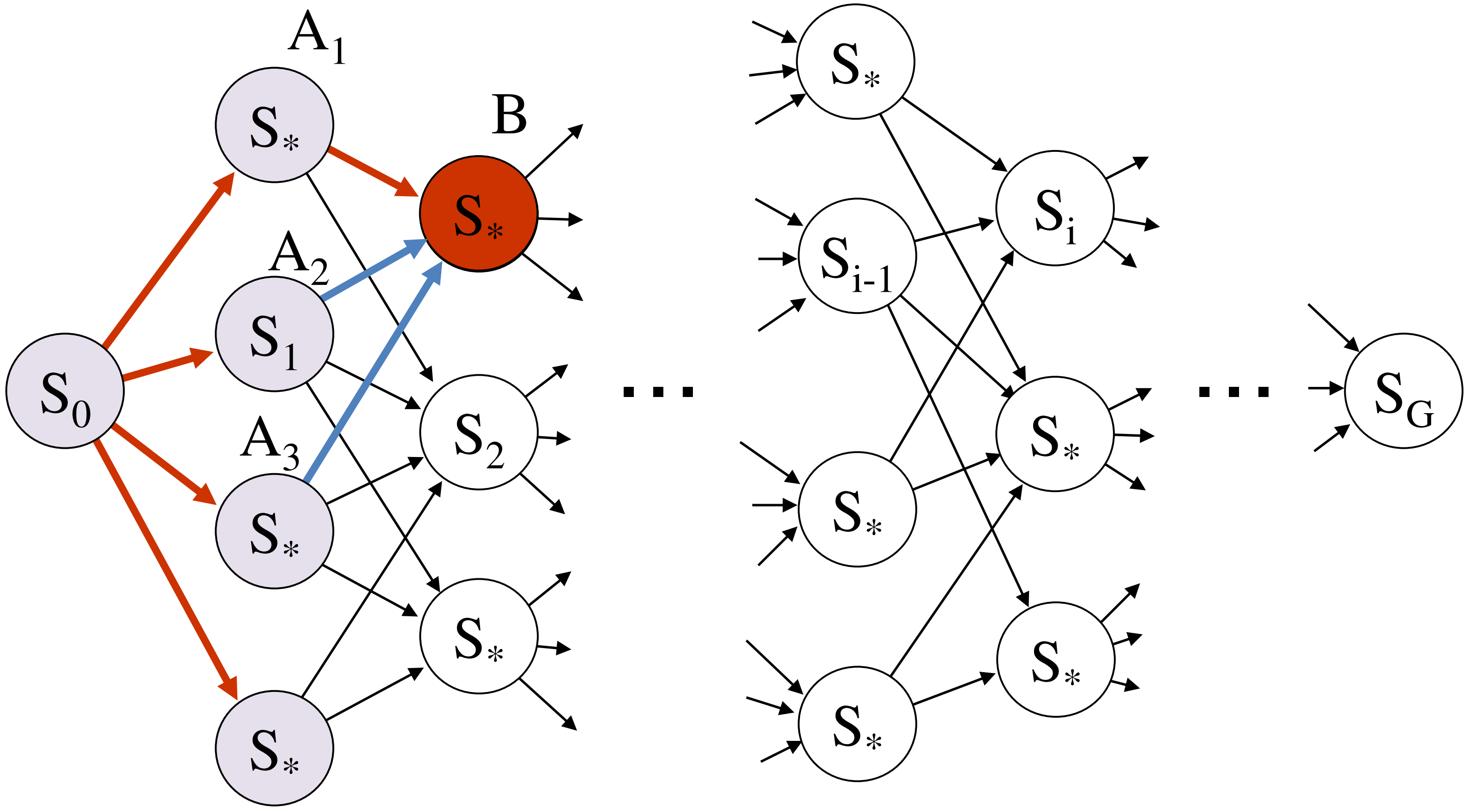
$$\text{OptimalCost}(\mathrm{S}_0 \to A_i \to B) = \text{OptimalCost}(\mathrm{S}_0 \to A_i) + \text{Cost}(A_i \to B)$$



A_i に続く状態 B への A_i 経由の最適政策のコストは，A_i への最適政策のコスト（➡）と，A_i から B へのコスト（➡）の和で与えられる。

動的計画法

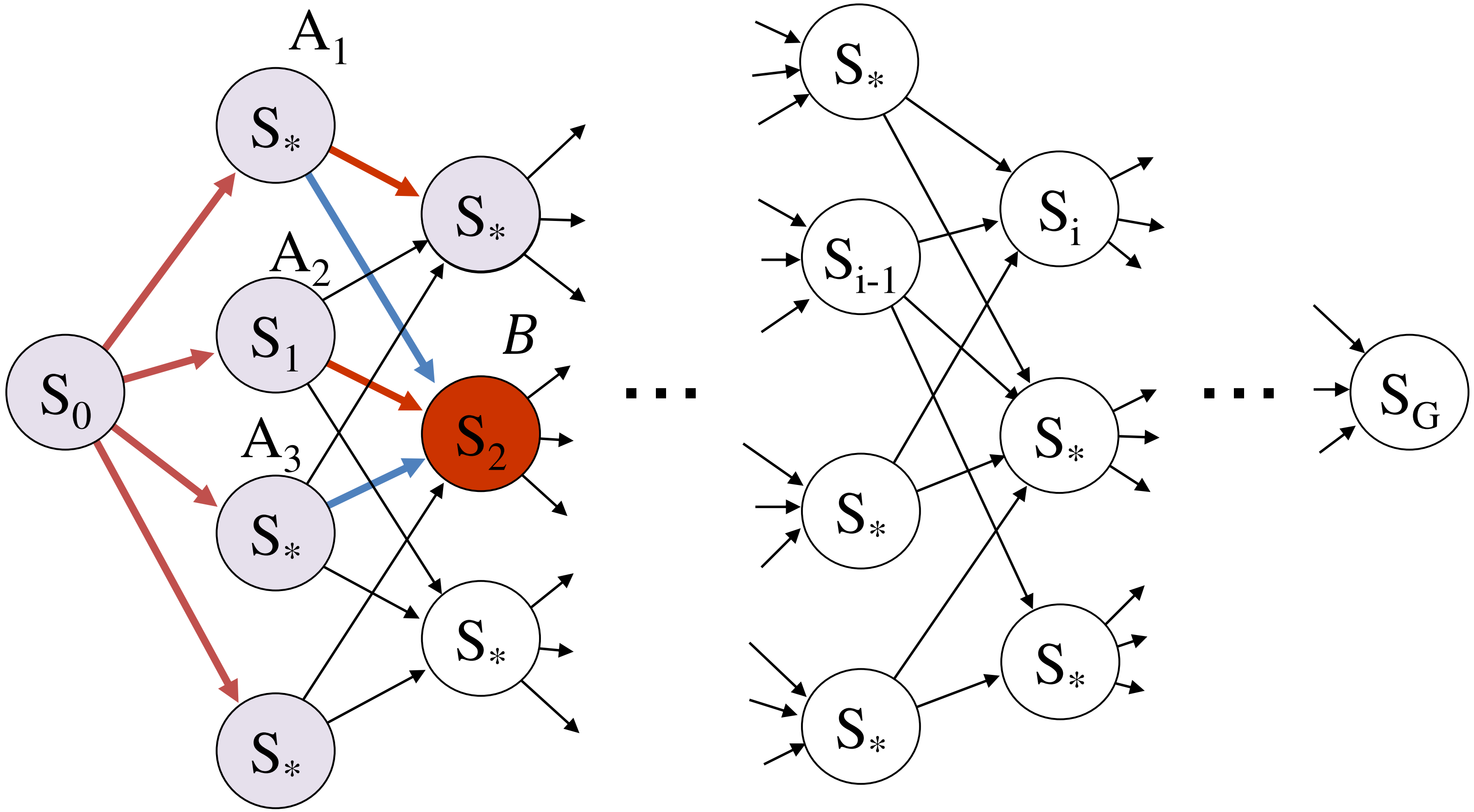
$$\mathrm{OptimalCost}(S_0 \to B) = \max[\mathrm{OptimalCost}(S_0 \to A_i \to B)]$$



