

# Algoritmos MST y SPT aplicados en artículos recientes

Delgado Romero, Gustavo<sup>1</sup>    Torres Reategui, Joaquin<sup>2</sup>

Facultad de Ciencias  
Escuela profesional de Ciencias de la computación  
Universidad Nacional de Ingeniería

24 de noviembre de 2025

Documentos fuente:

- <https://doi.org/10.3390/buildings14010213>
- <https://arxiv.org/pdf/2511.11598v1>



Facultad de  
**CIENCIAS**

1 Introducción

2 Algoritmo MST

3 Algoritmo SPT

4 Conclusiones

# Introducción

A continuación, se presentan dos algoritmos utilizados en la optimización de redes y estructuras: el Algoritmo de Árbol de Expansión Mínima (MST) y el Algoritmo de Árbol de caminos más cortos (SPT). Estos algoritmos son fundamentales en diversas aplicaciones, desde la planificación de infraestructuras hasta la optimización de rutas en redes de comunicación.

## 1 Introducción

## 2 Algoritmo MST

- Título y planteamiento del problema
- Formulación matemática (MILP)
- Algoritmo propuesto (detallado)
- Ejemplo de Aplicación del Algoritmo
- Análisis de complejidad — paso a paso

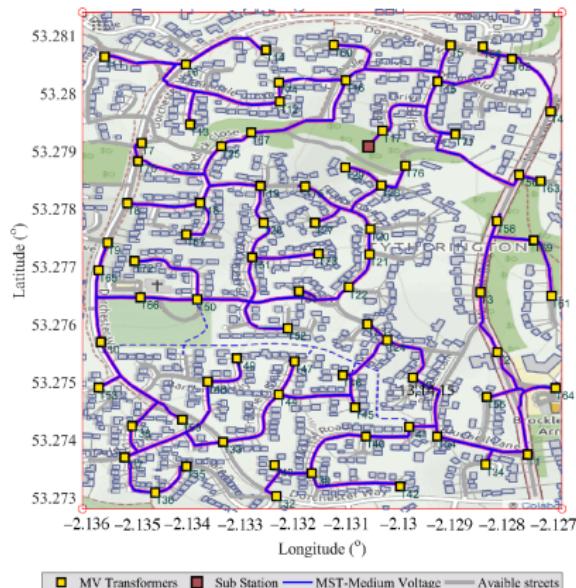
## 3 Algoritmo SPT

- Título y planteamiento del problema
- Formulación matemática
- Algoritmo propuesto (detallado)
- Ejemplo de aplicación del algoritmo
- Análisis de complejidad, paso a paso

## 4 Conclusiones

- **Título:** Integrating Minimum Spanning Tree and MILP in Urban Planning: A Novel Algorithmic Perspective.
- **Objetivo:** diseñar una asignación de viviendas a puntos (esquinas/POI) y una red que conecte esas esquinas minimizando un coste combinado.
- **Decisiones:**
  - para cada vivienda, elegir exactamente una esquina (asignación),
  - elegir las aristas del MST entre esquinas (diseño de red).
- **Restricciones relevantes:**
  - capacidad máxima de viviendas por esquina,
  - variables binarias (asignación y selección de aristas),
  - estructura de árbol para la red (MST).

# Ejemplo de aplicación del problema



**Figura:** Demostración de la aplicación del algoritmo en un paisaje urbano del Reino Unido, mostrando las conexiones optimizadas entre casas y transformadores a través del MST.

# Notación y parámetros

$N$  número de viviendas, indexadas por  $i = 1, \dots, N$ .

$M$  número de esquinas (candidatas o seleccionadas), indexadas por  $j = 1, \dots, M$ .

$d_{ij}$  distancia (Euclíadiana) entre vivienda  $i$  y esquina  $j$ .

$w_{uv}$  peso (distancia) entre esquina  $u$  y esquina  $v$  (posibles aristas).

$x_{ij} \in \{0, 1\}$  variable: 1 si vivienda  $i$  asignada a esquina  $j$ .

$y_{uv} \in \{0, 1\}$  variable: 1 si arista  $(u, v)$  está en la solución (MST).

$C$  capacidad máxima de viviendas por esquina.

