Analisi beyond worst-case: instance optimality di algoritmi per cammini minimi.

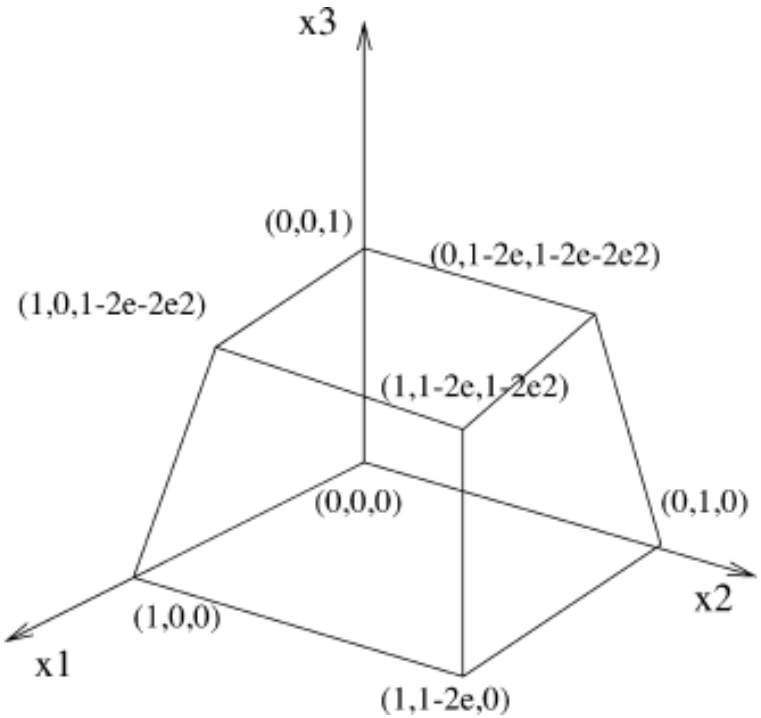
Daniele Bignardi
Mentore: Prof. Geppino Pucci

Università di Padova

Indice dei contenuti

Limiti della worst case analysis: simplesso

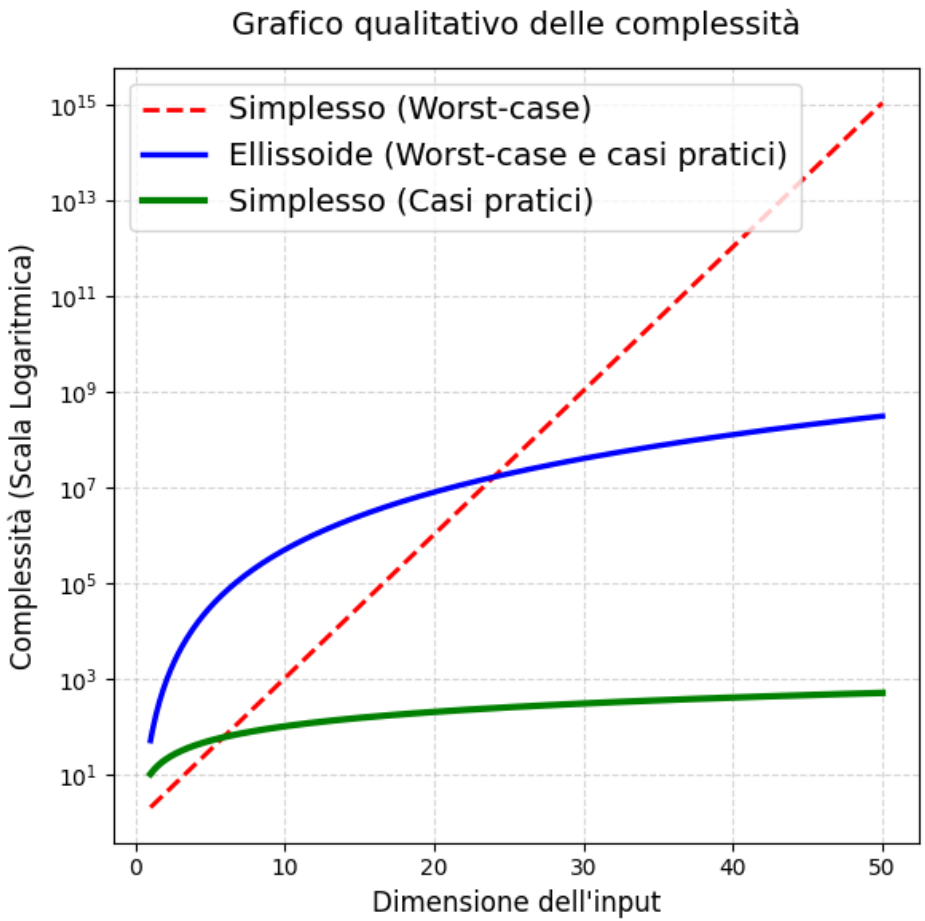
- **Simplesso:** algoritmo per la programmazione lineare. Procede esplorando i vertici del poliedro.
- **Caso peggiore (Klee e Minty, 1972):** complessità esponenziale.
- **Pratica:** tempi di esecuzione estremamente rapidi.
- **Problema:** l'istanza worst-case è artificiale e poco significativa.



Fonte: Encyclopedia of Optimization, Springer

Simplesso vs Ellissoide

- **Ellissoide:** primo algoritmo con complessità **polinomiale** al caso peggiore (1979).
- **Teoria:** la worst-case analysis "suggerisce" di usarlo al posto del Simplesso.
- **Pratica:** il Simplesso è più efficiente.
- **Problema:** suggerimento sbagliato su quale algoritmo adottare.