B への最適政策は，A_i 経由での B への政策のコストのうち，最適値を与えた政策として与えられる。

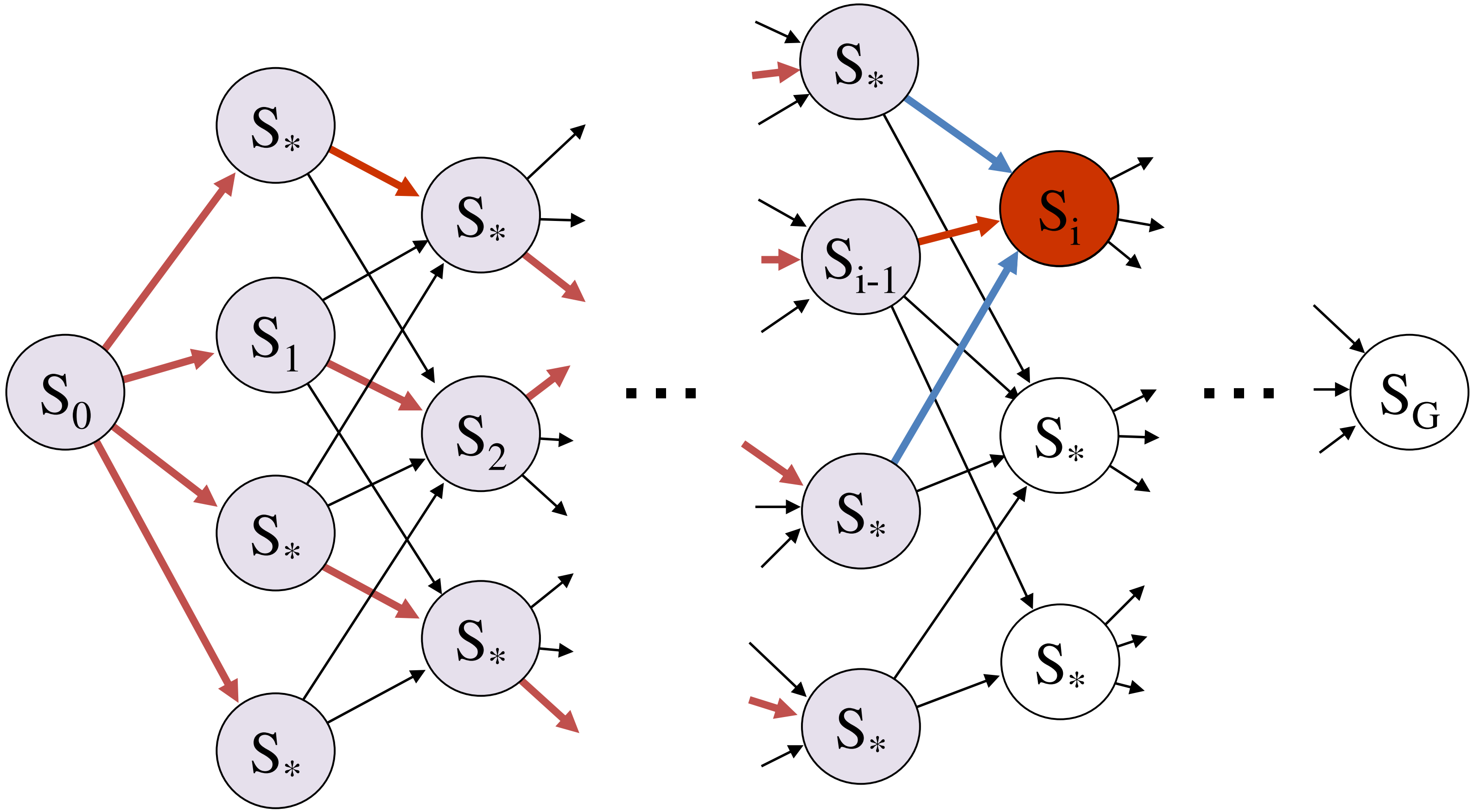
動的計画法

$$\mathrm{OptimalCost}(S_0 \rightarrow B) = \max[\mathrm{OptimalCost}(S_0 \rightarrow A_i \rightarrow B)]$$



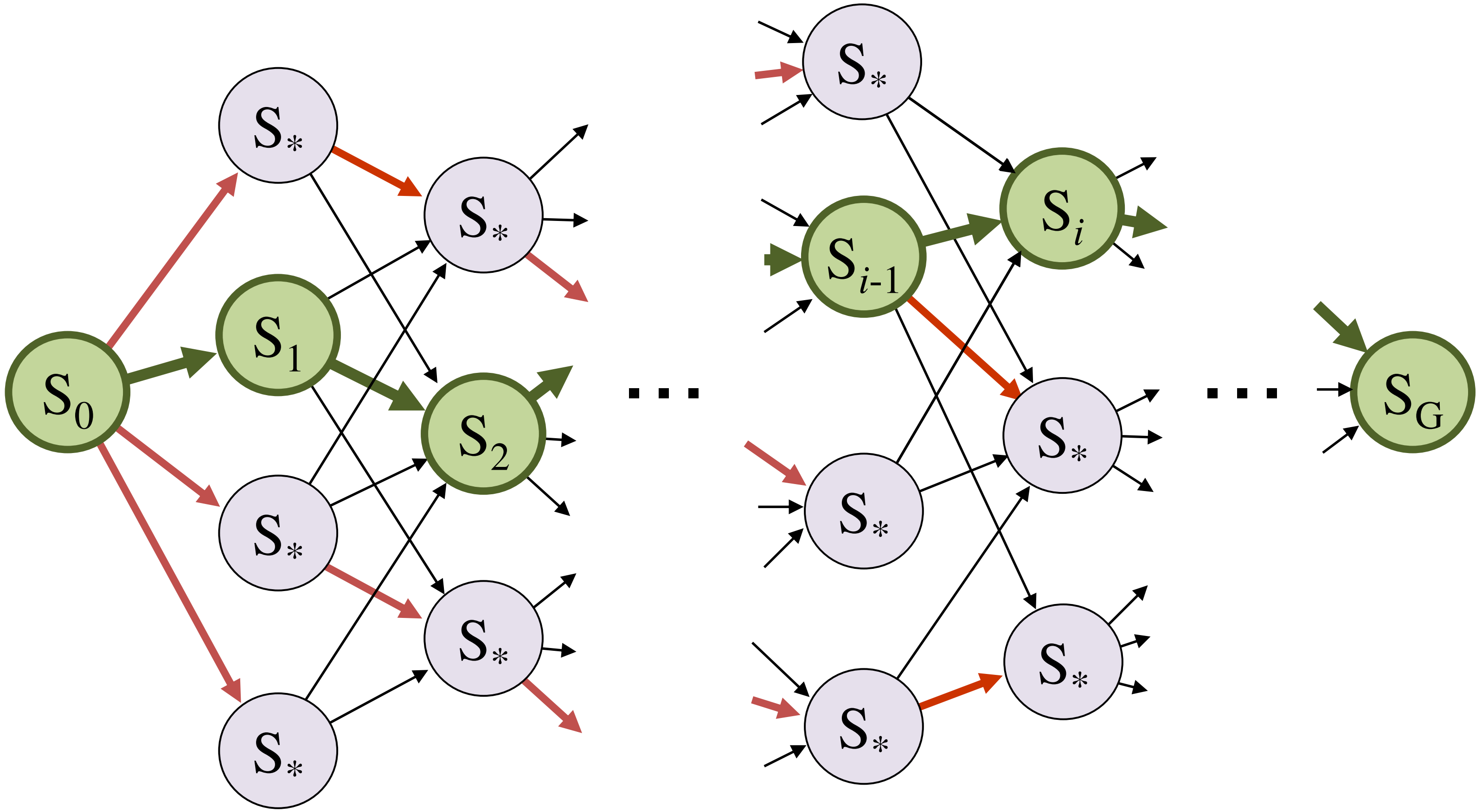
B への最適政策は，A_i 経由での B への政策のコストのうち，最適値を与えた政策として与えられる。

