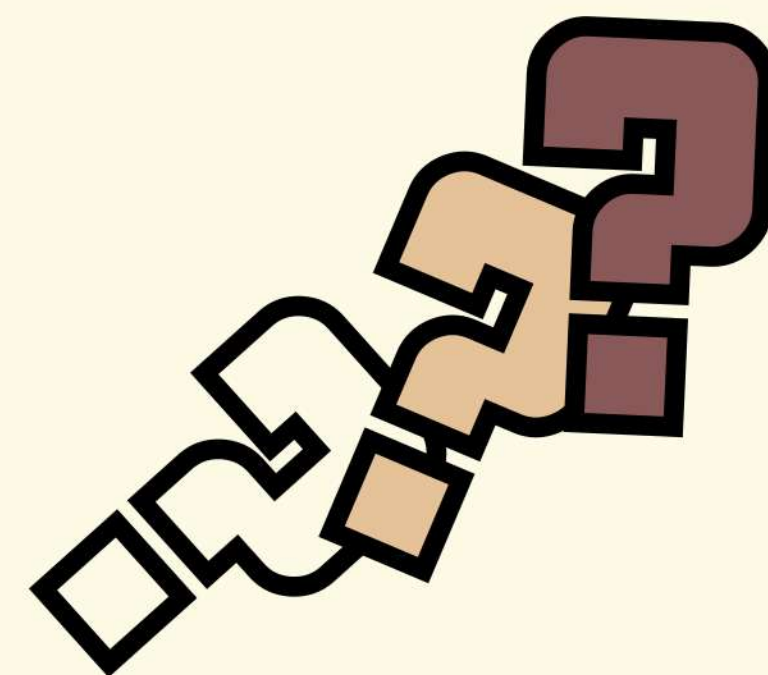


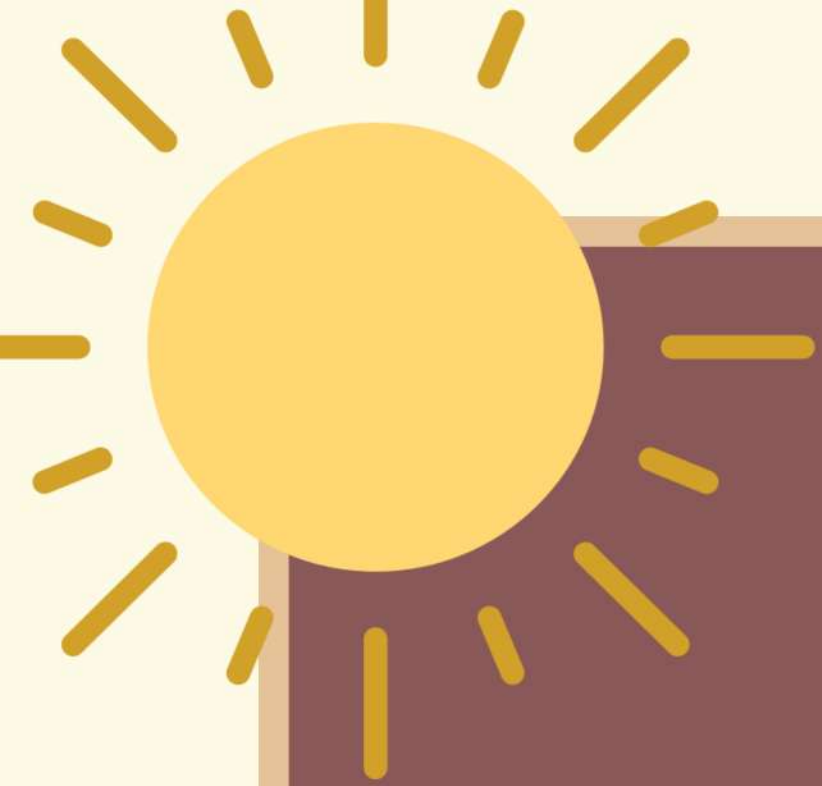


最佳化問題

第8組

410831209李博勛
411231108張明耀
411231124黃昱程
411231140劉恆彬
411231220曾億守

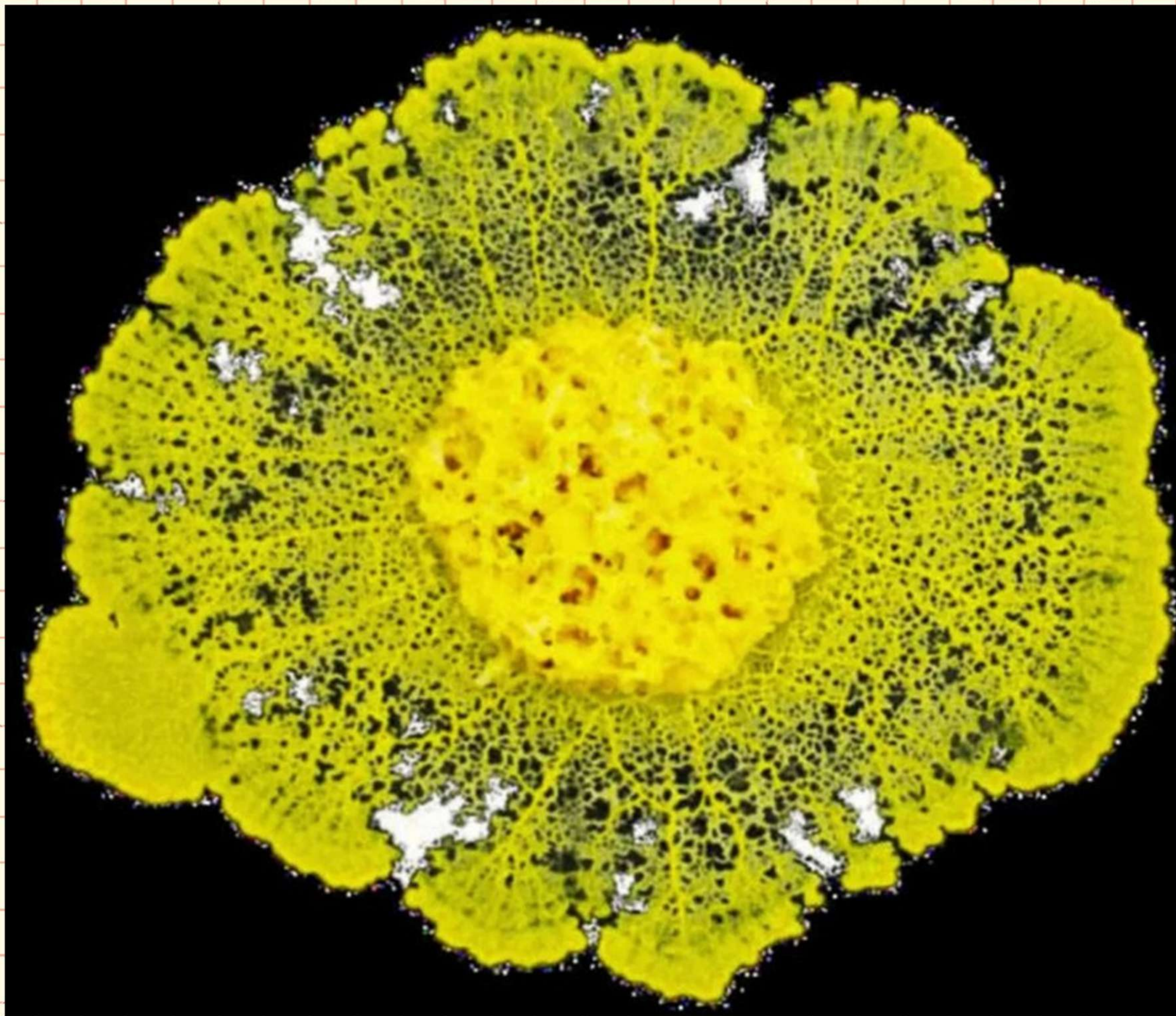




自然界啟發的最佳化現象

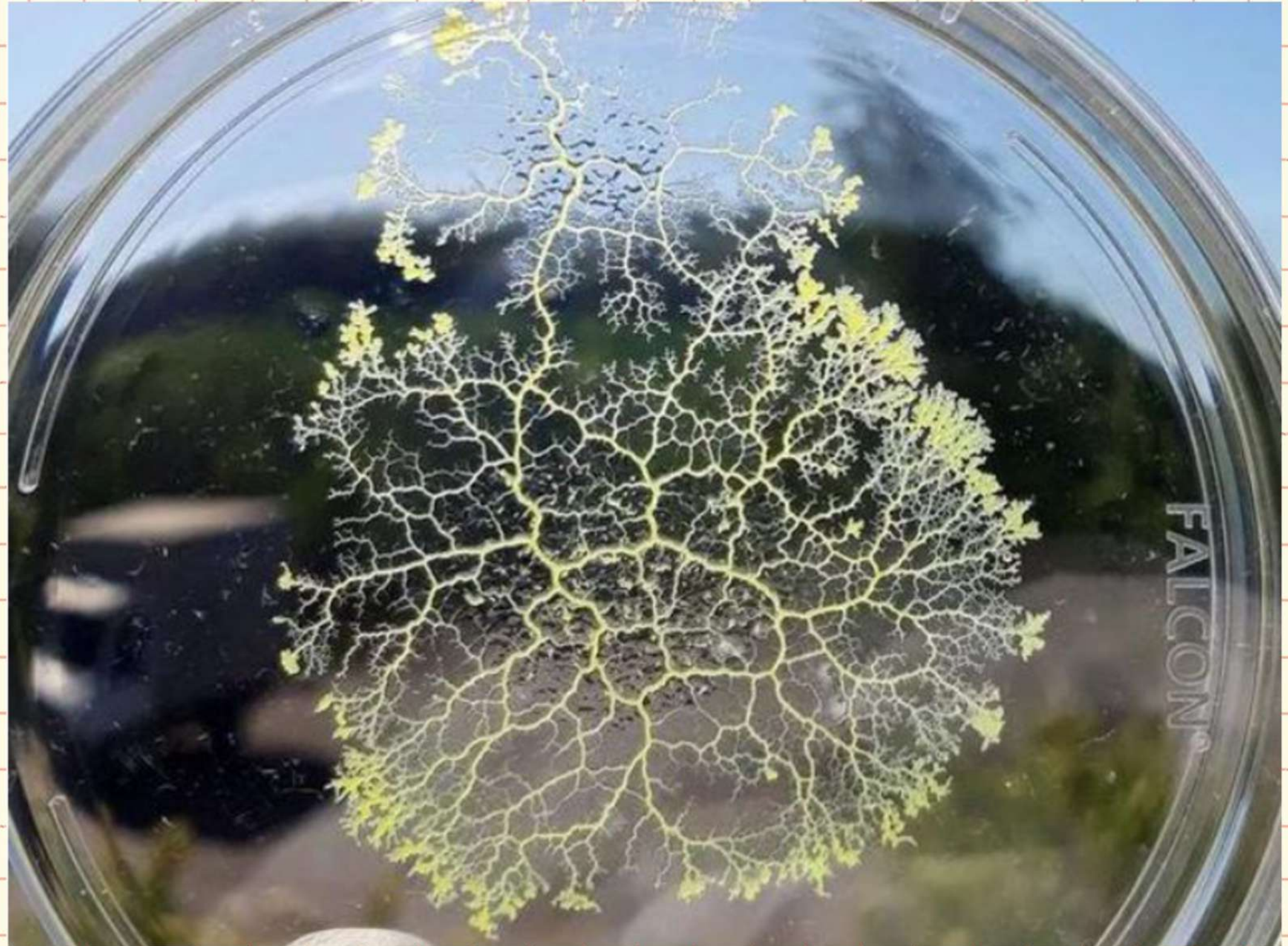


「最佳化問題」自古希臘時代以來，即在在人類追求知識與效率的過程中扮演關鍵角色。從傳統數學理論到當今的電腦運算與大數據技術，人類持續追求簡單、高效且普適的方法，以尋找問題的最優解。最佳化問題充斥於日常生活當中，如我們最常使用的導航，能為了覓食而規劃出最佳路徑的黏菌，以及資本家趨之若鶩的問題，"如何生產才最賺錢"，皆與最佳化問題息息相關，此次報告將由淺入深與大家分享，一探大家日常生活中或多或少所接觸到的最佳化問題。