# Función objetivo

$$\min Z = \underbrace{\sum_{i=1}^N \sum_{j=1}^M d_{ij} x_{ij}}_{\text{coste asignación}} + \underbrace{\sum_{(u,v) \in E} w_{uv} y_{uv}}_{\text{longitud de la red (MST)}}$$

donde  $E$  es el conjunto de aristas posibles entre esquinas.

# Restricciones (esenciales) I

(Asignación única)

$$\sum_{j=1}^M x_{ij} = 1 \quad \forall i = 1, \dots, N \quad (1)$$

(Capacidad por esquina)

$$\sum_{i=1}^N x_{ij} \leq C \quad \forall j = 1, \dots, M \quad (2)$$

(Binarias)

$$x_{ij} \in \{0, 1\}, y_{uv} \in \{0, 1\} \quad \forall i, j, (u, v) \in E \quad (3)$$

(MST:  $M$  nodos)

$$\sum_{(u,v) \in E} y_{uv} = M - 1 \quad (4)$$

*Nota:* la restricción que garantiza conectividad sin ciclos (estructura árbol) puede modelarse mediante cortes o mediante un flujo (formulación basada en órdenes o variables de subtour elimination).

# Visión general del algoritmo

- Mezcla heurística (Greedy), solución exacta parcial (Prim para MST) y refinamiento local mediante MILP.
- Es iterativo: se prueba con distintas capacidades u otros parámetros y se guarda la mejor solución.
- En la práctica se reduce el MILP a sub-problemas manejables (pocas variables binarias en conflicto).

---

## Algorithm 1: Algoritmo para optimización de asignación vivienda-esquina y diseño de red MST (Parte 1)

---

**Data:** Coordenadas de viviendas y esquinas

**Result:** Asignación óptima y árbol MST

**1 Paso 1: Definir el Escenario de Viviendas**

2 Obtener coordenadas de las viviendas.

3 Sea  $N$  el número total de viviendas.

4 Cada vivienda  $i$  con  $i \in \{1, 2, \dots, N\}$ .

**5 Paso 2: Identificar Esquinas**

6 Definir las esquinas como intersecciones de la red vial.

7 Sea  $M$  el número total de esquinas.

8 Cada esquina  $j$  con  $j \in \{1, 2, \dots, M\}$ .

**9 Paso 3: Matriz de Distancias**

10 Calcular distancias  $d_{ij}$  entre cada vivienda y esquina.

11 Inicializar la función objetivo con un valor muy grande.

12 Continúa en la siguiente diapositiva...

---

# Algoritmo 1: Optimización Vivienda–Esquina y MST (II)

---

## Algorithm 1: Algoritmo para optimización de asignación vivienda-esquina y diseño de red MST (Parte 2)

---

13 **while** la función objetivo no esté minimizada **do**

14     **Paso 4: Restricciones de Capacidad**

15         Definir capacidad máxima  $C$  por esquina.

16         Restricción:  $\sum_{i=1}^N x_{ij} \leq C \quad \forall j.$

17     **Paso 5: Asignación Greedy**

18         Asignar cada vivienda a la esquina más cercana respetando  $C$ .

19         Restricción:  $\sum_{j=1}^M x_{ij} = 1 \quad \forall i.$

20          $x_{ij} \in \{0, 1\}.$

21     **Paso 6: Cálculo del MST**

22         Calcular el árbol de expansión mínima (MST) entre las esquinas activas.

23          $y_{uv} \in \{0, 1\}$  para cada arista  $(u, v)$  del MST.

---

24 Continúa en la siguiente diapositiva...

---

# Algoritmo 1: Optimización Vivienda–Esquina y MST (III)

---

**Algorithm 1:** Algoritmo para optimización de asignación vivienda-esquina y diseño de red MST (Parte 3)

---

25   **while** la función objetivo no esté minimizada (*cont*) **do**

26     **Paso 7: Función Objetivo**

27       Minimizar:

$$28 \quad \sum_{i=1}^N \sum_{j=1}^M d_{ij} x_{ij} + \sum_{(u,v) \in E} w_{uv} y_{uv}.$$