動的計画法



全ての状態への最適政策は，S_0 からの距離が短いものから，順に求めることができる。

動的計画法

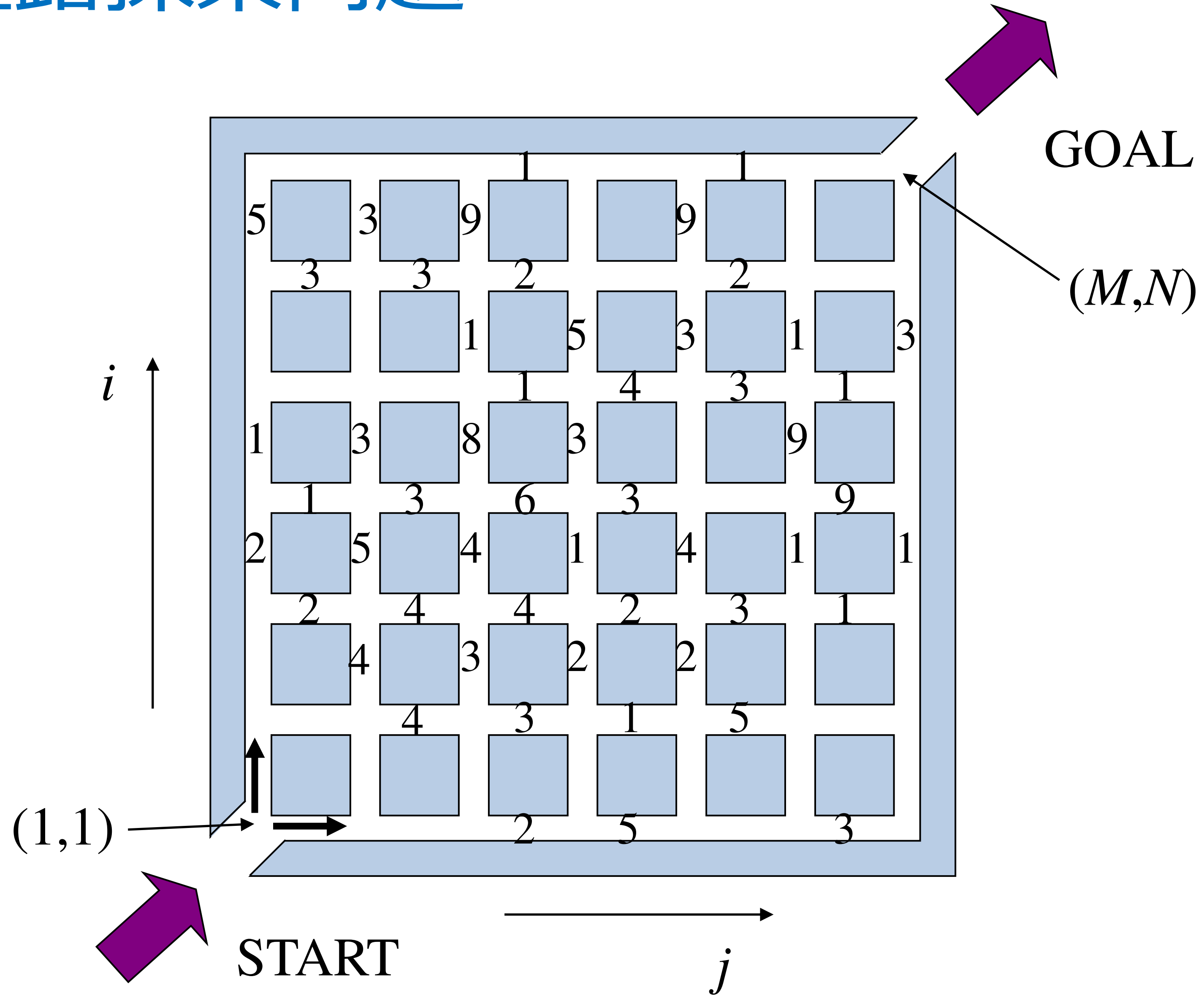


全体の最適政策は，それぞれの状態の最適値を与えた政策を，ゴールから，逆順に辿ることで得られる。

例題（１）：最適経路探索問題

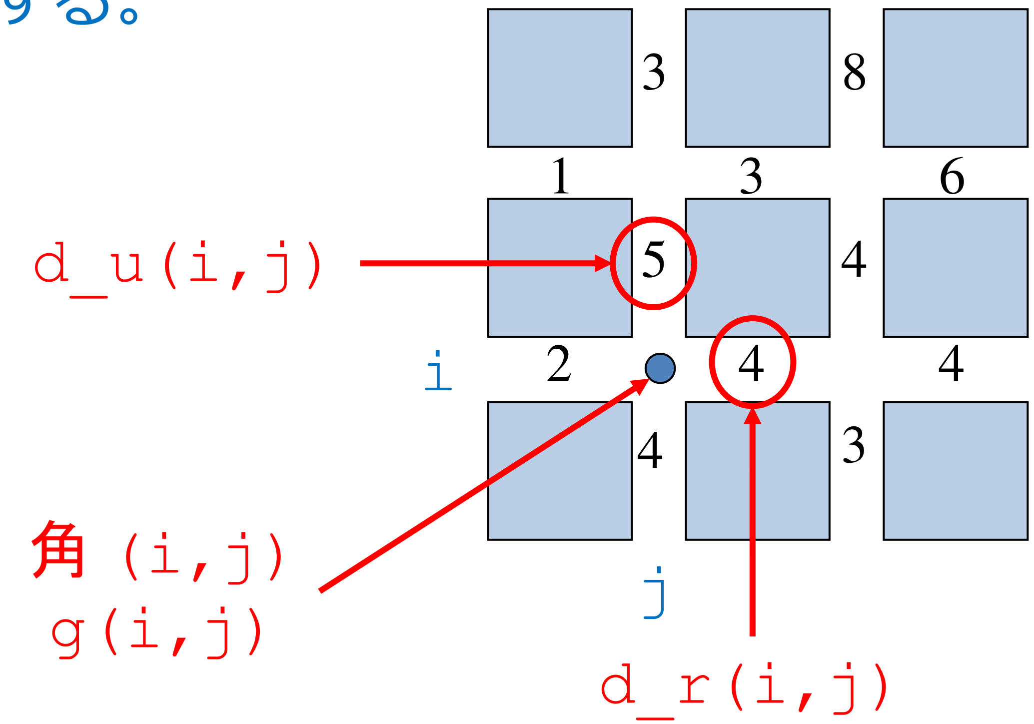
各パスに置かれた数字を収益と見て，総収益を最大にする経路を求める。

先に示した最適経路問題をDPを用いて解く方法について説明する。



最適経路探索問題

d_u(i,j)：角 (i,j) から上に進む道にある数字
d_r(i,j)：角 (i,j) から右に進む道にある数字
g(i,j)：　角 (i,j) に至る経路に沿って集めた数字の合計の最大値
とする。



d_u，d_r，g を上のように定義する。d_u および d_r は，下式（p.25から抜粋）の $\mathrm{Cost}(A_i \rightarrow B)$ に相当し，g は $\mathrm{OptimalCost}(S_o \rightarrow A_i)$ に相当する。

$$\mathrm{OptimalCost}(S_0 \rightarrow A_i \rightarrow B) = \mathrm{OptimalCost}(S_o \rightarrow A_i) + \mathrm{Cost}(A_i \rightarrow B)$$

最適経路探索問題

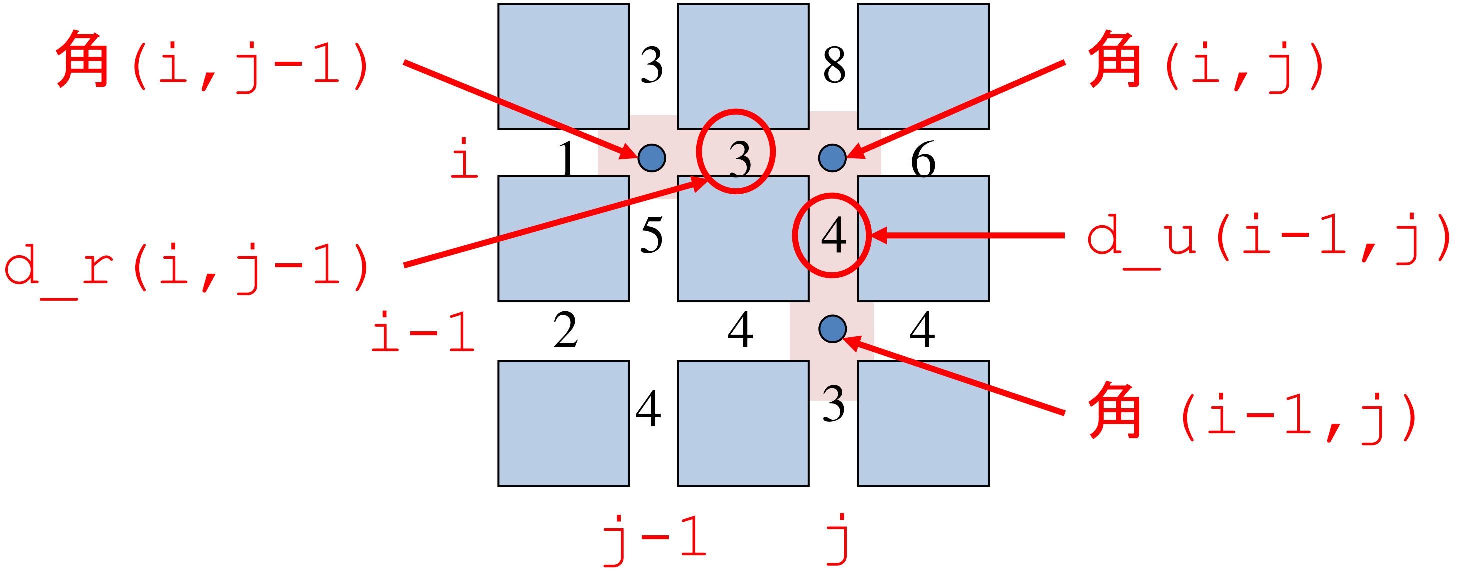
このとき，

(1,1)から(i,j-1)を経て(i,j)に至る経路における最適値

(1,1)から(i-1,j)を経て(i,j)に至る経路における最適値

g(i,j) = max(g(i,j-1)+d_r(i,j-1), g(i-1,j)+d_u(i-1,j))