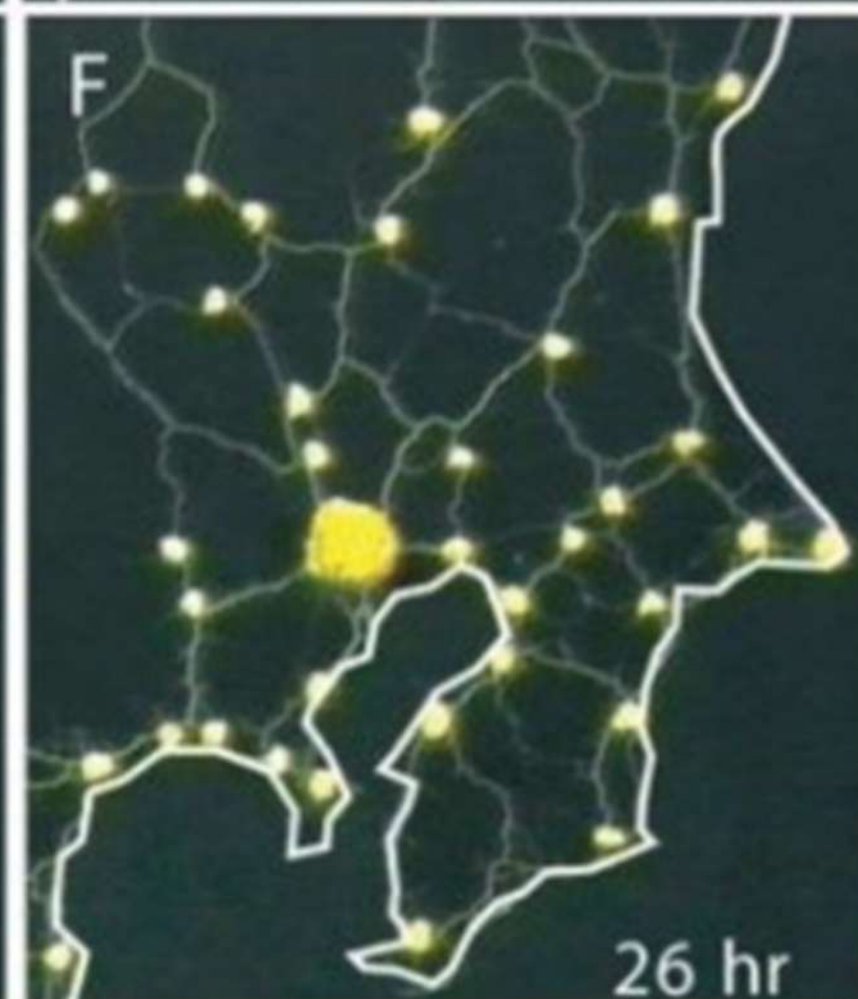
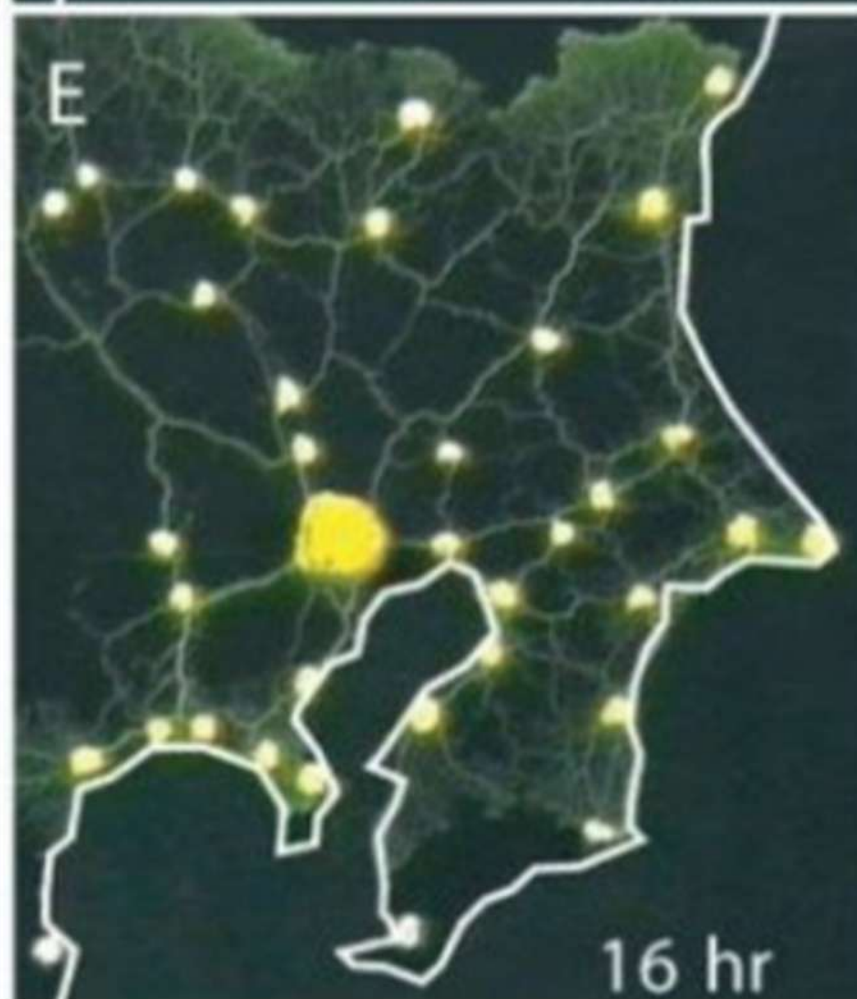
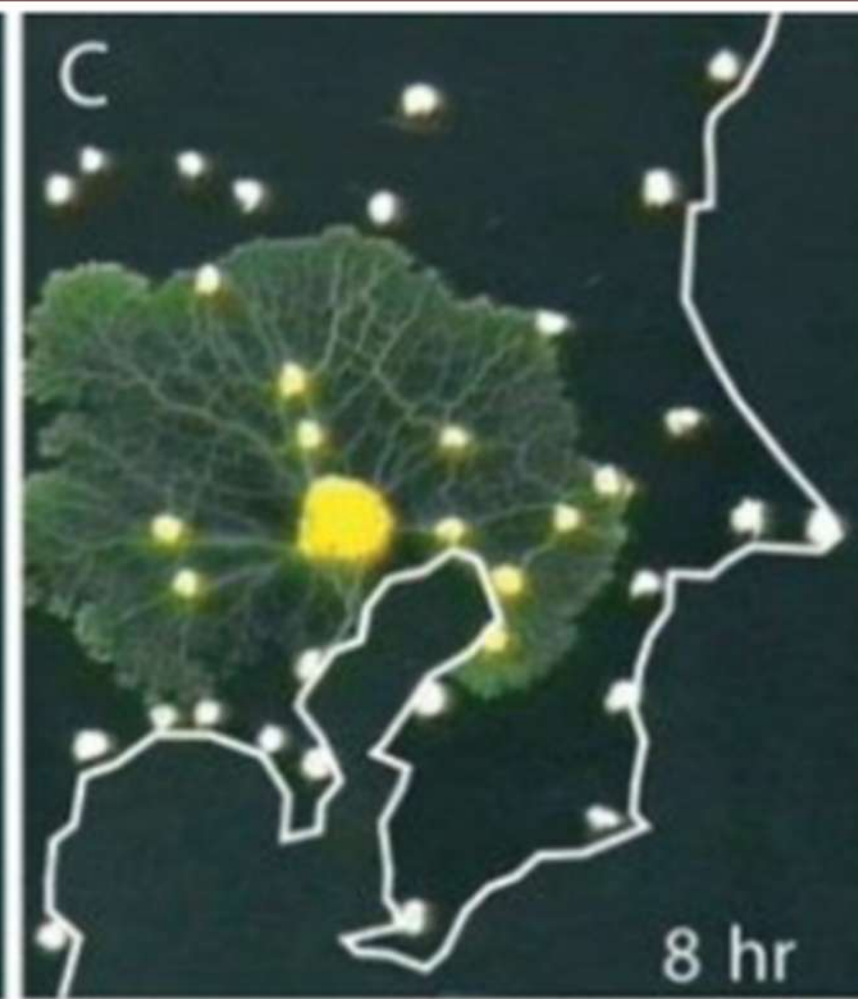