29       Actualizar valor de la función objetivo.

30     **Paso 8: Iteración**

31       Ajustar la capacidad  $C$  si fuera necesario.

32       Repetir el proceso hasta convergencia.

---

# Ejemplo Simple: Escenario Urbano

## Viviendas (coordenadas):

- $H_1 = (1,1)$
- $H_2 = (2,1)$
- $H_3 = (4,2)$
- $H_4 = (6,1)$

## Esquinas:

- $C_1 = (1,2)$
- $C_2 = (4,1)$
- $C_3 = (7,2)$

Capacidad por esquina:  $C = 2$

## Distancias vivienda–esquina:

	C1	C2	C3
H1	1.0	3.0	6.1
H2	1.4	2.0	5.1
H3	3.6	1.0	3.2
H4	5.4	2.0	1.4

## Asignación Greedy:

- $H_1 \rightarrow C_1$
- $H_2 \rightarrow C_1$  ( $C_1$  lleno)
- $H_3 \rightarrow C_2$
- $H_4 \rightarrow C_3$

Capacidades satisfechas.

# Ejemplo Simple: MST y Costo Final

## Paso 6: Cálculo del MST entre esquinas activas

Distancias entre esquinas:

- $C1-C2 = 3.2$
- $C2-C3 = 3.2$
- $C1-C3 = 6.1$

MST elegido:

$$\{(C1, C2), (C2, C3)\}$$

Longitud total del MST:

$$3,2 + 3,2 = 6,4$$

Costo de asignación:

$$1,0 + 1,4 + 1,0 + 1,4 = 4,8$$

Costo total:

$$4,8 + 6,4 = \boxed{11,2}$$

Resultado final:

- $C1 \leftarrow H1, H2$
- $C2 \leftarrow H3$
- $C3 \leftarrow H4$
- MST conecta  $C1-C2-C3$

# Comentarios del algoritmo

- La fase Greedy es rápida y proporciona una solución inicial factible (si  $C$  lo permite).
- El MILP se usa solo para arreglar conflictos puntuales (reduce coste computacional).
- El MST con Prim es exacto para la subparte de diseño de red.
- El carácter iterativo permite explorar el impacto de  $C$  y otras decisiones (robustez).

# Notación para complejidad

Definamos:

- $N$  = número de viviendas.
- $M$  = número de esquinas.
- $K$  = número de iteraciones (valores de  $C$  intentados) o cambios de configuración.
- $E$  = número de aristas entre esquinas (usualmente  $E = O(M^2)$  en grafo completo).

# Complejidades parciales

Cálculo de matriz de distancias: calcular  $d_{ij}$  para todo par  $(i,j)$ :

$$O(N \cdot M)$$

Asignación Greedy (por vivienda buscar POI más cercano con capacidad):

$$O(N \cdot M) \text{ (en implementación simple)}$$

Si se indexa adecuadamente o se ordenan vecinos puede reducirse a  $O(N \log M)$ .

Cálculo del MST (Prim con heap):

$$O(E \log M) \text{ o } O(M^2) \text{ si es denso}$$

Resolución del MILP (local):

NP-hard en general; tiempo exponencial en tamaño worst-case

# Complejidad total estimada

Combinando y multiplicando por  $K$  iteraciones:

$$T_{\text{total}}(N, M, K) = O\left(K \cdot (NM + \text{MST}(M, E) + \text{MILP}_{\text{local}})\right)$$

Simplificando:

$$T_{\text{total}}(N, M, K) = O\left(K(NM^2)\right)$$

## 1 Introducción

## 2 Algoritmo MST

- Título y planteamiento del problema
- Formulación matemática (MILP)
- Algoritmo propuesto (detallado)
- Ejemplo de Aplicación del Algoritmo
- Análisis de complejidad — paso a paso

## 3 Algoritmo SPT

- Título y planteamiento del problema
- Formulación matemática
- Algoritmo propuesto (detallado)
- Ejemplo de aplicación del algoritmo
- Análisis de complejidad, paso a paso

## 4 Conclusiones

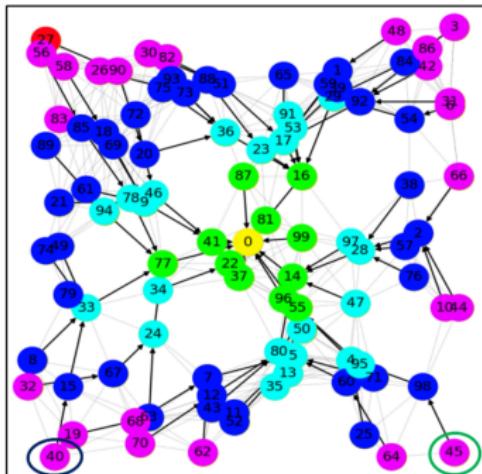
# Título y planteamiento del problema del artículo I