なる関係が成立。



よって，(1,1) から (M,N) まで，g(i,j) を漸化的に求めれば，収益（経路上の数字の和）を最大にする経路を求めることができる。

例題（２）：文字列の距離

問題：二つの文字列 R，X が与えられたとする。文字列は，<waseda>，<kobayasi> のように，必ず，文字 “<” で始まり，文字 “>” で終わるものとする。二つの文字列の距離を次ページのように定義したうえで，文字列 R と文字列 X の距離を求めるアルゴリズムを考える。

1. 文字列 X の１文字目から i 文字目までの部分文字列と文字列 R の１文字目から j 文字目までの部分文字列の距離を alpha(j,i) とするとき，alpha(j,i) を漸化式の形で表せ。必要な変数，関数があれば，適宜定義して用いよ。

2. 文字列 X と文字列 R の距離を求めるアルゴリズム（プログラム）を記述せよ。

補足：文字列の違いの評価の難しさ

例えば，ABC と ABD の距離は，最後の C が D に変わっただけであり，距離 1 と考えるのは妥当であろう。しかし，ABCDE と ACDE の距離を考える際，そのまま出現順序にしたがって対応づけると，B → C，C → D，D → E，E → 空白 となり，距離は 4 となる。しかし，よく見ると，ABCDE の B が1つ脱落しただけあるから，距離は 1 と考えることもできる。

このように，２つの文字列の比較における「誤り方」の解釈には，複数の可能性がある。そこで，文字列の距離を求める際には，多数ある置換，脱落，挿入の誤りの組み合わせの中で，２つの文字列の距離が最も小さくなるような解釈を探した上で，距離を評価する。

①１文字の脱落があるとき，1の距離があるものとする。

例）R=<waseda125> に対し X=<wseda125> は，Rの3文字目の「a」が脱落しているので，距離1

②１文字の挿入があるとき，1の距離があるものとする。

例）R=<waseda125> に対し X=<waseida125> は，Xの6文字目に「i」が挿入しているので，距離1

③１文字の置換があるとき，1の距離があるものとする。

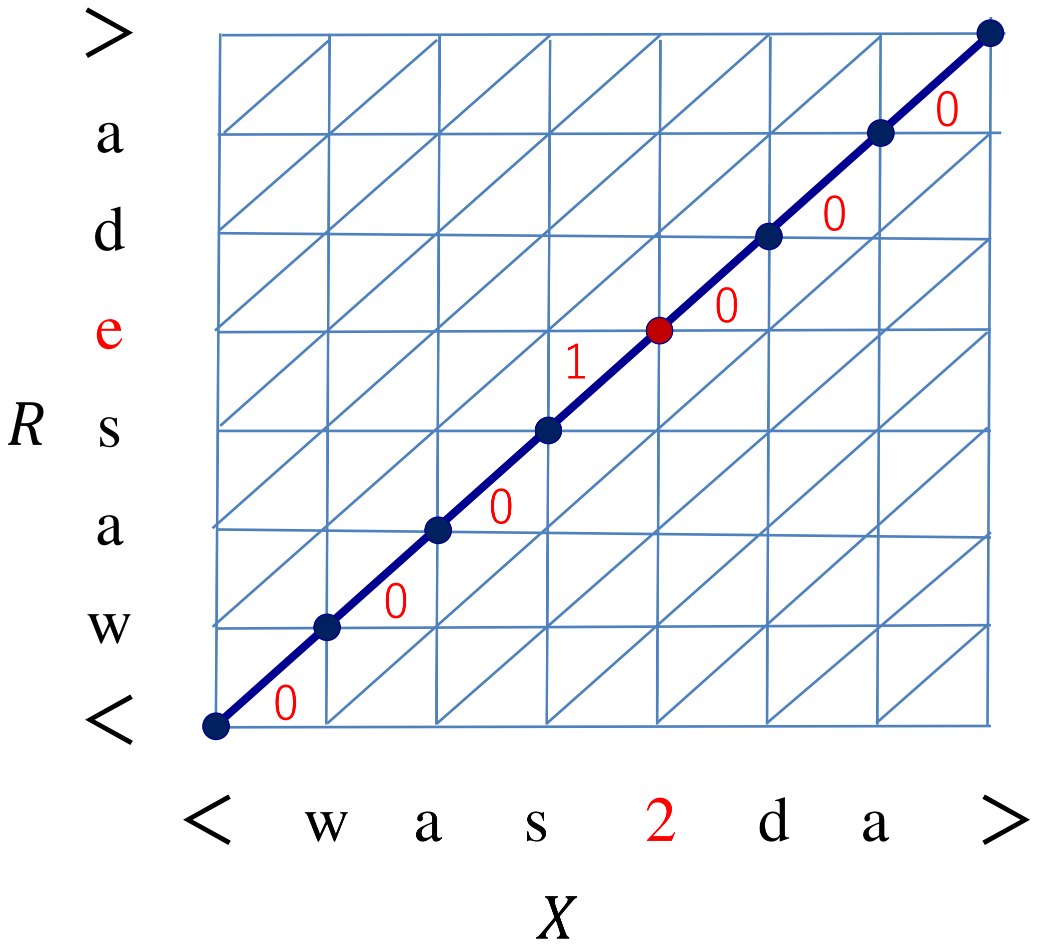
例）R=<waseda125> に対し X=<wasida125> は，Rの5文字目の「e」 が「i」に置換しているので，距離1

④ 上記の複合した現象は，それらの距離の和とする。ただし，複数の解釈に対しては，最も距離を小さくる解釈を採用する。

例）R=<waseda125> に対し X=<was2da125> は，5文字目の英字「e」 が数字「2」に置換しているとも，5文字目の英字「e」が脱落し，同じ場所に数字「2」が挿入したとも解釈できるが，前の解釈では距離1，後の解釈では距離2となるため，前の解釈を採用する。