Oltre il Caso Peggiore

Esistono numerose alternative alla Worst-Case Analysis, nate con scopi diversi (*Roughgarden, 2021*).

- **Il nostro focus:** risolvere il problema del confronto tra algoritmi.
- **La soluzione:** l'*Instance Optimality*.
- **Il vantaggio chiave:** garanzie forti sull'algoritmo migliore.

La valutazione dell'algoritmo: la funzione di costo

- Dato un problema computazionale Π, chiamiamo:
  - $\mathcal{I}$: l'insieme delle istanze di Π.
  - $\mathbb{A}$: l'insieme degli algoritmi corretti per Π.

- Possiamo ora definire formalmente una **funzione di costo**:

$$\text{cost} : \mathbb{A} \times \mathcal{I} \to \mathbb{R}_{\geq 0}$$

  che valuta le performance di un algoritmo su un'istanza.

- **Esempi:**
  - Complessità computazionale
  - Numero di query
  - Fattore di approssimazione

Il Confronto: la Dominanza

Fissata una funzione di costo, possiamo definire formalmente quando un algoritmo è superiore a un altro:

Definizione di Dominanza

Un algoritmo $\mathcal{A} \in \mathbb{A}$ **domina** $\mathcal{B} \in \mathbb{A}$ se, per ogni istanza $z \in \mathcal{I}$:

$$\text{cost}(\mathcal{A}, z) \leq \text{cost}(\mathcal{B}, z)$$

- **Attenzione:** la dominanza non è una proprietà assoluta, ma **dipende strettamente** dalla funzione di costo scelta.

Instance Optimality

Instance Optimality (Versione ideale)

Un algoritmo $\mathcal{A}$ per un problema Π è **instance optimal** se domina ogni algoritmo $\mathcal{B}$ per Π.

Instance Optimality (Versione finale)

Un algoritmo $\mathcal{A}$ per un problema Π è **instance optimal con approssimazione c rispetto a un insieme $\mathcal{C}$ di algoritmi** per Π se $\forall \mathcal{B} \in \mathcal{C}$ e $\forall z \in \mathcal{I}$:

$$\text{cost}(\mathcal{A}, z) \leq c \cdot \text{cost}(\mathcal{B}, z)$$

dove $c \geq 1$ è una costante indipendente da $\mathcal{B}$ e z.

I limiti dell'Instance Optimality

L'instance optimality è una garanzia **estremamente forte**: per molti problemi e modelli di costo, **non esistono** algoritmi instance optimal.

L'esempio dell'ordinamento (nel comparison model):

- Per ogni istanza esiste un algoritmo che ordina in $\mathcal{O}(n)$ confronti.
- Nessun algoritmo può ordinare in $\mathcal{O}(n)$ tutte le istanze.

Le soluzioni

Per rendere il modello applicabile, è possibile:

- Utilizzare garanzie **parametrizzate** ($\text{cost}(\mathcal{A}, z) \leq f(k) \cdot \text{cost}(\mathcal{B}, z)$).
- Introdurre **restrizioni** sulla classe delle **istanze** $\mathcal{I}$.
- Cambiare la classe $\mathcal{C}$ di algoritmi.

Il Problema dello Shortest *st*-Path

Input: Un multigrafo $G = (V, A)$ diretto, un nodo sorgente s e un nodo destinazione t.

Pesi e Distanze

Ad ogni arco è assegnato un peso non negativo:

$$\ell : A \to \mathbb{R}_{\geq 0}$$

$d(u, v) =$ costo del cammino minimo da u a v.

Output: Un cammino minimo da s a t.

Cosa vogliamo misurare?

Utilizziamo il seguente:

Modello di Accesso e di Costo (*Query Model*)

L'accesso agli archi avviene unicamente tramite l'oracolo:

```
Outedge(i, j)
```

Restituisce il j-esimo arco uscente dal nodo i (secondo la lista di incidenza).

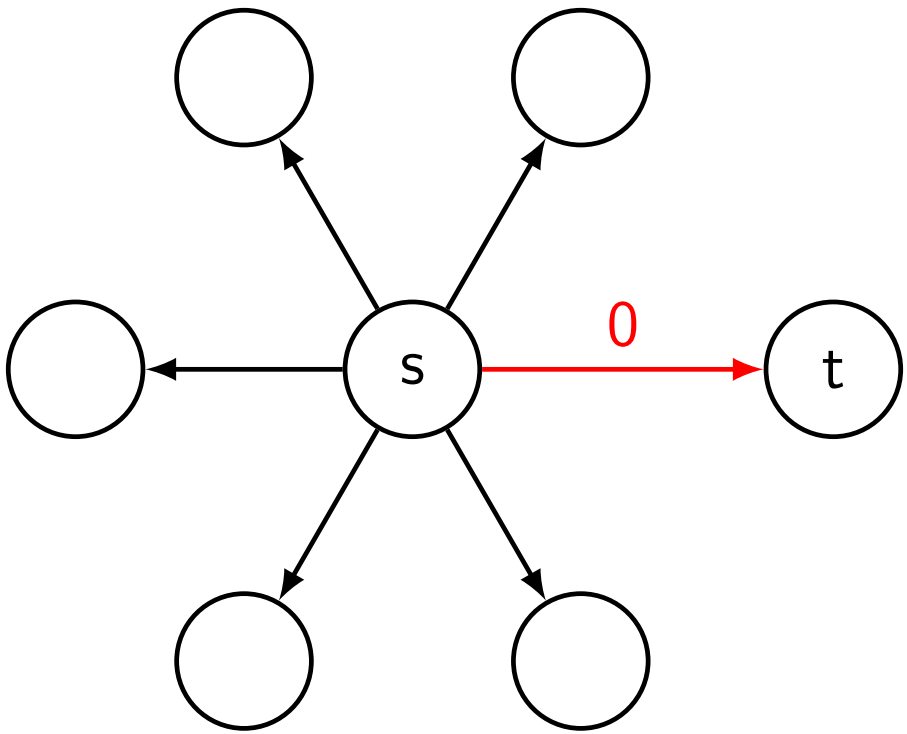
Costo: $\text{cost}(\mathcal{A}, z) =$ numero totale di query effettuate da $\mathcal{A}$ su z.