- **Título:** Distributed Q-learning-based Shortest-Path Tree Construction in IoT Sensor Networks.
- **Contexto:**
  - Redes de sensores IoT modeladas como un grafo no dirigido  $G = (V, E)$ .
  - Un nodo sumidero  $v_0$  actúa como raíz del árbol de enrutamiento.
- **Objetivo:** construir, de forma distribuida, un Árbol de Caminos Más Cortos (SPT) hacia  $v_0$ , donde cada nodo elige su siguiente salto usando únicamente información local.
- **Idea central:**
  - Cada nodo ejecuta Q-learning para aprender la “calidad” de enviar paquetes a cada vecino.
  - La política  $\pi(v) = \arg \max_{u \in N(v)} Q(v, u)$  define el padre de  $v$  en el SPT.

- **Restricciones y requisitos:**

- Operación totalmente distribuida (sin controlador central).
- Uso de información local (vecinos de un salto).
- Árbol libre de ciclos y estable para tráfico convergecast.
- Bajo consumo de energía y baja sobrecarga de señalización.

# Ilustración conceptual del SPT en IoT



(b) Dijkstra's Algorithm

**Figura:** Topología de una red de sensores IoT (nodos en azul) y sumidero  $v_0$  (en rojo), donde las aristas resaltadas muestran el Árbol de Caminos Más Cortos.

# Modelo de red y problema SPT

- Red de sensores:
  - Grafo no dirigido  $G = (V, E)$ ,  $|V| = N$ .
  - Enlace  $(u, v) \in E$  si la distancia euclídea es menor que un radio  $R$  y la calidad de enlace es suficiente.
  - Nodo sumidero fijo  $v_0 \in V$ .
- Costo de enlace:
  - $w(u, v) \geq 0$ ; en el artículo se utiliza principalmente **conteo de saltos**, es decir,  $w(u, v) = 1$ .
- **Árbol de Caminos Más Cortos (SPT)**: árbol  $T = (V, E_T)$  enraizado en  $v_0$  tal que

$$d_T(v, v_0) = d_G(v, v_0), \quad \forall v \in V,$$

donde  $d_T$  es la longitud del camino en el árbol y  $d_G$  la longitud del camino más corto en  $G$ .

- Condiciones adicionales:
  - Operación descentralizada y basada en vecinos de un salto.
  - Mensajes de control pequeños y poco frecuentes.
  - Libre de ciclos para evitar bucles de enrutamiento.

# Componentes del modelo de Q-learning

- **Estado:** nodo actual  $v \in V$  donde se toma la decisión de enrutamiento.
- **Acción:** elegir un vecino  $u \in N(v)$  como siguiente salto.
- **Recompensa inmediata:**

$$r(v, u) = \begin{cases} 100, & \text{si } u = v_0, \\ 0, & \text{si } u \in N(v) \setminus \{v_0\}. \end{cases}$$

- **Episodio:** comienza en un nodo  $v \neq v_0$  y termina cuando se alcanza  $v_0$  o no hay vecinos válidos.
- **Matriz Q:**
  - Cada nodo  $v$  mantiene valores  $Q(v, u)$  para todos sus vecinos  $u \in N(v)$ .
  - La política  $\pi(v) = \arg \max_{u \in N(v)} Q(v, u)$  define el padre de  $v$  en el árbol.
- **Parámetros:** tasa de aprendizaje  $\alpha$ , factor de descuento  $\gamma$  y probabilidad de exploración  $\epsilon$ .