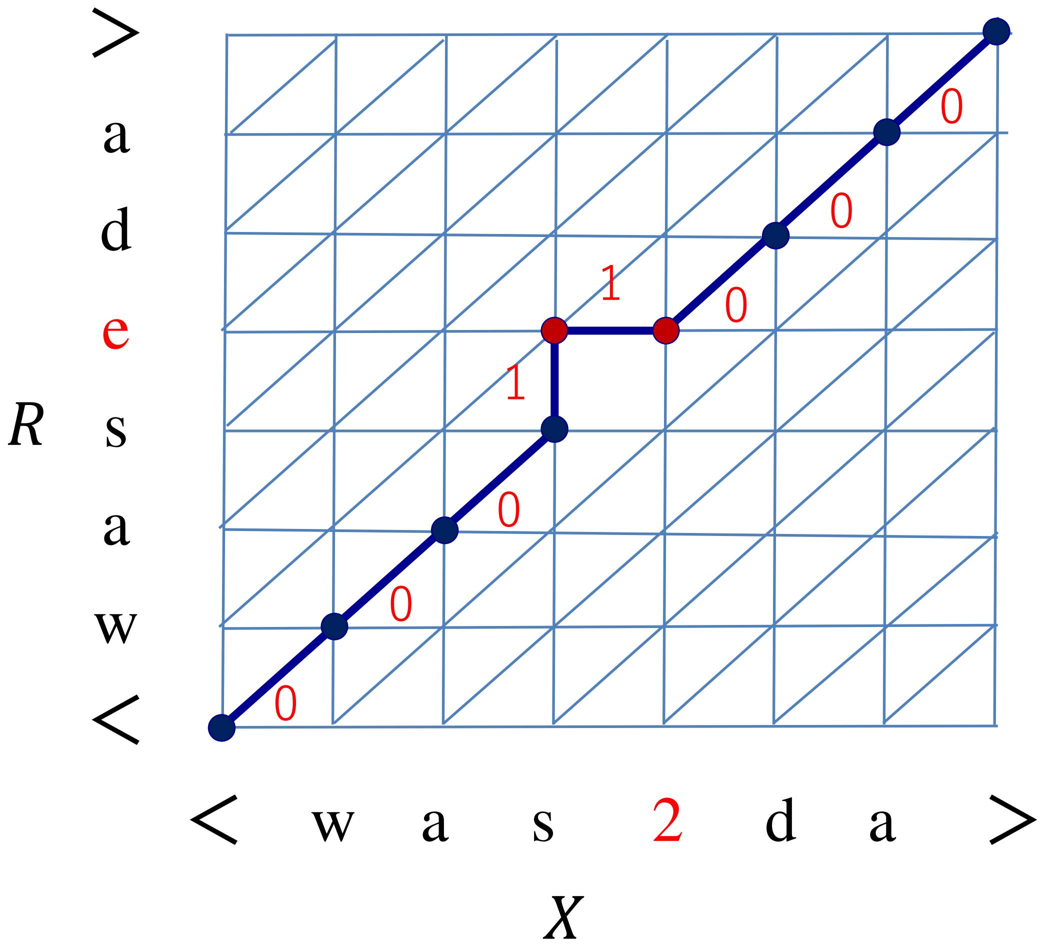
「e」が「2」に置換

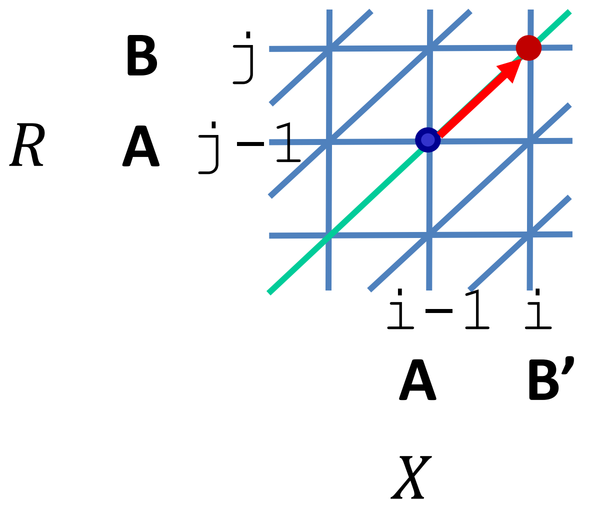
距離 ＝ 1

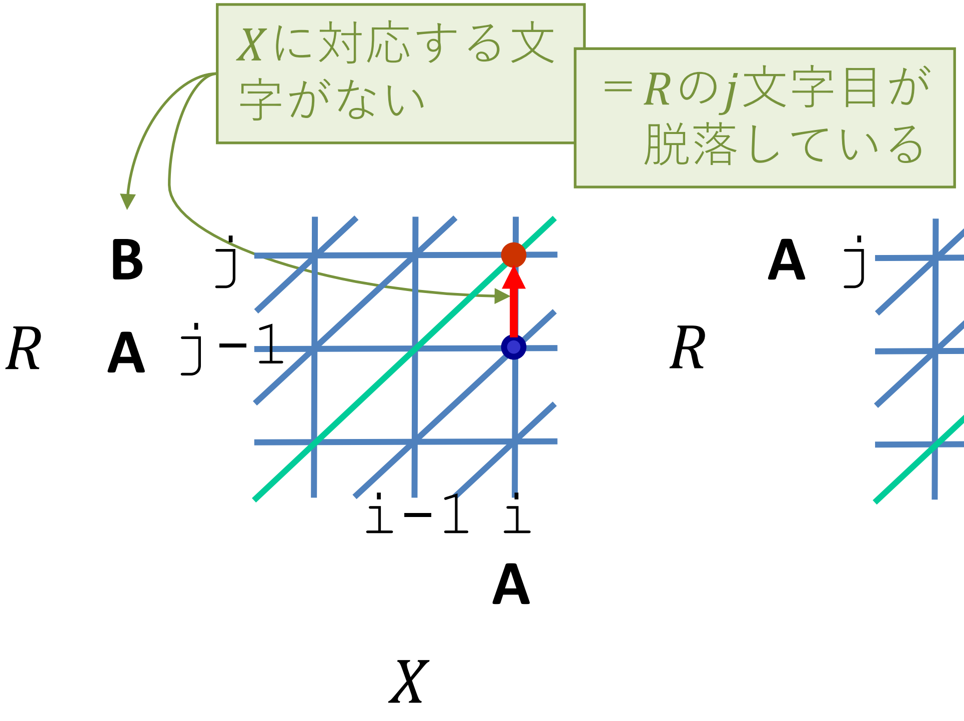


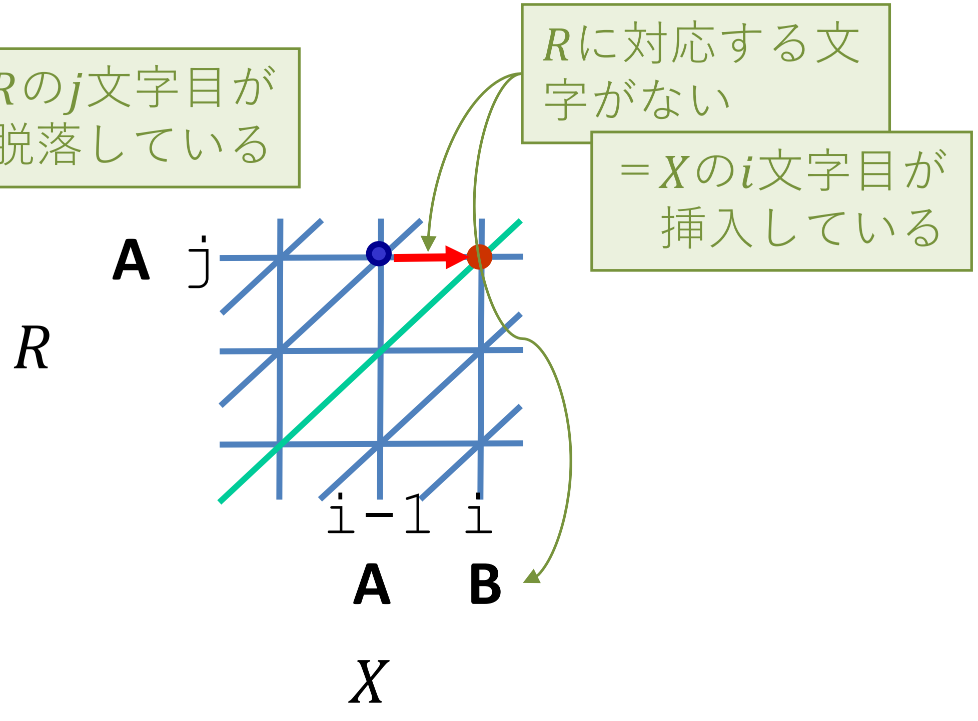
「e」が脱落 ＋「2」が挿入

距離 ＝ 2










```
subCost(X(i),R(j)) =
  1 (X(i) != R(j)),
  0 (X(i) == R(j))
```