La Restrizione per l'Instance Optimality

Pesi Strettamente Positivi ($\ell(e) > 0$)

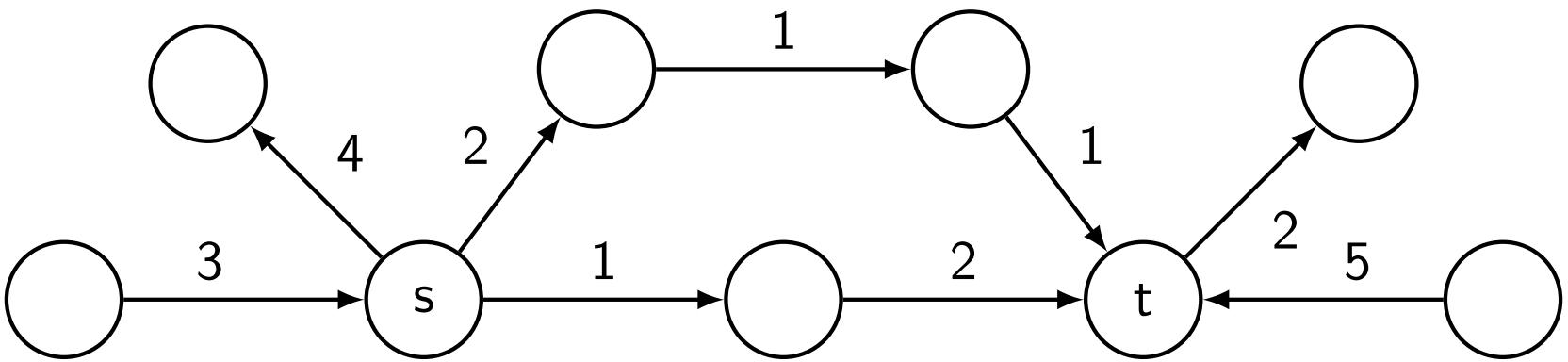
Richiediamo che ogni arco abbia un costo strettamente maggiore di zero.

Perché questa restrizione è vitale per l'Instance Optimality?



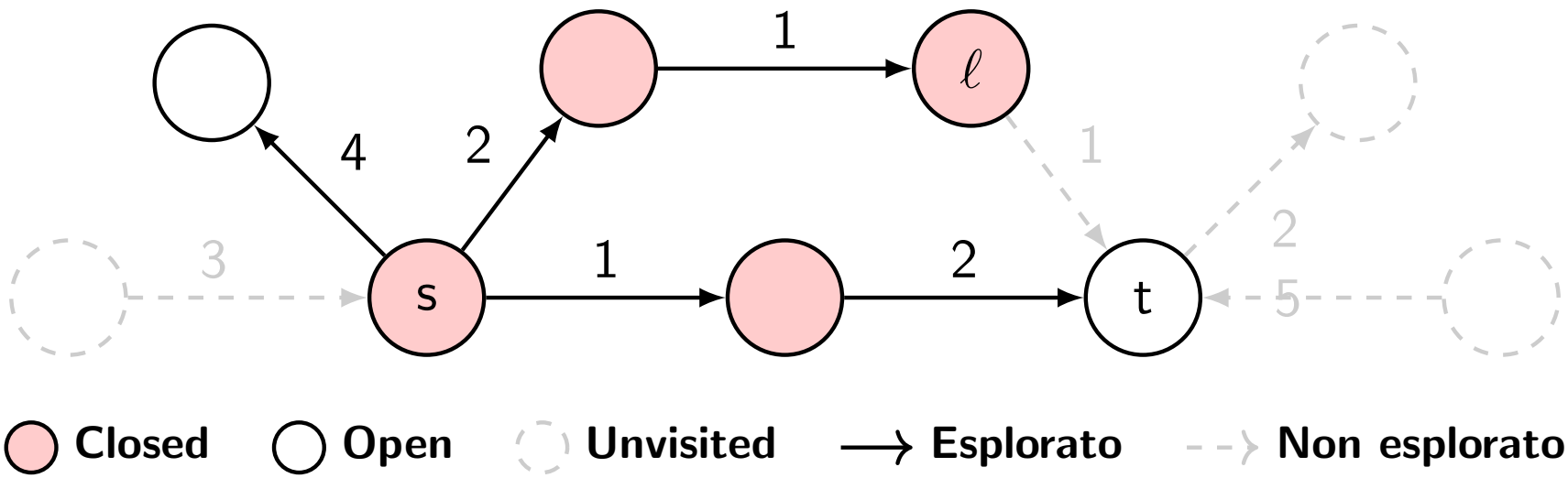
Algoritmo di Dijkstra unidirezionale

- **Stati:** Ogni nodo è *Unvisited* o *Open* (visto) o *Closed* (distanza da s certificata).
- **Inizializzazione**: s è *Open*; gli altri nodi sono *Unvisited*
- **Iterativamente:**
  - Si estrae il nodo *Open* u più vicino a s e lo si pone *Closed*.
  - Si esplorano gli archi uscenti da u tramite `Outedge` per scoprire i vicini.