# Actualización de Q-valores

- Política  $\epsilon$ -greedy en el nodo  $v$ :
  - Con probabilidad  $\epsilon$ : elegir  $u$  al azar en  $N(v)$  (exploración).
  - Con probabilidad  $1 - \epsilon$ : elegir  $u$  que maximiza  $Q(v, u)$  (explotación).
- Tras tomar acción  $u$ , se recibe recompensa  $r(v, u)$  y se pasa al estado  $u$ .
- **Ecuación de actualización (Bellman):**

$$Q(v, u) \leftarrow (1 - \alpha)Q(v, u) + \alpha \left( r(v, u) + \gamma \max_{u' \in N(u)} Q(u, u') \right).$$

- En convergencia, los Q-valores aproximan la estructura de caminos más cortos hacia  $v_0$  y la política inducida define el SPT.

- **Fase de entrenamiento (offline):**

- Se generan múltiples topologías aleatorias de red.
- Cada nodo ejecuta Q-learning de manera distribuida sobre esos grafos.
- Se obtiene una matriz  $Q$  entrenada que captura patrones de enrutamiento eficientes.

- **Fase de prueba (online):**

- Sobre una nueva red, cada nodo usa la matriz  $Q$  entrenada.
- Cada nodo selecciona a su parente combinando  $Q(v, u)$  con la distancia al sumidero.
- Se construye un SPT libre de ciclos usando solo información local.

- **Ventaja clave:**

- No se requiere recomputar Dijkstra ni mantener una visión global de la red.
- La señalización se limita a intercambios locales entre vecinos.

## Algoritmo 2: Entrenamiento Q-learning para SPT

---

### Algorithm 2: Entrenamiento distribuido de Q-valores para construcción de un SPT

---

**Data:** Conjunto de grafos de entrenamiento  $G_m = (V_m, E_m)$ ; sumidero  $v_0$ ; parámetros  $\alpha, \gamma, \epsilon$   
**Result:** Matriz  $Q$  entrenada

```
1 Inicializar  $Q(v, u)$  para todas las parejas  $(v, u)$  válidas (vecinos en radio  $R$ ).  
2 for  $m = 1$  to  $M$  do  
3   Cargar el grafo  $G_m = (V_m, E_m)$  con posiciones de nodos y enlaces.  
4   for  $k = 1$  to  $K$  do  
5     Elegir al azar un nodo inicial  $v \in V_m \setminus \{v_0\}$ .  
6     while  $v \neq v_0$  do  
7       // Selección  $\epsilon$ -greedy del siguiente salto  
8       Con probabilidad  $\epsilon$ , escoger  $u$  uniforme en  $N_{V_m}(v)$ .  
9       Con probabilidad  $1 - \epsilon$ , escoger  $u = \arg \max_{u' \in N_{V_m}(v)} Q(v, u')$ .  
10      Observar recompensa  $r = r(v, u)$ .  
11      Calcular  $Q_{\max} = \max_{u' \in N_{V_m}(u)} Q(u, u')$ .  
12      // Actualización de Bellman  
13       $Q(v, u) \leftarrow (1 - \alpha)Q(v, u) + \alpha(r + \gamma Q_{\max})$ .  
14       $v \leftarrow u$ .
```

---

# Algoritmo 3: Construcción del SPT en fase de prueba

---

## Algorithm 3: Construcción distribuida del SPT usando la matriz $Q$ entrenada

---

**Data:** Matriz  $Q$  entrenada; grafo de prueba  $G_t = (V, E)$ ; sumidero  $v_0$

**Result:** Árbol de caminos más cortos aproximado  $T_t = (V, E_{T_t})$

```
1 Inicializar  $E_{T_t} \leftarrow \emptyset$ .
2 for cada nodo  $v \in V \setminus \{v_0\}$  do
3     Inicializar camino  $P \leftarrow \{v\}$ , nodo actual  $c \leftarrow v$ .
4     while  $c \neq v_0$  y  $|P| \leq |V|$  do
5         Obtener vecinos no visitados  $N'(c) = \{u \in N(c) \mid u \notin P\}$ .
6         if  $N'(c) = \emptyset$  then
7             break (no hay ruta válida).
8         // Heurística: Q-valor y distancia al sumidero
9         Para cada  $u \in N'(c)$ , calcular  $S(u) = Q(c, u) - d(u, v_0)$ .
10        Escoger  $u = \arg \max_{u \in N'(c)} S(u)$ .
11        Añadir arista dirigida  $(c, u)$  a  $E_{T_t}$ .
12        Añadir  $u$  a  $P$  y actualizar  $c \leftarrow u$ .
```