斜めに進むパスでは終端の文字が異なれば距離１を加算

置換

```
delCost( ) = 1
```

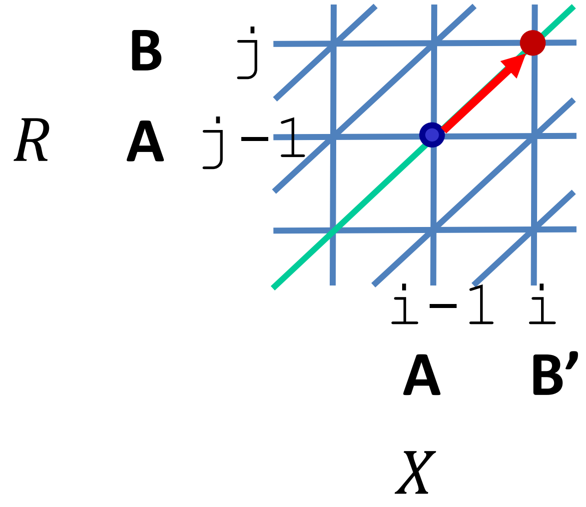
上に進むパスでは距離１を加算

脱落

```
insCost( ) = 1
```

右に進むパスでは距離１を加算

挿入




```
subCost(X(i),R(j)) =
  1 (X(i) != R(j)),
  0 (X(i) == R(j))
```

delCost( ) = 1

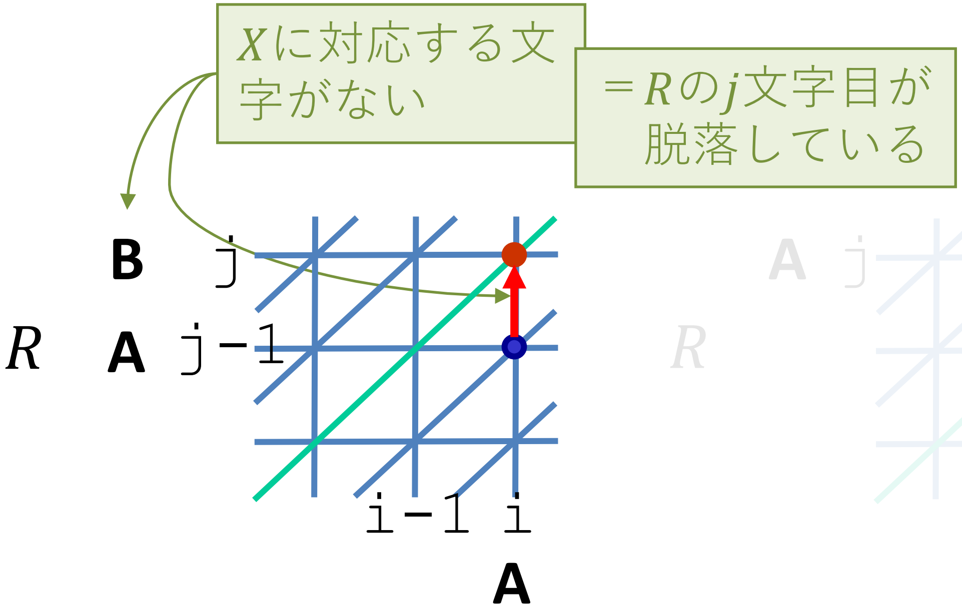
insCost( ) = 1

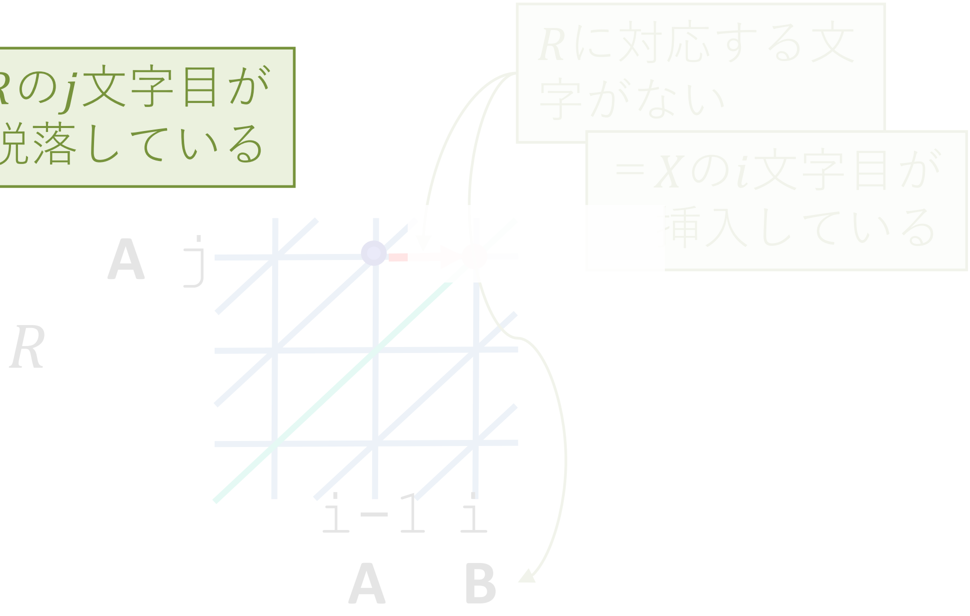
斜めに進むパスでは終端の文字が異なれば距離１を加算

上に進むパスでは距離１を加算

右に進むパスでは距離１を加算

置換　**脱落**　**挿入**






```
subCost(X(i),R(j)) =
  1 (X(i) != R(j)),
  0 (X(i) == R(j))
```

```
delCost( ) = 1
```

```
insCost( ) = 1
```