Algoritmo di Dijkstra unidirezionale

- **Terminazione**: appena si chiude un nodo v a distanza $d(s, v) = \hat{d}(s, t)$



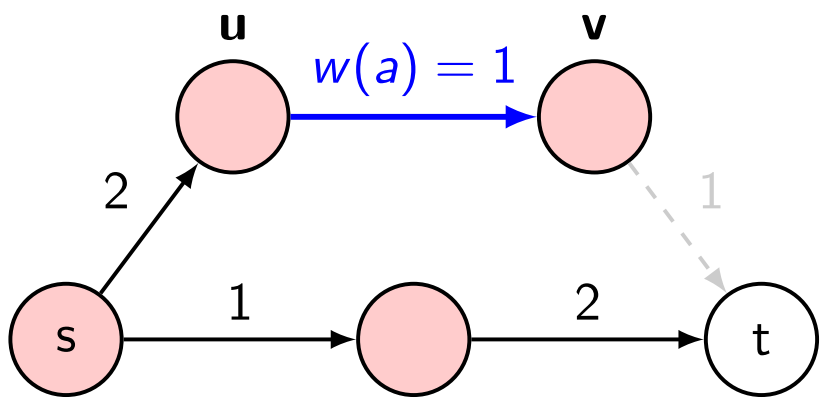
Dijkstra Unidirezionale è Instance Optimal

Teorema di Ottimalità (*Haeupler et al., SOSA 2025*)

Dijkstra unidirezionale è **Instance Optimal** con costante $c = 1$ rispetto alla classe $\mathcal{C}$ di tutti gli algoritmi deterministici corretti nel Query Model definito prima.

Sketch della dimostrazione (1/2)

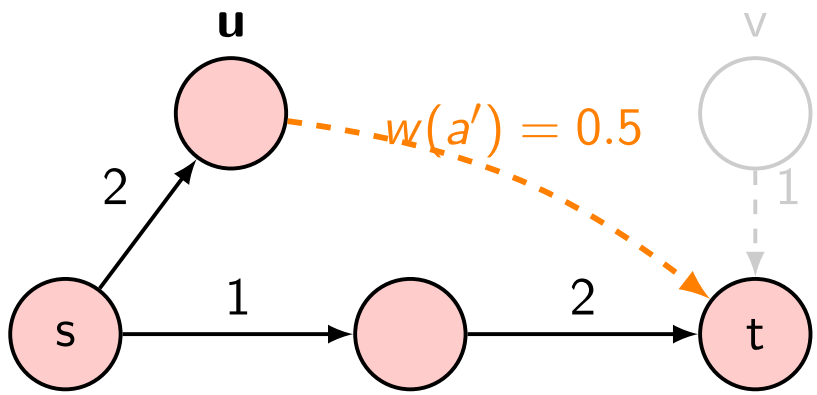
- Sia $a = uv$ un qualsiasi arco esplorato dall'algoritmo di Dijkstra.
- Grazie alla condizione di terminazione si ha $d_G(s, u) < d_G(s, t)$.
- Sia $\mathcal{B}$ un algoritmo che **non fa la query** sull'arco a.



Grafo originale G

Sketch della dimostrazione (2/2)

- Costruiamo un grafo G' sostituendo $a = uv$ con $a' = ut$ con peso $w(a') < d_G(s,t) - d_G(s,u)$
- G' ha **shortest path** di costo $d_{G'}(s,t) = d_G(s,u) + w(a') < d_G(s,t)$.
- L'unica query che distingue G da G' è quella su a.
- $\mathcal{B}$, non avendo interrogato a, darà la stessa soluzione per G e G'.
  $\implies$ $\mathcal{B}$ **non corretto**.



Grafo modificato G'

Un modello di query più generale

Espandiamo le capacità del nostro Oracolo.

Modello di Accesso e di Costo (*Query Model*)

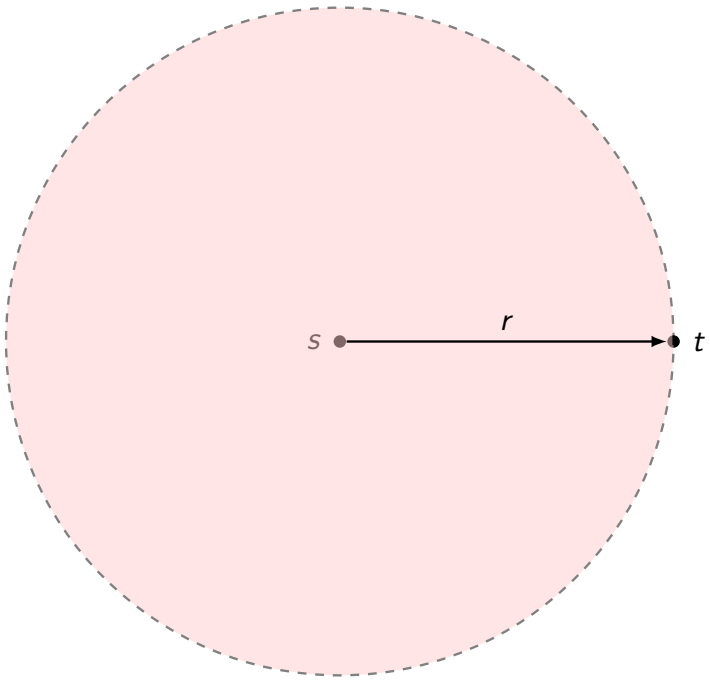
L'algoritmo può ora interrogare il grafo tramite:

1. `Outedge(i, j)`: restituisce il j-esimo arco uscente da i.
2. `Inedge(i, j)`: restituisce il j-esimo arco entrante in i.