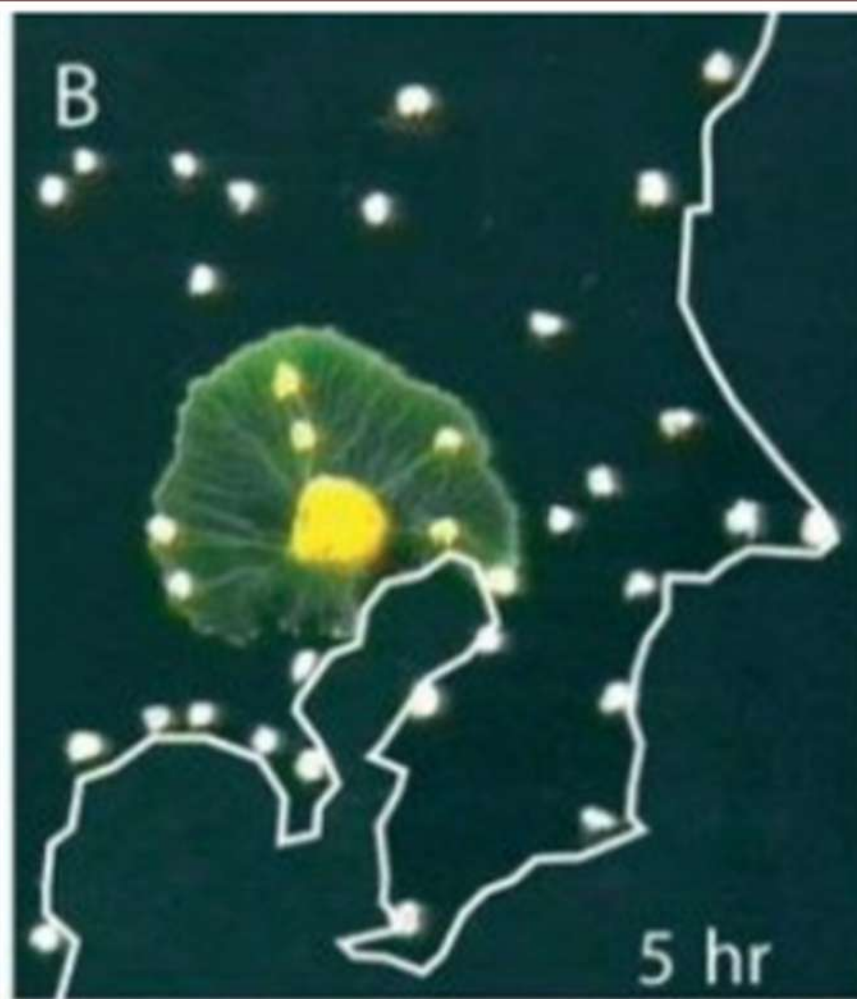
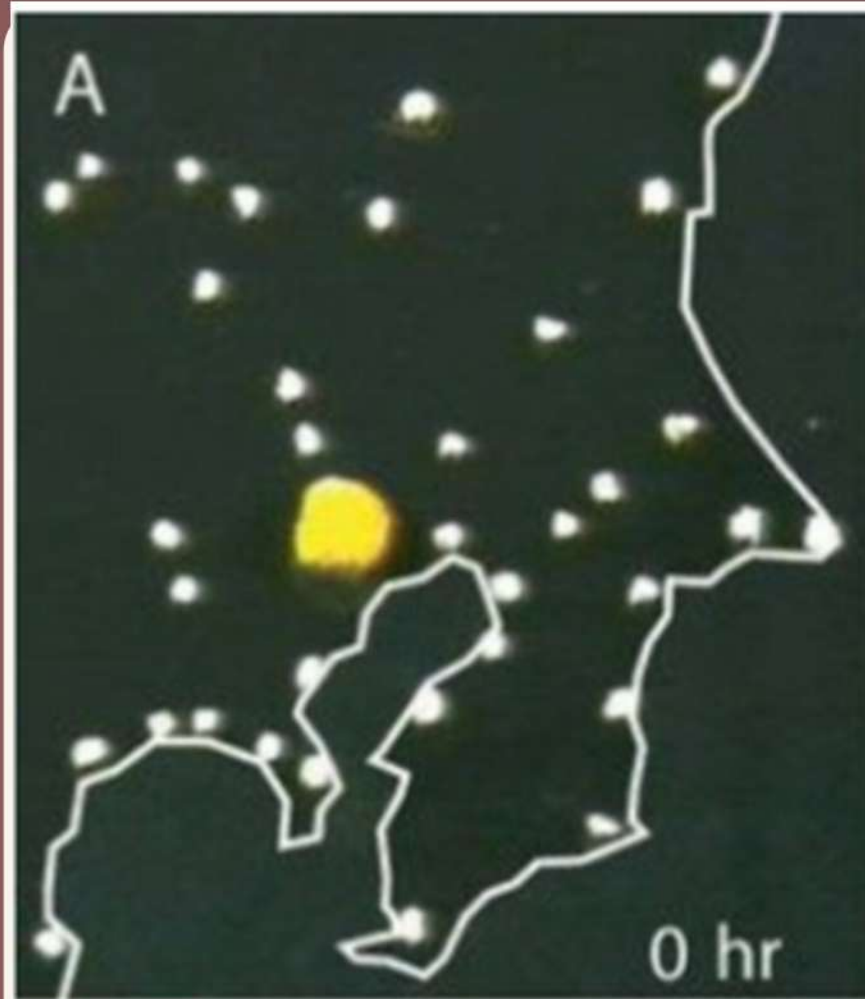
黏菌