---

# Ejecución del algoritmo sobre una topología de prueba

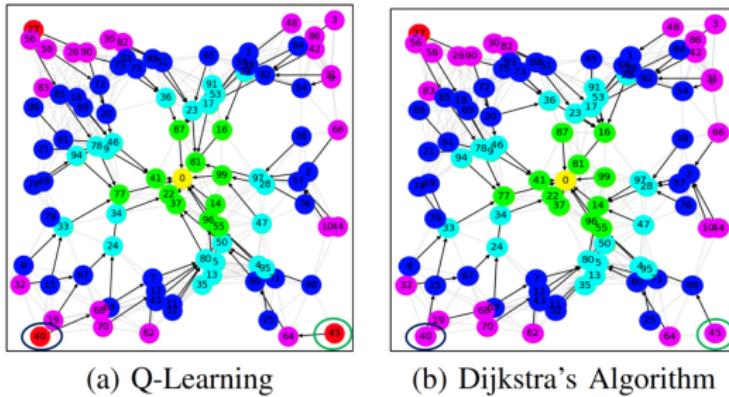


Figura: Comparación entre el SPT óptimo (por ejemplo, calculado con Dijkstra) y el SPT obtenido mediante Q-learning en una topología de prueba.

# Ejemplo Simple: Red IoT geométrica I

## Escenario:

- Red IoT con 6 nodos numerados de 0 a 5.
- Nodo sumidero (raíz del SPT): 0.
- Nodos distribuidos aleatoriamente en el cuadrado  $[0, 1] \times [0, 1]$ .
- Enlace  $(u, v)$  si la distancia euclídea  $\leq R = 0,5$ .

# Ejemplo Simple: Red IoT geométrica II

**Posiciones (ejemplo fijo):**

- $0 = (0,10, 0,50)$  (sumidero)
- $1 = (0,20, 0,80)$
- $2 = (0,40, 0,60)$
- $3 = (0,70, 0,70)$
- $4 = (0,60, 0,30)$
- $5 = (0,30, 0,20)$

**Vecindarios obtenidos con  $R = 0,5$ :**

Nodo	Vecinos $N(v)$
0	{1, 2, 5}
1	{0, 2}
2	{0, 1, 3, 4, 5}
3	{2, 4}
4	{2, 3, 5}
5	{0, 2, 4}

# Ejemplo Simple: SPT aprendido y comparación I

## Parámetros de entrenamiento:

- Número de episodios:  $\text{NUM\_EPISODIOS} = 4000$ .
- Máximo de pasos por episodio:  $\text{MAX\_PASOS} = 2N$ .
- Tasa de aprendizaje:  $\alpha = 0,5$ .
- Factor de descuento:  $\gamma = 0,9$ .
- Exploración  $\epsilon$ -greedy:  $\epsilon = 0,2$ .

# Ejemplo Simple: SPT aprendido y comparación II

Política aprendida (padre de cada nodo):

$$\text{parent} = \begin{cases} \text{parent}(0) = \text{None} & (\text{sumidero}) \\ \text{parent}(1) = 0, \\ \text{parent}(2) = 0, \\ \text{parent}(3) = 2, \\ \text{parent}(4) = 2, \\ \text{parent}(5) = 0. \end{cases}$$

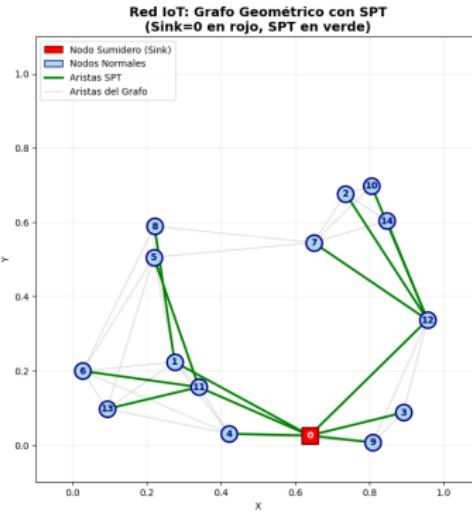
Esto define un SPT aproximado con las aristas:

$$E_T = \{(1, 0), (2, 0), (5, 0), (3, 2), (4, 2)\}.$$

Comparación distancias reales vs. política Q-learning:

Nodo	$d_{\text{real}}(v, 0)$	$d_Q(v, 0)$	Camino por la política
1	1	1	1 → 0
2	1	1	2 → 0
3	2	2	3 → 2 → 0
4	2	2	4 → 2 → 0
5	1	1	5 → 0

# Ejemplo Simple: SPT aprendido y comparación III



**Figura:** Visualización del ejemplo: los puntos representan los nodos de la red IoT, el nodo 0 (sumidero) se resalta en rojo y las aristas en verde muestran el SPT aprendido por Q-learning.