Costo: $\text{cost}(\mathcal{A}, z) =$ numero totale di query effettuate da $\mathcal{A}$ su z.

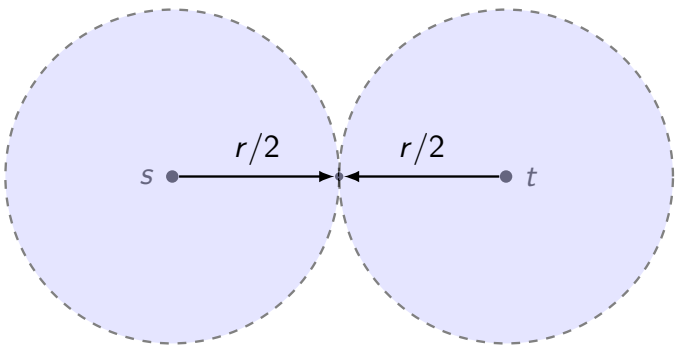
Perché la ricerca Bidirezionale?

L'intuizione geometrica: Espandiamo due palle invece che una.



Unidirezionale

Area $\propto r^2$

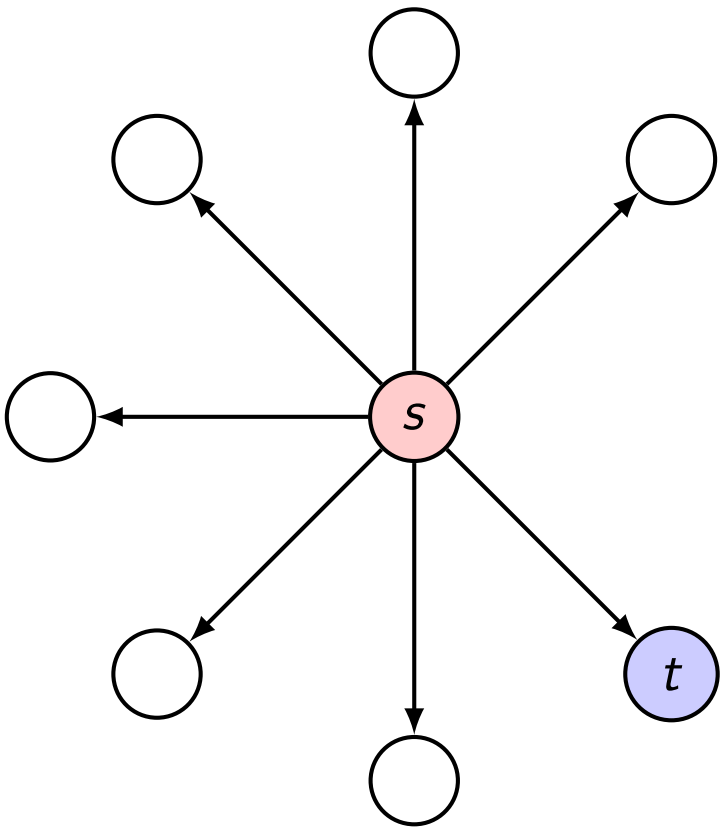


Bidirezionale

Area $\propto 2(r/2)^2 = \frac{r^2}{2}$

Unidirezionale vs Bidirezionale: Il Grafo a Stella

Ci sono casi in cui la ricerca bidirezionale batte quella unidirezionale di un fattore **non costante**.



- **Unidirezionale (Da s):** Esplora tutti gli archi uscenti da s.
  $\Longrightarrow$ **Costo:** $O(|V|)$ **query.**
- **Bidirezionale (Da t):** Interroga gli archi entranti in t (`Inedge`) e vede subito s.
  $\Longrightarrow$ **Costo:** $O(1)$ **query.**

Algoritmo di Dijkstra Bidirezionale: Le 3 Regole

Esegue due ricerche di Dijkstra simultaneamente: *forward* (da s) e *backward* (da t).

- **1. Espansione**
  Si esplora esattamente **un arco** in avanti, e poi **un arco** all'indietro.

- **2. Aggiornamento del cammino**
  Se un arco esplorato incide su un nodo già *Open* nella direzione opposta, si valuta l'aggiornamento del miglior cammino provvisorio μ.

- **3. Condizione di terminazione**
  Siano u_f e u_b gli ultimi nodi chiusi nelle rispettive direzioni.
  L'algoritmo termina appena:

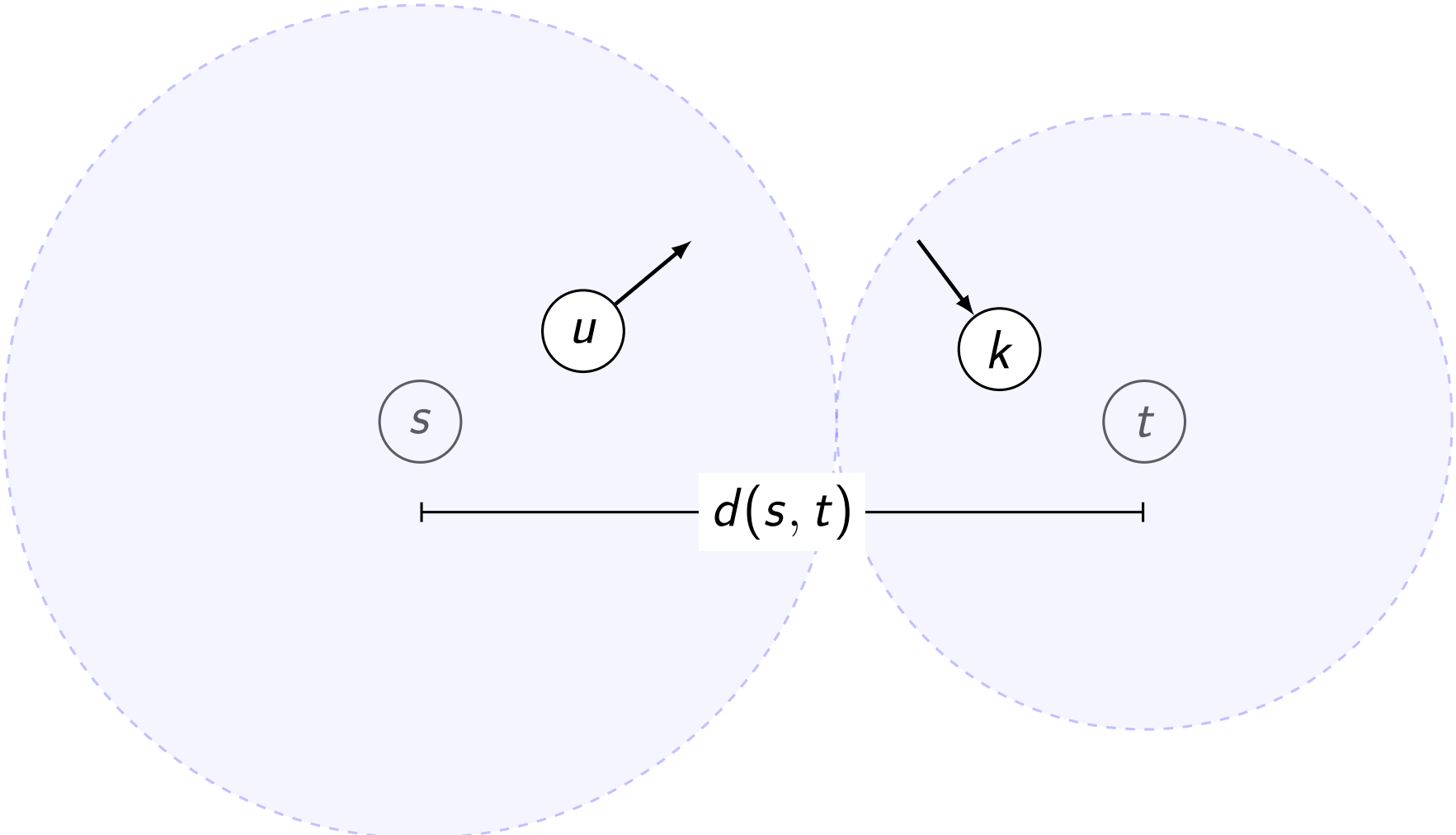
$$d(s, u_f) + d(u_b, t) \geq \mu$$

Dijkstra Bidirezionale è Instance Optimal

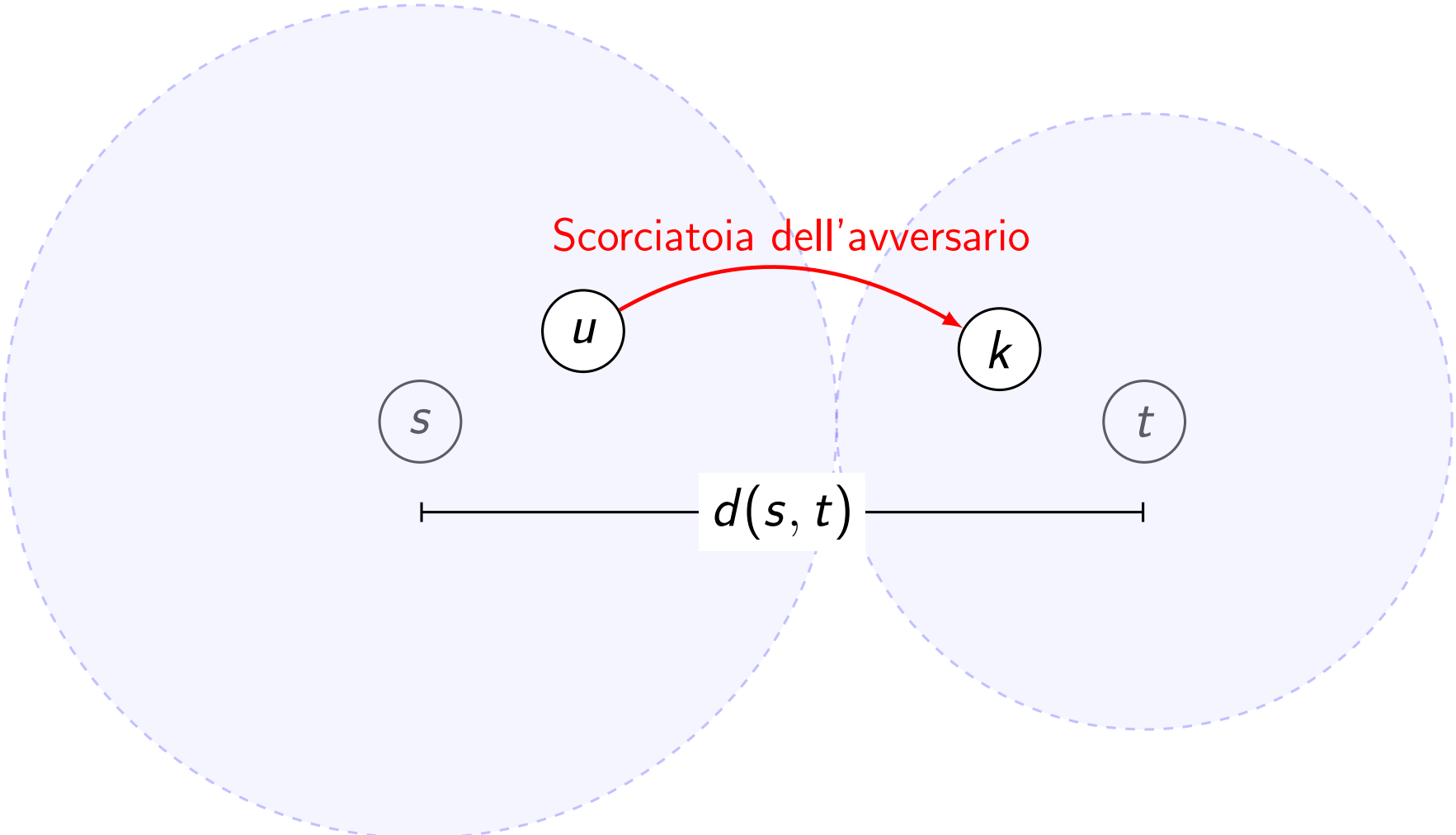
Teorema di Ottimalità (*Haeupler et al., SOSA 2025*)