黏菌屬於原生生物界，具有細胞的特性，並且在生活史中展現出多樣的形態，包括單細胞和多細胞的階段。通常生活於潮濕環境，在生態系統中扮演著重要的分解者角色。

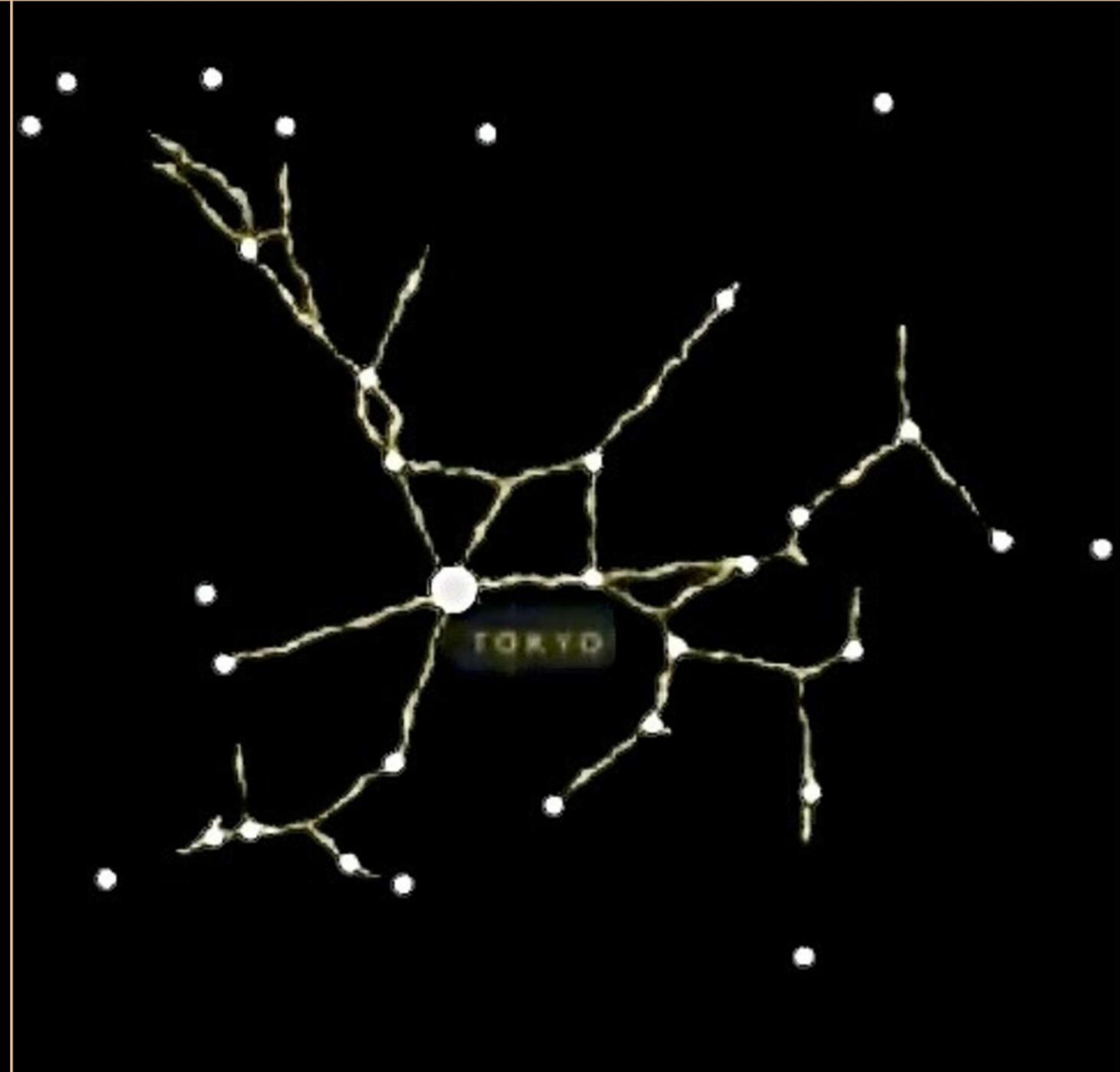
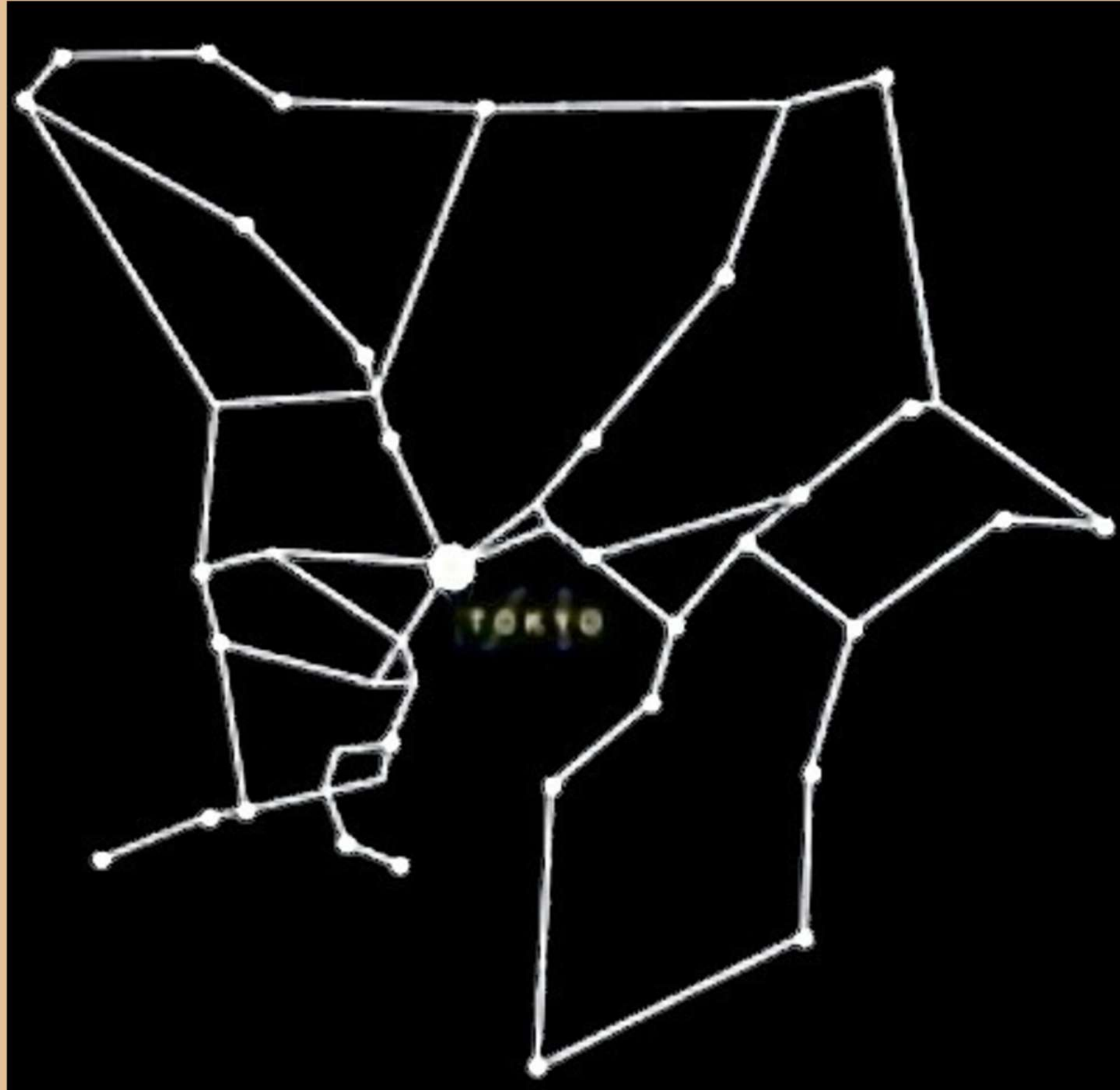


為了尋找食物，黏菌會先向四面八方擴展，定位所有潛在的食物來源後，再收縮自身，建立一個高效率的傳輸網路，將各處的食物連結起來。這樣的網路讓黏菌能有效地吸收營養，但結構不能過於複雜，否則會消耗過多能量。