斜めに進むパスでは終端の文字が異なれば距離１を加算

上に進むパスでは距離１を加算

右に進むパスでは距離１を加算

置換

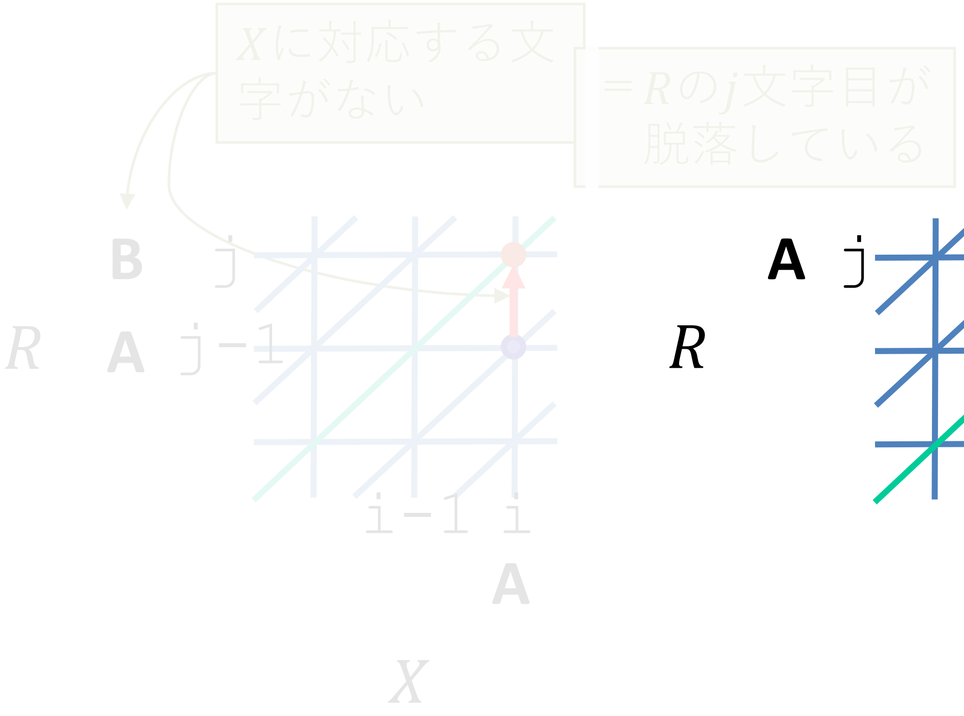
脱落

挿入

```
subCost(X(i),R(j)) =
  1 (X(i) != R(j)),
  0 (X(i) == R(j))
```

斜めに進むパスでは終端の文字が異なれば距離１を加算

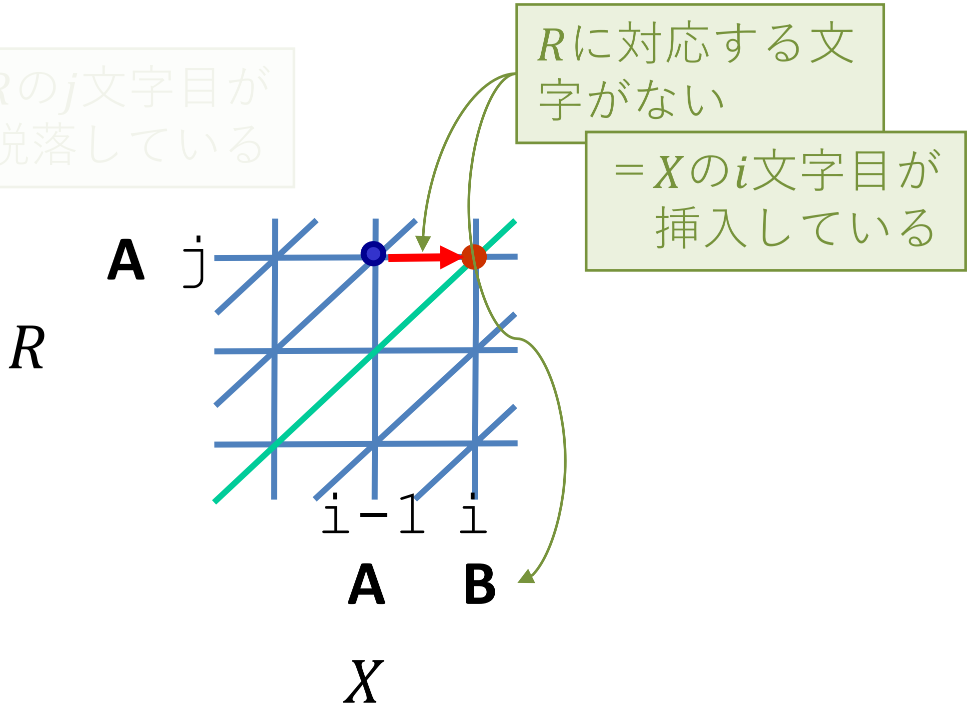
置換




```
delCost( ) = 1
```

上に進むパスでは距離１を加算

脱落




```
insCost( ) = 1
```

右に進むパスでは距離１を加算

挿入

解答

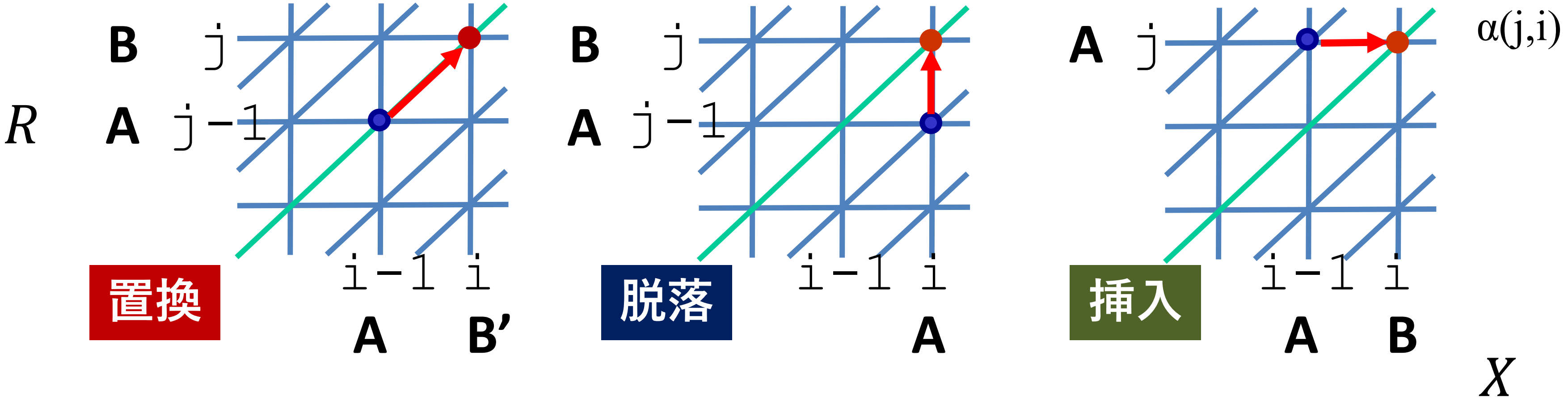
```
1) alpha(j,i)  = min( alpha(j-1,i-1) + subCost(X(i),R(j)),
                      alpha(j-1,i) + delCost(),
                      alpha(j,i-1) + insCost() )
```

ただし,

subCost(a,b)：文字 a を文字 b に誤るコスト

delCost()：R の1文字が脱落する（R にあるものが X にない）コスト

insCost()：X の1文字が挿入する（R にないものが X にある）コスト

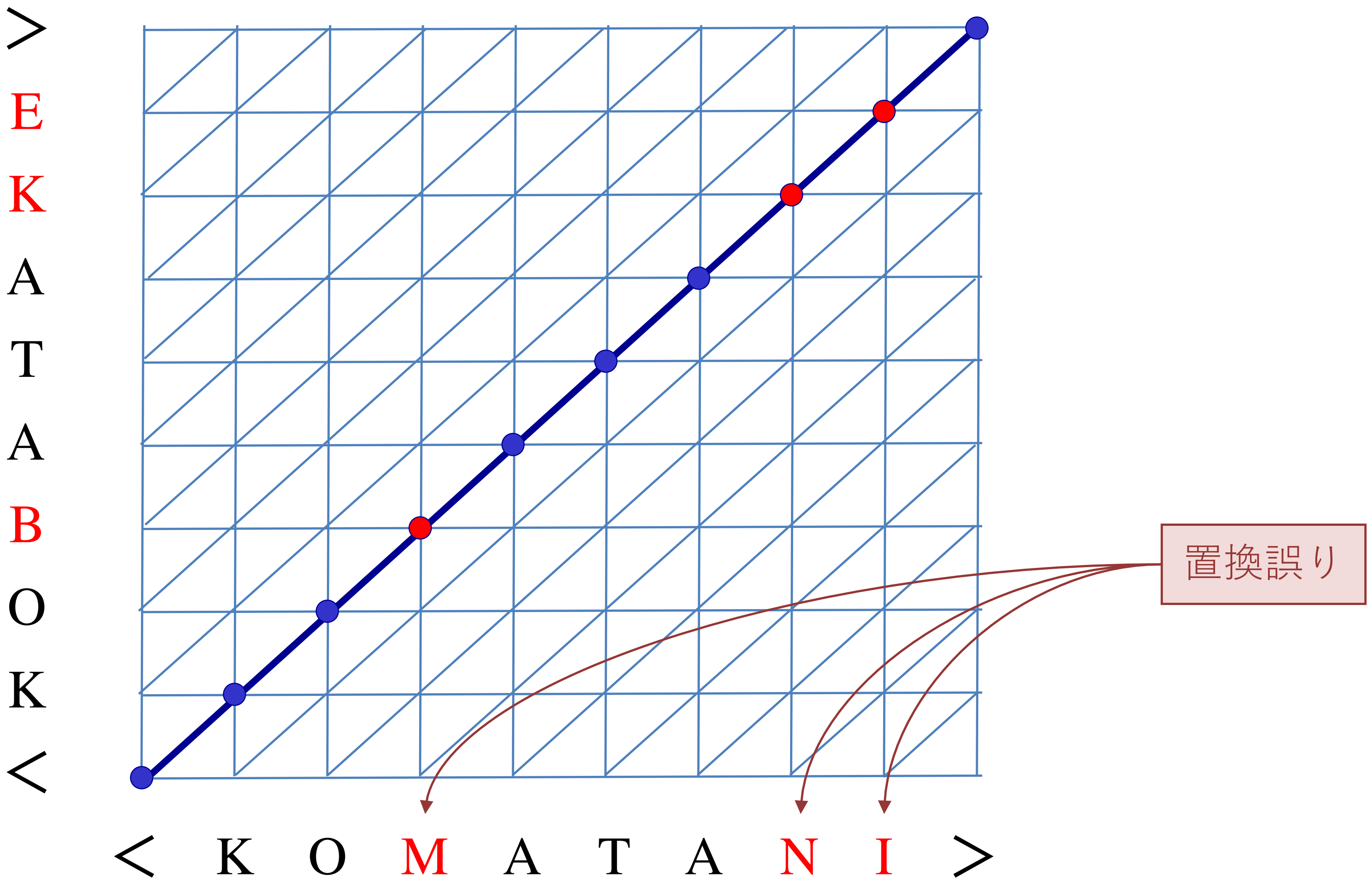


2)