Considerando come costo il numero di query effettuate, Dijkstra bidirezionale è **Instance Optimal** con costante $c = 2$ rispetto alla classe $\mathcal{C}$ degli algoritmi deterministici corretti.

Intuizione



Intuizione



Lavori correlati e Sviluppi Futuri

- **Grafi non pesati: BFS Bidirezionale** (*Haeupler et al., SOSA 2025*)
  - **Problema:** shortest *st*-path.
  - **Risultato:** quasi instance optimal. Nessun algoritmo può fare meglio.

- **Universal Optimality of Dijkstra** (*Haeupler et al., FOCS 2024*).
  - **Problema:** ordinare i vertici per distanza da una sorgente s.
  - **Risultato:** Dijkstra unidirezionale è universal optimal (heap ad-hoc).

- **A* e ricerca con euristiche**.

Bibliografia


V. Klee, G. J. Minty.
How good is the simplex algorithm?.
Inequalities III, 1972.


T. Roughgarden (Ed.).
Beyond the Worst-Case Analysis of Algorithms.
Cambridge University Press, 2021.


B. Haeupler, R. Hladík, V. Rozhoň, R. Tarjan, J. Tětek.
Bidirectional Dijkstra's Algorithm is Instance-Optimal.
SOSA, 2025.


B. Haeupler, R. Hladík, V. Rozhoň, R. Tarjan, J. Tětek.
Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps.
FOCS, 2024.


N. Sturtevant, A. Felner.
A Brief History and Recent Achievements in Bidirectional Search.
AAAI, 2018.

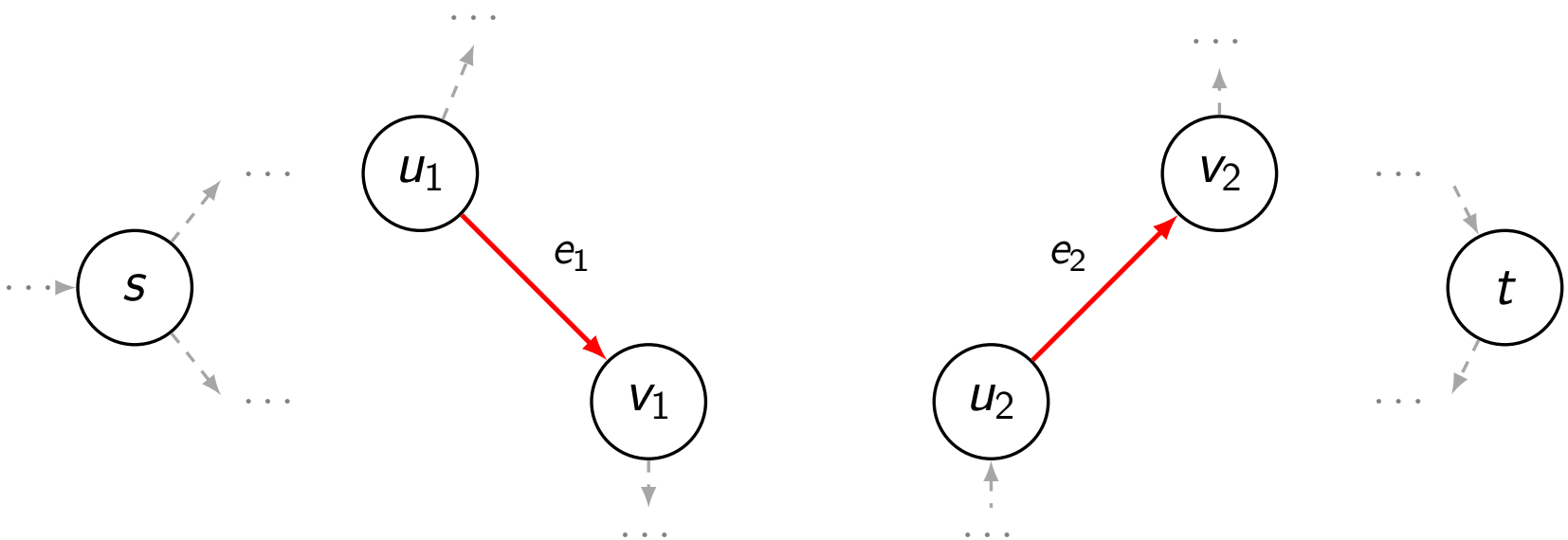

T. Roughgarden.
Lecture Notes for CS264: Beyond Worst-Case Analysis.
Stanford University, Autunno 2014.

Frame Title

Sketch della dimostrazione: le query necessarie (1/2)