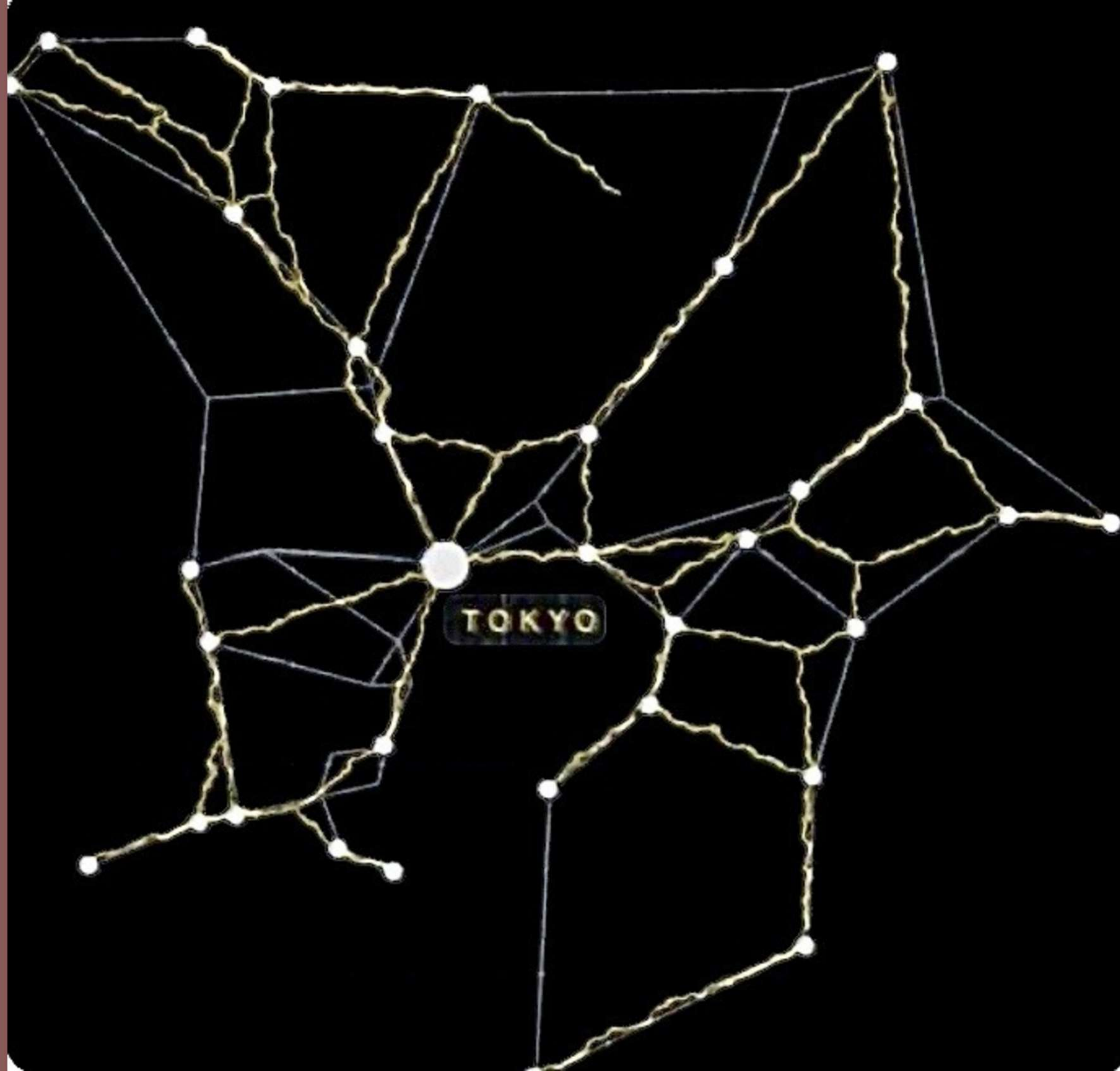
有趣的是，黏菌在建構路徑時也會考量風險。如果通往某個食物的位置只有單一路線，它們可能會額外建立備用路徑，以防主路徑中斷。這種策略在效率與安全之間取得了良好的平衡。



科學家利用黏菌具有避光的特性，以光點模擬日本的地形，並在東京幾個重要地鐵站的對應位置放置食物來源。結果發現，黏菌為了有效吸收營養，鋪設出一條高效率的網路，而這條由自然生成的路徑竟然與東京複雜的地鐵系統幾乎完全相符。換句話說，黏菌在沒有地圖、演算法或計算能力的情況下，只花了 26 小時，便重建了人類工程師花數十年設計出的交通網路，展現了自然演化出的驚人效率與智慧。



左邊是東京地鐵圖，右邊是黏菌所繪製出的



科學家表示：黏菌所設計出來的路線與現今的路線非常相似，有些路線設計甚至比原先來的更好。

黏菌走出路線分三個階段

接近食物

包圍食物

獲得食物



接近食物

$$X^{t+1} = \begin{cases} X_b^t + vb * (W * X_A^t - X_B^t) & r < p \\ vc * X^t & r \geq p \end{cases}$$

$$W(\text{Sindex}(i)) = \begin{cases} 1 + r_2 \times \lg\left(\frac{b_F - S_i}{b_F - w_F} + 1\right), & F_{\text{condition}} \\ 1 - r_2 \times \lg\left(\frac{b_F - S_i}{b_F - w_F} + 1\right), & \text{其他} \end{cases}$$

$$\text{Sindex}(i) = \text{sort}(N)$$

參數 r 為 $[0, 1]$ 區間的隨機數， X_b 表示目前適應度最佳的個體位置， X_a 與 X_b 為隨機選取的兩個個體位置。振盪參數 v_b 與 v_c 分別模擬黏菌個體間的資訊交互與自我調整行為，其中 $v_b \in [-a, a]$, $a = \text{arctanh}(1 - t/T_{\text{max}})$ ；而 $c = 1 - t/T_{\text{max}}$ 則控制 v_c 由 1 線性遞減至 0。

權重因子 W 代表黏菌個體的品質，與其適應度相關， t 代表當前反覆運算次數，位置更新策略受參數 p 控制，其定義為： $p = \tanh(|S_i - DF|)$
其中 S_i 為個體的適應度值， DF 為反覆運算中的最優適應度值。此設計可根據個體與最優解的差距自適應地調整更新幅度。

族群前半數適應度較佳的個體（記為 $F_{\text{condition}}$ ）會依適應度排序（ S_{index} ），以加強對優質解的聚焦。隨機數 $r_2 \in [0, 1]$ 模擬黏菌靜脈收縮過程中的不確定性，最優與最差適應度分別記為 bF 與 WF ，其變化經由 $\log$ 函數平滑處理，避免劇烈震盪影響搜索穩定性。黏菌群體會根據食物濃度調整搜索策略：在高濃度區域中，個體權重 W 較大，更集中探索；而在低濃度區域則降低權重，促使演算法轉向其他區域尋找潛在最優解。

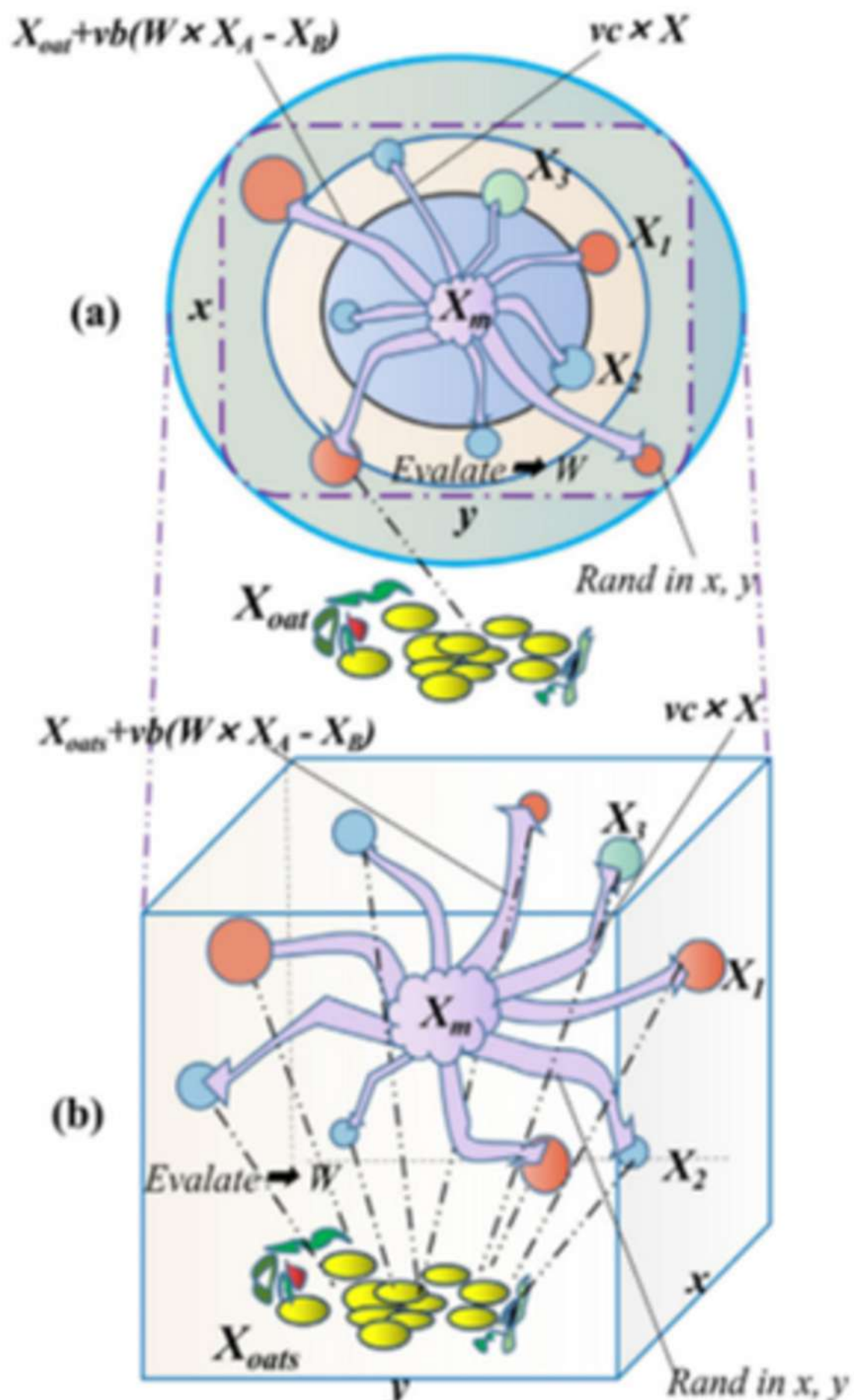
包圍食物

$$X^{t+1} = \begin{cases} rand * (UB - LB) + LB & rand < z \\ X_b^t + vb * (W * X_A^t - X_B^t) & r < p \\ vc * X^t & r \geq p \end{cases}$$