# Notación para complejidad (SPT)

Definamos:

- $N = |V|$  : número de nodos de la red.
- $\Delta$  : grado máximo (número máximo de vecinos por nodo).
- $M$  : número de grafos usados en el entrenamiento.
- $K$  : número de episodios por grafo.
- $L$  : longitud media del camino (en saltos) desde un nodo al sumidero durante el entrenamiento.

# Complejidades parciales

Actualización de Q-valores: en cada paso de un episodio:

$$O(|N(v)|) \leq O(\Delta),$$

para calcular  $Q_{\max}$  sobre los vecinos del siguiente nodo.

Episodio de entrenamiento: si el trayecto típico tiene longitud  $L$ :

$$O(L \cdot \Delta).$$

Entrenamiento total: sobre  $M$  grafos y  $K$  episodios por grafo:

$$O(M \cdot K \cdot L \cdot \Delta).$$

Construcción del SPT en prueba: para cada nodo  $v$  se sigue una ruta hacia  $v_0$  evitando ciclos:

$$O(L \cdot \Delta) \text{ por nodo} \Rightarrow O(N \cdot L \cdot \Delta).$$

Comunicación: cada actualización requiere solo información de vecinos de un salto, con sobrecarga  $O(1)$  por paso.

# Complejidad total y comparación

- **Entrenamiento (offline):**

$$T_{\text{train}}(N) = O(M \cdot K \cdot L \cdot \Delta).$$

- **Construcción del SPT (online):**

$$T_{\text{test}}(N) = O(N \cdot L \cdot \Delta),$$

ejecutado de forma *distribuida* en todos los nodos.

- **Comparación con algoritmos clásicos:**

- Dijkstra centralizado:  $O(N^2)$  (o  $O((N + E) \log N)$ ) más el coste de recolectar la topología completa.
  - SPT con Q-learning distribuido: coste aproximadamente lineal en  $N$  por ejecución online, sin recopilar el grafo completo.
- **Conclusión parcial:** el esfuerzo de entrenamiento se paga una sola vez; luego el enrutamiento es ligero y adecuado para redes IoT grandes y dinámicas.

# Conclusiones generales

- **Algoritmo MST (Buildings 2024):**

- Integra la asignación vivienda–esquina con un MST sobre esquinas.
- Combina heurísticas Greedy, MST clásico y MILP para refinar la solución.
- Es adecuado para planificación urbana estática con restricciones de capacidad.

- **Algoritmo SPT (Q-learning distribuido):**

- Ataca el problema de enrutamiento en redes IoT con recursos limitados.
- Cada nodo aprende localmente su siguiente salto hacia el sumidero, formando un SPT casi óptimo.
- No requiere conocimiento global ni recomputar rutas de forma centralizada.

# Conclusiones específicas del SPT con Q-learning

- El marco propuesto logra **altas tasas de precisión** en la reconstrucción del árbol de caminos más cortos:
  - Errores típicos de 1–2 saltos en redes pequeñas.
  - Exactitud muy cercana al 100 % en redes de tamaño medio y grande reportadas en el paper.
- El algoritmo es:
  - **Distribuido y escalable**, con baja sobrecarga de comunicación.
  - **Robusto** frente a cambios de topología y fallas de nodos.
  - **Generalizable** entre distintos tamaños de red, reduciendo la necesidad de reentrenamiento.
- **Líneas futuras:**
  - Extender a *deep reinforcement learning* para topologías aún más grandes.
  - Incorporar métricas adicionales (energía residual, calidad de enlace, retraso).
  - Validar el enfoque en plataformas IoT reales y escenarios urbanos inteligentes.