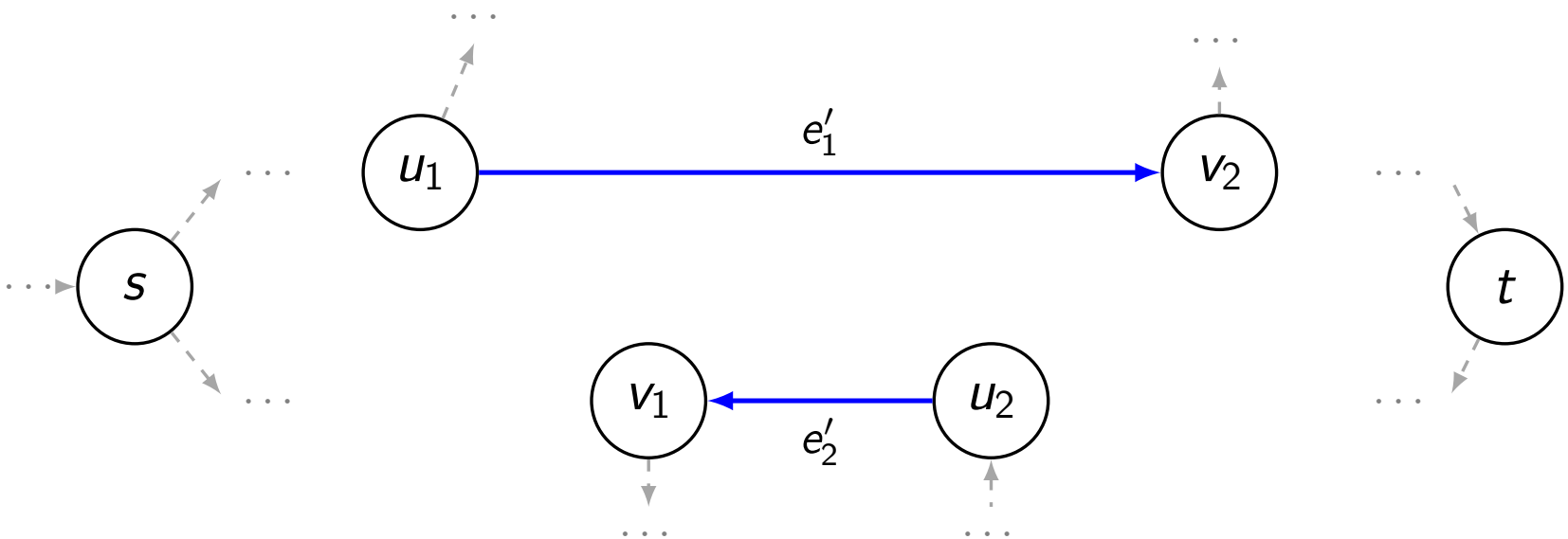
LEMMA: Siano u_1, v_2 tali che $d_G(s, u_1) + d_G(v_2, t) < d_G(s, t)$. Un algoritmo $\mathcal{B}$ che **non esplora** un arco $e_1 = u_1 v_1$ uscente da u_1 e un arco $e_2 = u_2 v_2$ entrante in v_2 **non è corretto**.

- Supponiamo che $\mathcal{B}$ termini senza esplorare e_1 ed e_2.



Sketch della dimostrazione: le query necessarie (2/2)

- **Costruzione di G':** Sostituiamo gli archi ignorati con $e'_1 = u_1 v_2$ $e'_2 = u_2 v_1$, **preservando i gradi**. Impostiamo il peso $\ell(e'_1) < d_G(s,t) - d_G(s,u_1) - d_G(v_2,t)$.
- In G', il cammino tramite e'_1 costa meno di $d_G(s,t)$. $\mathcal{B}$, essendo **ignaro** dello scambio, restituirà la stessa soluzione per G e per G' $\implies$ $\mathcal{B}$ **non è corretto**.



Sketch della dimostrazione: terminazione e bilanciamento

- **Terminazione Ottima:**
  Siano u_f, u_b i nodi correntemente **processati** (avanti/indietro).
  Si esplorano archi **solo** finché $d(s, u_f) + d(u_b, t) < d(s, t)$.
  L'algoritmo termina senza fare query superflue.

- **Bilanciamento (Alternanza stretta):**
  Siano E_f ed E_b gli insiemi di archi esplorati. Si ha sempre:

$$|E_b| \leq |E_f| \leq |E_b| + 1$$

- **CONCLUSIONE ($c = 2$):**
  Per il Lemma, **ogni** algoritmo corretto deve esplorare $\geq |E_b|$ archi.
  La Bidirezionale ne esplora $|E_f| + |E_b| \approx 2|E_b|$.

  $\Longrightarrow$ **Dijkstra Bidirezionale è Instance Optimal con $c = 2$.**