CSDN @KAU的云实验台

其中rand和r取區間 $[0, 1]$ 中的隨機值，UB和LB分別是搜索範圍的上邊界和下邊界。z是一個參數，為黏菌分離個體偏離當前路徑、探索其他食物源的機率，亦即產生變異的機率，通常設為 0.03。

獲得食物



Phase 1: Approach food

To model the approaching behavior of slime mould as a mathematical equation

$$\vec{X}(t+1) = \begin{cases} \vec{X}_b(t) + \vec{vb} \cdot (\vec{W} \cdot \vec{X}_A(t) - \vec{X}_B(t)), & r < p \\ \vec{vc} \cdot \vec{X}(t), & r \geq p \end{cases}$$

Where $p = \tanh |S(i) - DF|$

$$\vec{vb} = [-a, a] \quad a = \operatorname{arctanh} \left(-\left(\frac{t}{\max_t} \right) + 1 \right)$$

$$\vec{W}(\text{SmellIndex}(i)) = \begin{cases} 1 + r \cdot \log \left(\frac{bF - S(i)}{bF - wF} + 1 \right), & \text{condition} \\ 1 - r \cdot \log \left(\frac{bF - S(i)}{bF - wF} + 1 \right), & \text{others} \end{cases}$$

$$\text{SmellIndex} = \text{sort}(S)$$

Phase 2: Wrap food

The mathematical formula for updating the location of slime mould is as follows:

$$\vec{X}^i = \begin{cases} \text{rand} \cdot (UB - LB) + LB, & \text{rand} < z \\ \vec{X}_b(t) + \vec{vb} \cdot (\vec{W} \cdot \vec{X}_A(t) - \vec{X}_B(t)), & r < p \\ \vec{vc} \cdot \vec{X}(t), & r \geq p \end{cases}$$

where LB and UB denote the lower and upper boundaries of the search range

Phase 3: Grabble food

The value of $\vec{vb}$ oscillates randomly between $[a, a]$ and gradually approaches zero

as the iterations increase. The value of $\vec{vc}$ oscillates between $[-1, 1]$ and tends to zero eventually.

圖中顯示，黏菌個體在無方向性約束下進行覓食，能夠向任意方向接近最優解。其核心更新機制如下圖所示：第一個式子引入隨機性，增強演算法的多樣性；後兩個式子則根據震盪幅度的變化，分別實現全域與局部搜索能力。

獲得食物

$$X^{t+1} = \begin{cases} rand * (UB - LB) + LB & rand < z \\ X_b^t + vb * (W * X_A^t - X_B^t) & r < p \\ vc * X^t & r \geq p \end{cases}$$

CSDN @KAU的云实验台

食物源的吸引會引發黏菌自身的振蕩，進而改變其靜脈網路中細胞質的流動，使其逐步向食物源靠近。v_b和v_c即是類比這種振蕩的參數，v_b的值在 $[-a, a]$ 之間隨機振蕩，v_c的值在 $[-1, 1]$ 之間振蕩，並隨著反覆運算次數的增加逐漸趨於零。

導航怎麼提供最佳化路徑?

地圖資料 + 即時交通



圖形建模 (節點 + 邊 + 權重)



選擇演算法 (如 A*)



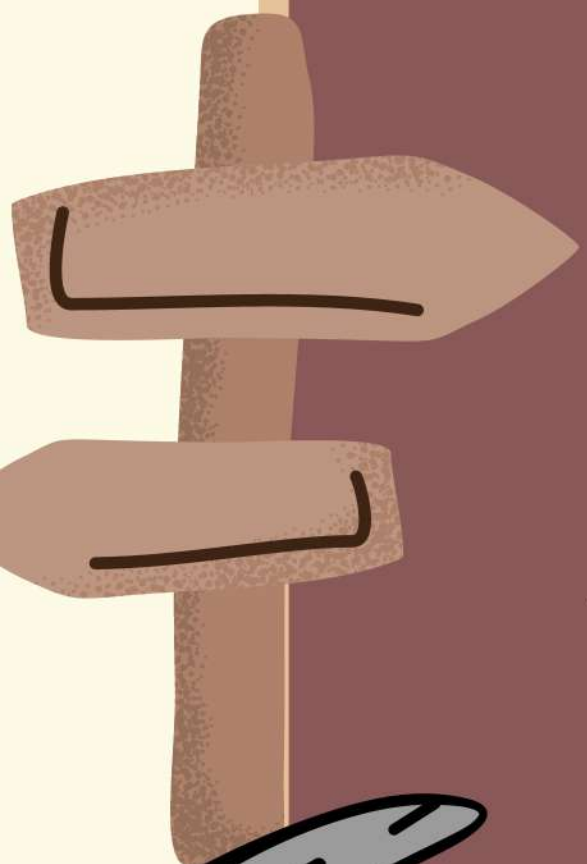
計算最佳路徑 (以時間或距離為目標)



持續更新 (動態調整)

主要是利用路徑規劃
演算法，結合交通資
料與地圖資訊。

現代應用的路徑最佳化



```
var origin1 = new google.maps.LatLng(55.930385, -3.118425);  
var origin2 = 'Greenwich, England';  
var destinationA = 'Stockholm, Sweden';  
var destinationB = new google.maps.LatLng(50.087692, 14.421150);
```

```
var service = new google.maps.DistanceMatrixService();
```

```
service.getDistanceMatrix(  
  {
```

方法會向距離矩陣服務發出要求，並將含有起點、目的地和交通方式的 DistanceMatrixRequest 物件常值，以及收到回應後要執行的回呼方法傳遞至該服務。

```
    origins: [origin1, origin2],  
    destinations: [destinationA, destinationB],  
    travelMode: 'DRIVING',  
    transitOptions: TransitOptions,  
    drivingOptions: DrivingOptions,  
    unitSystem: UnitSystem,  
    avoidHighways: Boolean,  
    avoidTolls: Boolean,  
  }, callback);
```

```
function callback(response, status) {  
  // See Parsing the Results for  
  // the basics of a callback function.  
}
```

- origins：計算距離和時間時要做為起點的陣列，內含一或多個地址字串、google.maps.LatLng 物件或 Place 物件。
- destinations：計算距離和時間時要做為目的地的陣列，內含一或多個地址字串、google.maps.LatLng 物件或 Place 物件。
- travelMode：計算路線時要使用的交通方式。
- unitSystem：顯示距離時要使用的單位系統。
- avoidHighways：如為 true，系統計算起點與目的地之間的路線時，會盡量避開高速公路。
- avoidTolls：如為 true，系統計算點與點之間的路線時，會盡量避開收費路段。

```
{
  departureTime: Date,
  trafficModel: TrafficModel
}
```

- `departureTime` 會將所需的出發時間指定為 `Date` 物件。如果在要求中加入 `departureTime`，API 就會根據當時的預期路況傳回最佳路線，並在回應中附上預估交通時間 (`duration_in_traffic`)。

```
{
  origins: [{lat: 55.93, lng: -3.118}, 'Greenwich, England'],
  destinations: ['Stockholm, Sweden', {lat: 59.087, lng: 14.421}],
  travelMode: 'DRIVING',
  drivingOptions: {
    departureTime: new Date(Date.now() + N), // for the time N milliseconds from now.
    trafficModel: 'optimistic'
  }
}
```

- `trafficModel`: 指定計算交通時間時要採用的假設。這項設定會影響回應中 `duration_in_traffic` 欄位傳回的值，其中包含根據歷來平均值預估的交通時間。
- `optimistic`: 表示傳回的 `duration_in_traffic` 在大多數日子應該都會比實際交通時間短，但偶爾路況特別好時，實際交通時間可能會短於這個值。

```

{
  "originAddresses": [ "Greenwich, Greater London, UK", "13 Great Carleton Square, Edinburgh, City of Edinburgh EH16 4, UK" ],
  "destinationAddresses": [ "Stockholm County, Sweden", "Dlouhá 609/2, 110 00 Praha-Staré Město, Česká republika" ],
  "rows": [ {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 70778,
        "text": "19 hours 40 mins"
      },
      "distance": {
        "value": 1887508,
        "text": "1173 mi"
      }
    }, {
      "status": "OK",
      "duration": {
        "value": 44476,
        "text": "12 hours 21 mins"
      },
      "distance": {
        "value": 1262780,
        "text": "785 mi"
      }
    }
  ], {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 26020