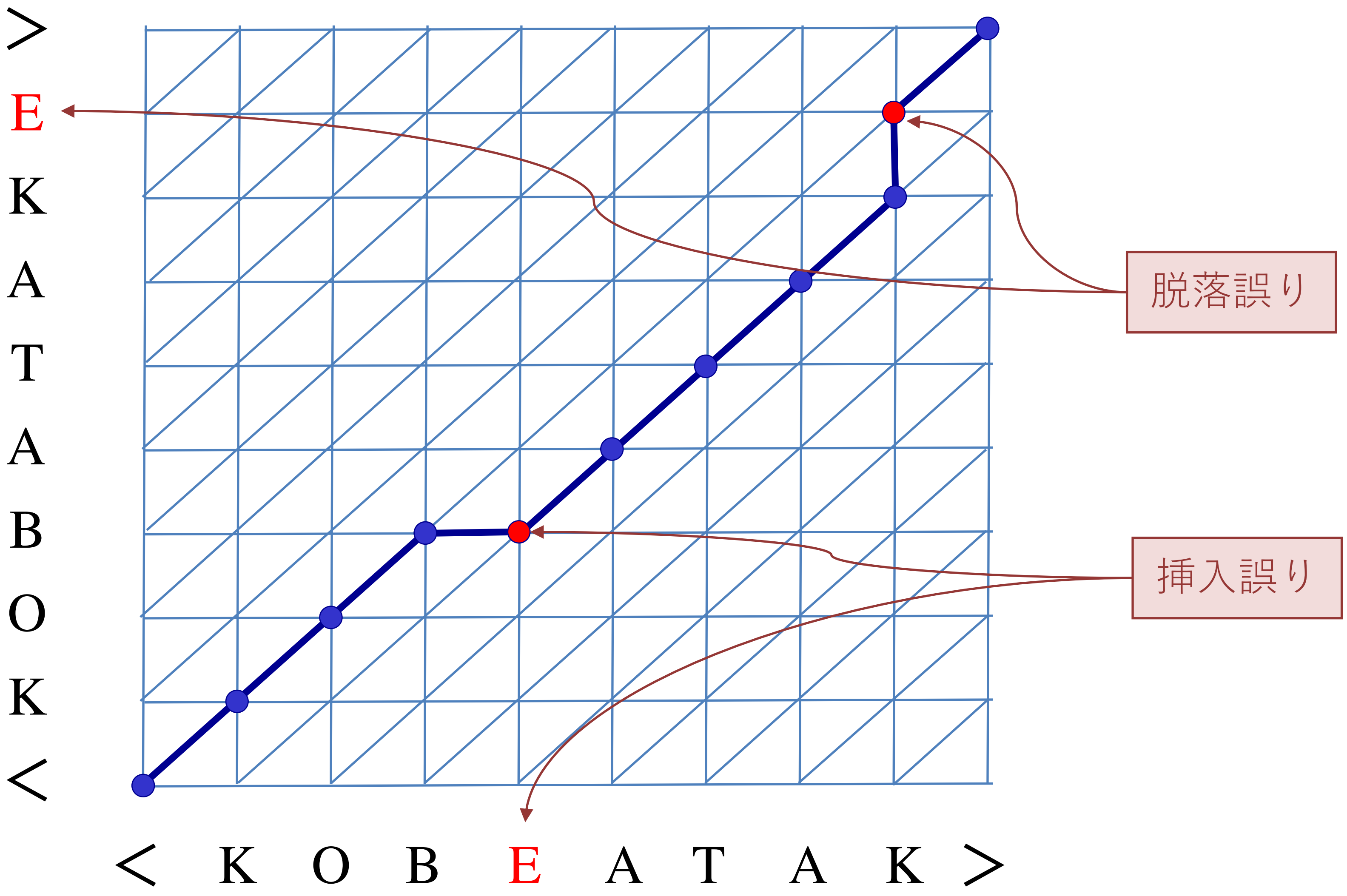
```
import sys
def localCost(a,b):
        if a==b :
                return 0
        else:
                return 1
def delCost1c():
        return 1
def insCost1c():
        return 1
argvs = sys.argv
X = '<'+argvs[1]+'>'
R = '<'+argvs[2]+'>'
lenX = len(X)
lenR = len(R)
```

```
alp = [[float("inf") for i in range(lenX)] for j in
range(lenR)]
alp[0][0] = 0
for i in range(1,lenX):
        alp[0][i] = alp[0][i-1] + insCost1c()
for j in range(1,lenR):
        alp[j][0] = alp[j-1][0] + delCost1c()
        for i in range(1,lenX):
                a = alp[j][i-1] + insCost1c()
                b = alp[j-1][i-1] + localCost(X[i],R[j])
                c = alp[j-1][i] + delCost1c()
                alp[j][i] = min(a,b,c)
print('Distance : ', alp[lenR-1][lenX-1])
```

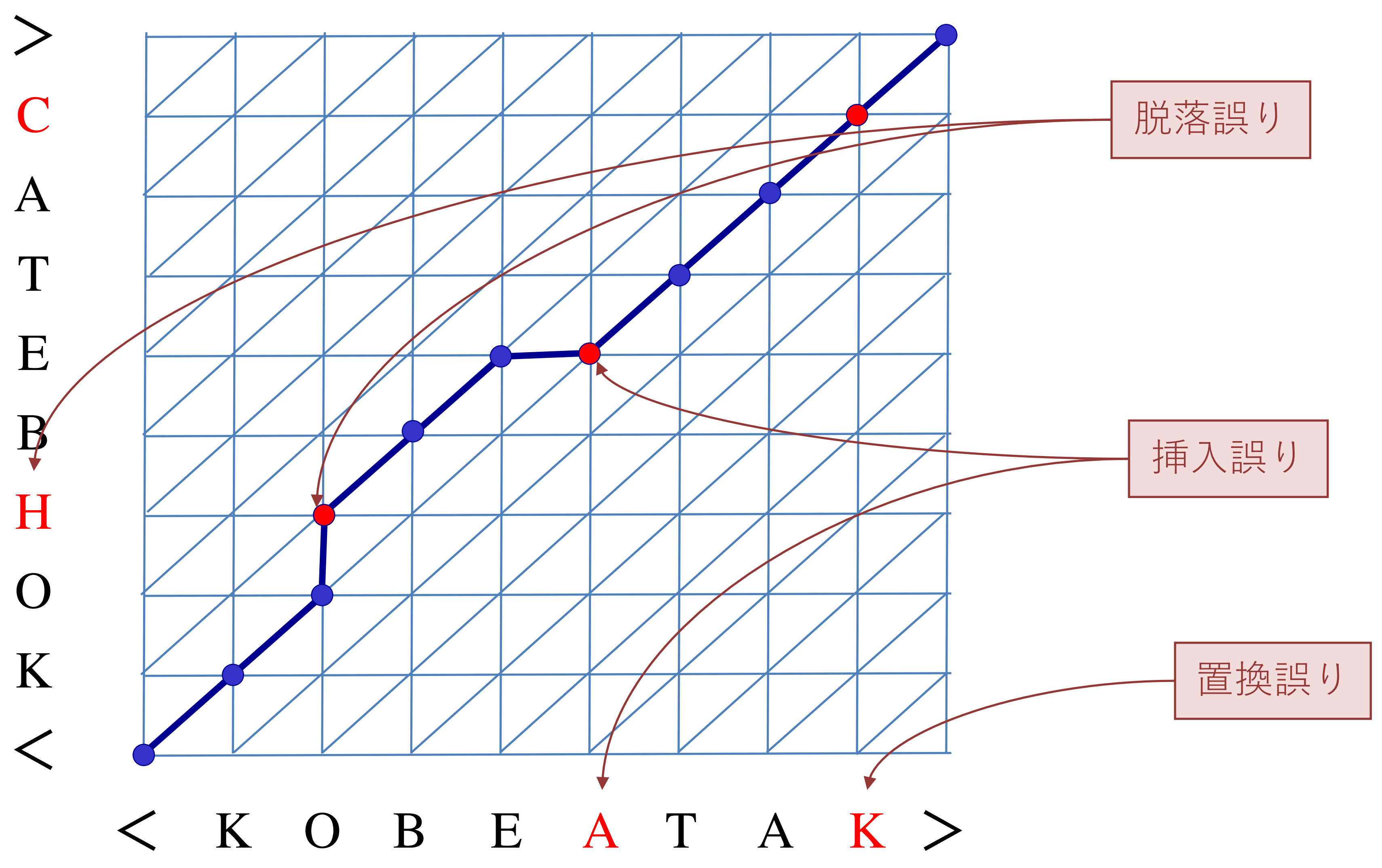
例）<KOBATAKE>（正解文字列）と <KOMATANI>（入力文字列）の距離 ＝ 3



例）<KOBATAKE>（正解文字列）と <KOBEATAK>（入力文字列）の距離 = 2



例）<KOHBETAC>（正解文字列）と <KOBEATAK>（入力文字列）の距離 = 3



例題（３）：配分問題

$$\max\left[\sum_{i=1}^{N} g_i(x_i)\right] \qquad \mathrm{s.t.} \quad \sum_{i=1}^{N} x_i = K \ \ (Const.), \qquad x_i > 0 \text{ なる整数}$$

総量 K の資源と，N 個の工場があり，工場 i は資源 x_i をもとに収益 $g_i(x_i)$ をあげることができるものとする。このとき，収益の合計を最大化する資源の分配の方法を求める。

配分問題を動的計画法を用いて解く

n 個の工場で，資源 k を用いてあげることのできる収益の最大値を $f_n(k)$ とする。

このとき，次の漸化式が成立する。

$$f_n(k) = \max_{0 \leq x_n \leq k} [f_{n-1}(k - x_n) + g(x_n)]$$

$$f_1(k) = g(k)$$

$n = 1,2,\cdots,N$， $k = 1,2,\cdots,K$ まで順に漸化式をとき，$f_N(K)$ を求める。

まとめ

- 説明変数が離散値をとる最適化の問題を，離散最適化問題，組み合わせ最適化問題と呼ぶ。
- 逐次決定過程は，組み合わせ最適化の代表的問題である。
- 動的計画法は，逐次決定過程の最適化を行う代表的手法である。
- 動的計画法は，ベルマンの最適化の原理を，小さな部分問題に適用するところから始めて，徐々に大きな問題へと繰り返し漸化的に適用することで，大きな最適化の問題を解く方法である。