```

- originAddresses 陣列包含距離矩陣要求 origins 欄位中所傳遞的地點。地址會由地理編碼器進行格式化，然後傳回。
- destinationAddresses 陣列包含 destinations 欄位中所傳遞的地點，並採用地理編碼器傳回的格式。
- rows 是 DistanceMatrixResponseRow 物件的陣列，其中每一列都對應一個起點。
- elements 是 rows 的子項，會對應該列中起點與每個目的地的組合，且包含每個起點/目的地組合的狀態、時間長度、距離和車資資訊 (如有)。

```

    "distance": {
      "value": 1262780,
      "text": "785 mi"
    }
  } ]
}, {
  "elements": [ {
    "status": "OK",
    "duration": {
      "value": 96000,
      "text": "1 day 3 hours"
    },
    "distance": {
      "value": 2566737,
      "text": "1595 mi"
    }
  }, {
    "status": "OK",
    "duration": {
      "value": 69698,
      "text": "19 hours 22 mins"
    },
    "distance": {
      "value": 1942009,
      "text": "1207 mi"
    }
  } ]
} ]
}

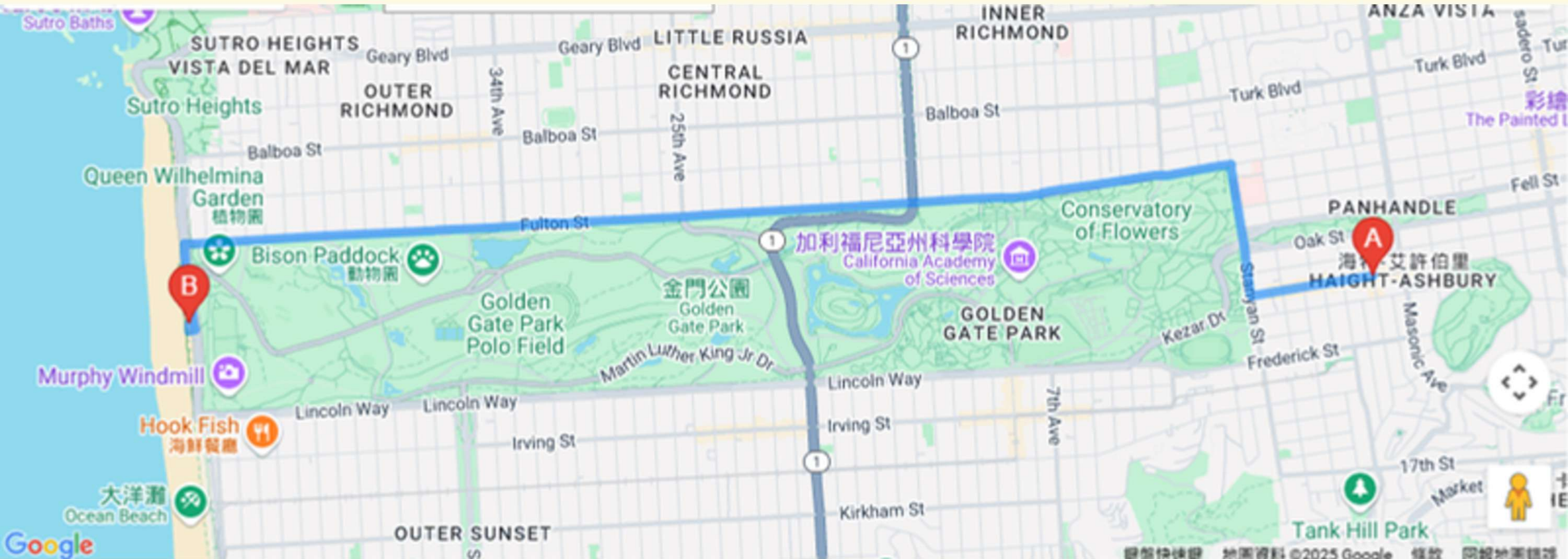
```

- **status**：距離矩陣回應包含整個回應的狀態碼，以及各項元素的狀態
- **duration**：行經這條路線所需的時間，以秒為單位 (value 欄位)，格式為 text。文字值的格式取決於要求中指定的 **unitSystem** (如未提供偏好設定，則採用公制)。
- **duration_in_traffic**：將目前路況列入考量，行經這條路線所需的時間，以秒 (value 欄位) 為單位，格式為 text。文字值的格式取決於要求中指定的 **unitSystem** (如未提供偏好設定，則採用公制)。只有在有車流量資料可用、mode 已設為 driving，且 **departureTime** 包含在要求的 **distanceMatrixOptions** 欄位中的情況下，才會傳回 **duration_in_traffic**。
- **distance**：這條路線的總距離，以公尺 (value) 為單位，格式為 text。文字值的格式取決於要求中指定的 **unitSystem** (如未提供偏好設定，則採用公制)。

Great Hwy, San Francisco, CA



Ashbury St, San Francisco, CA



```
function initMap(): void {
    const directionsRenderer = new google.maps.DirectionsRenderer();
    const directionsService = new google.maps.DirectionsService();
    const map = new google.maps.Map(
        document.getElementById("map") as HTMLElement,
        {
            zoom: 14,
            center: { lat: 37.77, lng: -122.447 },
        }
    );

    directionsRenderer.setMap(map);

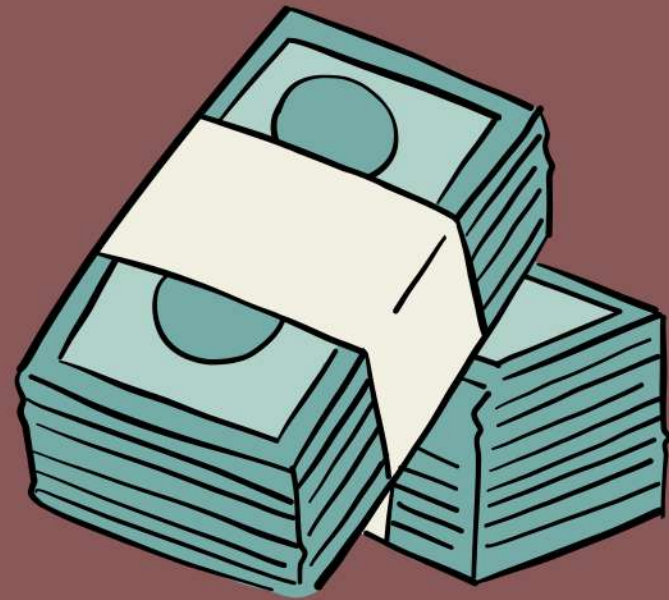
    calculateAndDisplayRoute(directionsService, directionsRenderer);
    (document.getElementById("mode") as HTMLInputElement).addEventListener(
        "change",
        () => {
            calculateAndDisplayRoute(directionsService, directionsRenderer);
        }
    );
}

function calculateAndDisplayRoute(
    directionsService: google.maps.DirectionsService,
    directionsRenderer: google.maps.DirectionsRenderer
) {
    const selectedMode = (document.getElementById("mode") as HTMLInputElement)
        .value;
```

```
directionsRenderer: google.maps.DirectionsRenderer
) {
  const selectedMode = (document.getElementById("mode") as HTMLInputElement)
    .value;

  directionsService
    .route({
      origin: { lat: 37.77, lng: -122.447 }, // Haight.
      destination: { lat: 37.768, lng: -122.511 }, // Ocean Beach.
      // Note that Javascript allows us to access the constant
      // using square brackets and a string value as its
      // "property."
      travelMode: google.maps.TravelMode[selectedMode],
    })
    .then((response) => {
      directionsRenderer.setDirections(response);
    })
    .catch((e) => window.alert("Directions request failed due to " + status));
}

declare global {
  interface Window {
    initMap: () => void;
  }
}
window.initMap = initMap;
```



經濟面向的最佳化實務



怎麼生產才最賺

玩具工廠正考慮下個月要產出多少塑膠貝殼和塑膠蝦？每個塑膠貝殼與塑膠蝦的利潤分別是3元及4元，耗費塑膠粒數分別為125粒和175粒，到月底前能夠生產五百個塑膠貝殼和七百隻塑膠蝦，但庫存的塑膠粒僅剩十一萬粒，請問以最佳利潤為前提下，分別要生產出多少個？

有獎徵答

B10		fx =B3*B6+C3*C6		
	A	B	C	D
1	決策變量			
2		塑膠貝殼	塑膠蝦	
3	生產數量(個)	499	272	
4				
5	目標函式			
6	單位利潤(元)	3	4	
7				
8				
9	目標變量	2585 ×		
10	總利潤(元)	=B3*B6+C3*C6		
11				
12	限制條件			
13	時間內限制(個)	500	700	
14	單位塑膠粒(個)	125	175	
15	總共使用塑膠粒(	109975		
16	塑膠粒庫存上限(	110000		
17				

B15		fx =B14*B3+C14*C3		
	A	B	C	D
1	決策變量			
2		塑膠貝殼	塑膠蝦	
3	生產數量(個)	499	272	
4				
5	目標函式			
6	單位利潤(元)	3	4	
7				
8				
9	目標變量			
10	總利潤(元)	2585		
11				
12	限制條件			
13	時間內限制(個)	500	700	
14	單位塑膠粒(個)	125	175	
15	總共使用塑膠粒(	109975 ×		
16	塑膠粒庫存上限(	110000		
17				

Optimization Solver - Free Trial

Main Menu

More

FREE TRIAL

4 days, 21 hours, 34 minutes remaining

Objective Function Cell: ?

Cell Reference

B10

Min / Max?

Maximize

Decision Variables: ?

Check the boxes for integers ?

Cell Reference or Range

☒ B3

☒ C3

Add Variables

Constraints: ?

☒ Enforce Decision Variable Non-Negativity

LHS	Operator	RHS
C3	<=	C13
B3	<=	B13
B15	<=	B16

Add Constraints

Solution Method

Solve

Linear

怎麼生產才最賺

一家製造公司生產三種產品：A、B、C，並有兩條產線：產線1與產線2。公司目標是在滿足市場需求與資源限制的條件下，最大化每月總利潤。利潤分別是40、35、60(美元)，已知資訊如下：

每種產品在不同產線的生產所需時間 (小時)

每單位產品所需要消耗原料(公斤)

產品	產線1 (小時)	產線2(小時)	原料 消耗(公斤)
A	2	1	4
B	3	2	3
C	4	3	5

每個月最多運作時數

產線1：1,600 小時

產線2：1,200 小時

原料上限:4300公斤

\$市場需求限制\$

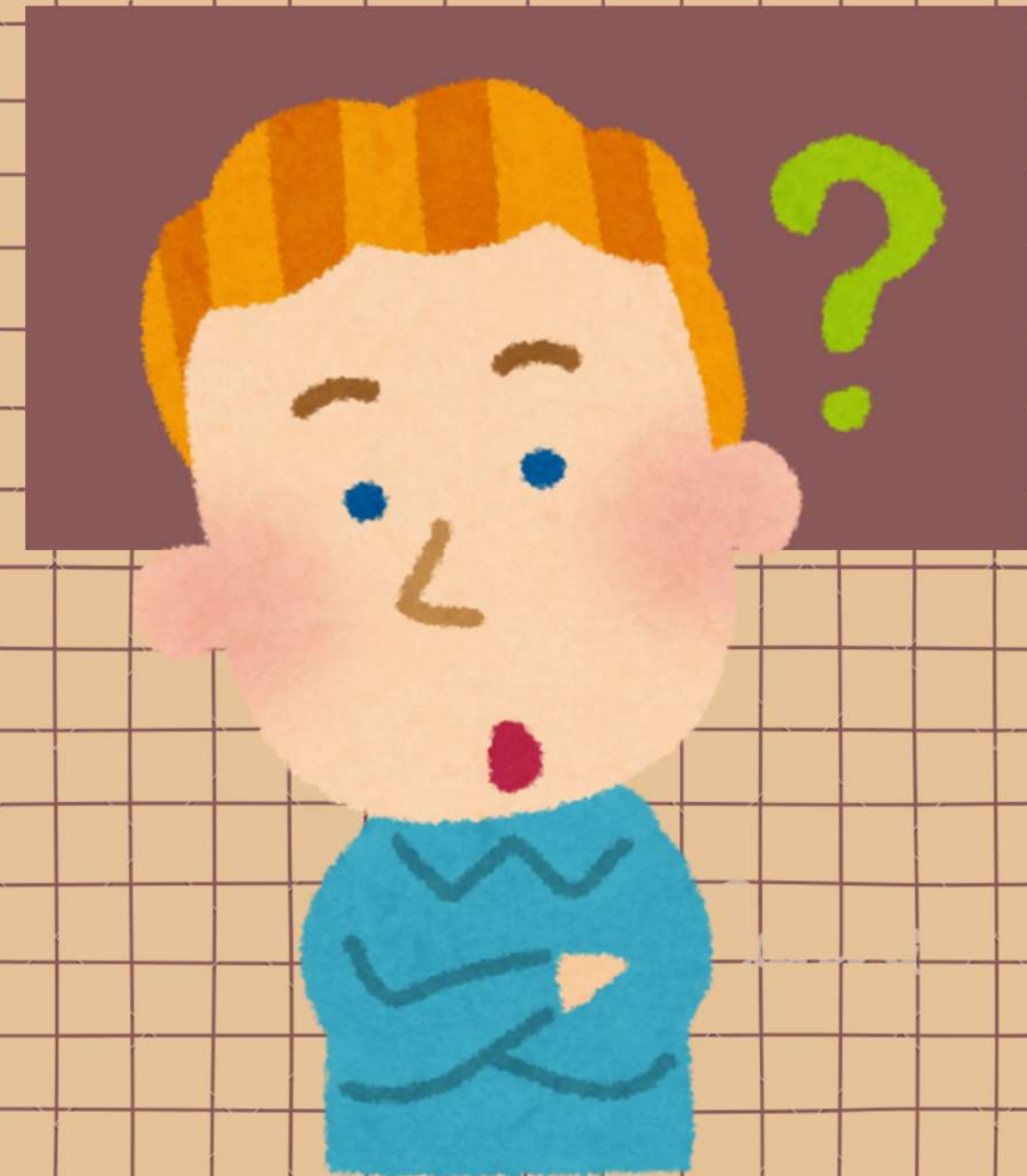
- 1.至少生產200單位的產品A
- 2.最多只能銷售500單位產品C

\$產線要求\$

產線1的A物和B物至少產出50單位物出來

產線2的A物和B物要至多產出225單位物出來

最佳利潤
為多少？



F5								
	A	B	C	D	E	F	G	H
1	決策變量	A物	B物	C物				
2								
3	產線1					產線1總時數		
4	生產量	2	288	183				
5	單位產量所需時數	2	3	4		$=B5*B4+C5*C4+D5*D4$		
6	A+B	290						
7	產線2					產線2總時數		
8	生產量	198	27	316				
9	單位產量所需時數	1	2	3		$=B9*B8+C9*C8+D9*D8$		
10	A+B	225						
11								
12	單位耗材(公斤)	4	3	5				
13								
14	總生產數量	200	315	499				
15	總耗量(公斤)	$=B12*B14+C12*C14+D12*D14$						
16								
17	目標函式							
18	單位利潤(美元)	40	35	60				
19								
20	目標變量							
21	總利潤(美元)	$=B14*B18+C14*C18+D14*D18$						
22								
23	限制條件							
24		產線1	產線2					
25	運作時間上限(小時)	1600	1200					
26								
27	原料上限(公斤)	4300						
28	市場需求	200		500				
29	產線1的A+B	50						
30	產線2的A+B	225						

分鐘前 (編輯者: 彬)

Excel Solver - Free Trial

Cell Reference

B21

Min / Max?

Maximize

Decision Variables: ?

Check the boxes for integers ?

Cell Reference or Range

☒ B4

☒ C4

☒ D4

☒ B8

☒ C8

☒ D8

Add Variables

Constraints: ?

☒ Enforce Decision Variable Non-Negativity

LHS	Operator	RHS	
F5	$\leq$	B25	
F9	$\leq$	C25	
B14	$\leq$	B28	
B15	$\leq$	B27	
D14	$\leq$	D28	
B6	$\geq$	B29	
B10	$\leq$	B30	

啟用 Windows
移至 [設定] 以啟用 Windows。

參考資料

1.黏菌演算法原理、實作及其改良與利用

<https://blog.csdn.net/sfejojno/article/details/135043239>

2.揭秘大規模多目標黏菌算法：高效優化與未來應用展望

<https://www.oryoy.com/news/jie-mi-da-gui-mo-duo-mu-biao-nian-jun-suan-fa-gao-xiao-you-hua-yu-wei-lai-ying-yong-zhan-wang.html>

3.這種「單細胞生物」只花幾小時就超越「東京地鐵」數十年努力！實驗結果「沒腦」
是比人聰明關鍵

<https://www.teepr.com/805484/adrianchiang/黏菌東京地鐵/>

4.最佳化問題：規劃求解

<https://blog.pulipuli.info/2017/09/optimization-problem-solver.html>

參考資料

5.距離矩陣服務

https://developers.google.com/maps/documentation/javascript/legacy/distancematrix?hl=zh-tw#distance_matrix_requests

6.路線中的出行方式

https://developers.google.com/maps/documentation/javascript/examples/directions-travel-modes#maps_directions_travel_modes-typescript

7.Number Engine:外掛程式

<https://www.numberengine.app/home>

本報告從自然界、生產管理與現代科技三個層面探討了最佳化問題的應用與意義。透過黏菌模擬東京地鐵路網的實驗，我們看到生物體在無中心控制下仍能自發形成高效路徑，展現了自然演化出的最佳化能力。導航系統則是數學與演算法在人類日常生活中的實踐，為使用者提供成本最低或時間最短的路徑。而在生產管理中，工廠如何在原物料受限的條件下最大化利潤，則體現了最佳化在資源配置上的關鍵價值。綜上所述，最佳化不僅是理論上的數學問題，更深刻地影響著自然、生產與生活。未來隨著人工智慧與計算技術的進步，最佳化方法將持續拓展其應用範疇，為人類解決更複雜的決策問題提供支持。



THANK YOU

GROUP 8